DB2 Server for VSE & VM

Database Services Utility

Version 7 Release 5

SC09-2983-03

DB2 Server for VSE & VM

Database Services Utility

Version 7 Release 5

SC09-2983-03

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 255.

Contents

About This Manual

This manual is intended to help DB2 Server for VSE & VM users use the Database Services (DBS) utility; it contains descriptions of the tasks connected with the use of the Database Services Utility in a Virtual System Extended/Enterprise Systems Architecture (VSE/ESA™) environment and in a Virtual Machine/Enterprise Systems Architecture (VM/ESA®) environment. It also contains a reference section for database users or application programmers who need more information about the Database Services Utility. This manual follows the convention that VM refers to the VM/ESA system unless otherwise specified and VSE refers to the VSE/ESA system unless otherwise specified.

Who Should Use This Manual

This manual is a guide and reference for users of the Database Services Utility. Any user of the DB2 Server for VSE & VM product is a potential user of this manual; it is, however, particularly useful to database users who want to use batch processing in their database operations.

How to Use This Manual

This manual describes and explains what the Database Services Utility is, how it functions, and when to use it.

Utilization

This manual contains two parts. Each chapter in Part 1 has a task area, for example, *loading data* or *interpreting output*. Within each task area, member subtasks are grouped according to their importance or in order of performance.

To use the user-guide part of this manual, select the chapter that corresponds to the general type of Database Services Utility activity that you want to perform. Within that chapter, find the procedure that provides specific instructions for the subtask that you want. Supplementary information, alternative procedures, and examples are in boxes within the text. Perform the procedure's numbered steps and refer to the supplementary text within frames, figures, and examples as necessary.

To use the reference part of this manual, find the general or specific topic of interest in the table of contents or index and refer directly to its listed page or pages.

Organization

The **Summary of Changes** summarizes the changes made to DB2 Server for VSE & VM Version 7 Release 5.

Part 1 contains the following chapters:

Chapter 1, "Getting Started," on page 3,
introduces the Database Services Utility and explains its use. It also provides an example of a Database Services Utility job.

Chapter 2, "Loading Data with the Database Services Utility," on page 27,
shows how to load tables with data specified by the user.

Chapter 3, "Unloading Data with the Database Services Utility," on page 51,
shows how to unload tables in a format specified by the user or in a format provided by the Database Services Utility.

Chapter 4, "Reloading Data with the Database Services Utility," on page 71,
shows how to reload tables with data in a format provided by the Database Services Utility.

Chapter 5, "Unloading and Reloading Packages with the Database Services Utility," on page 83,
shows how to unload and reload packages.

Chapter 6, "Interpreting the Output of the Database Services Utility," on page 95,
describes VSE report output or VM message-file output, how to read it, and how to understand it.

Part 2 contains the following chapters:

Chapter 7, "Using the Database Services Utility from Application Programs," on page 105,
contains rules for naming objects, lists reserved words, describes the procedures required to initiate Database Services Utility processing from application programs, and describes how to use the Database Services Utility application program interface.

Chapter 8, "Command Reference," on page 135,
describes command processing and contains complete descriptions of all Database Services Utility commands.

Chapter 9, "Error Handling and Debugging," on page 223,
describes the processing undertaken by the Database Services Utility whenever errors are encountered and supplies information on the processing of debug-type errors.

Chapter 10, "Improving Performance," on page 227,
describes measures that could help improve the Database Services Utility's processing speed or efficiency.

Appendix A, "Sample Tables," on page 235,
shows the contents of the sample tables supplied with the DB2 Server for VSE & VM product.

Appendix B, "FILEDEF Command Syntax and Notes," on page 249,
presents a syntax diagram and usage notes on the Conversational Monitor System (CMS) FILEDEF command as it relates to the Database Services Utility.

The **Bibliography** lists the publications that are related to this book.

Components of the Relational Database Management System

Figure 1 on page ix depicts a typical configuration with one database and two users.

Figure 2 on page x depicts a typical configuration with one database, one batch partition user, and a CICS® partition with several interactive users.

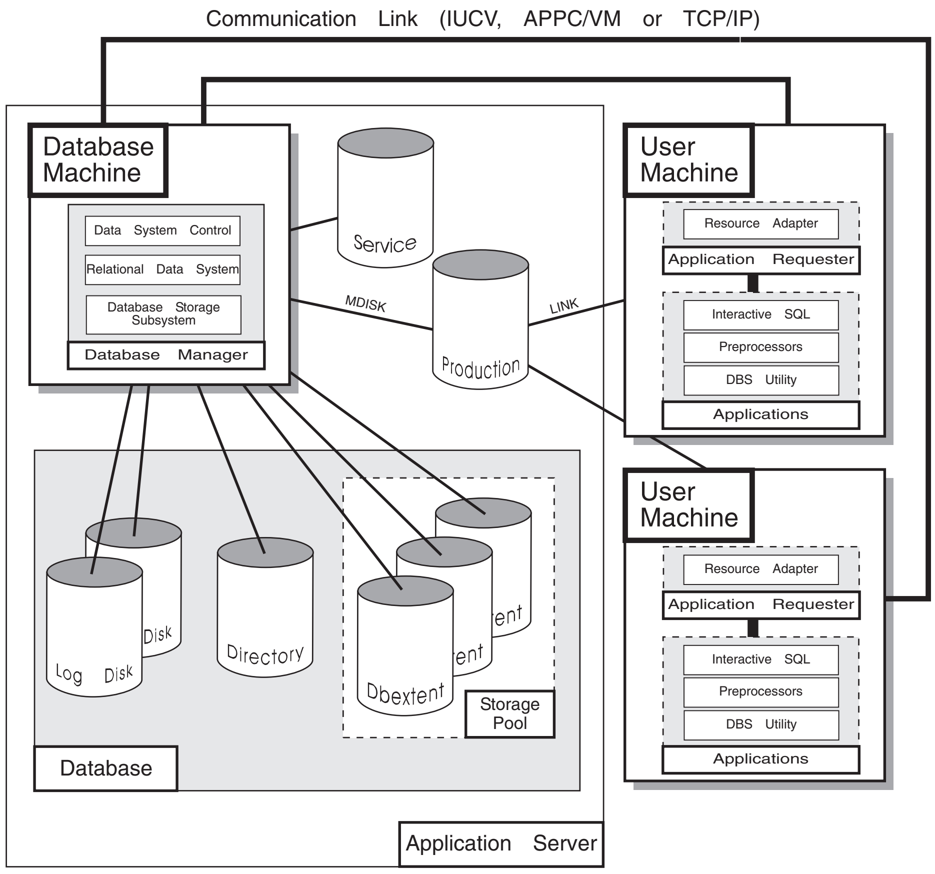


Figure 1. Basic Components of the RDBMS in VM/ESA

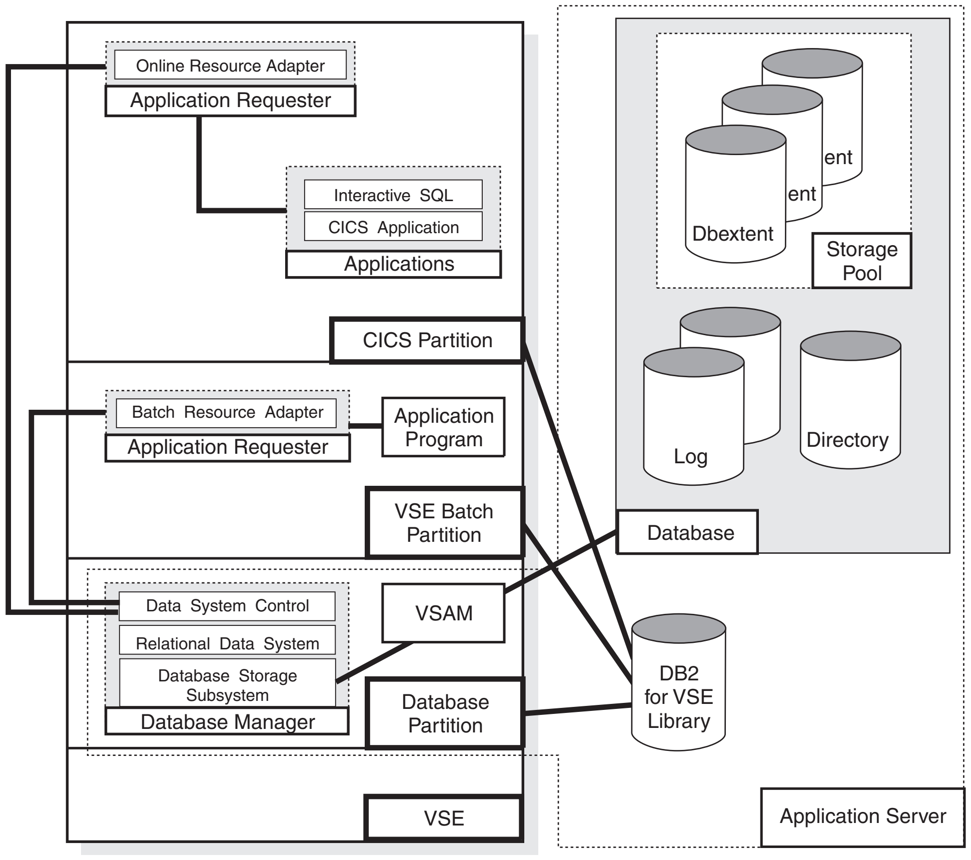


Figure 2. Basic Components of the RDBMS in VSE/ESA

The **database** is composed of :

- A collection of data contained in one or more *storage pools*, each of which in turn is composed of one or more *database extents (dbextents).* A dbextent is a VM minidisk or a VSE VSAM cluster.
- A *directory* that identifies data locations in the storage pools. There is only one directory per database.
- A *log* that contains a record of operations performed on the database. A database can have either one or two logs.

The **database manager** is the program that provides access to the data in the database. In VM it is loaded into the database virtual machine from the production disk. In VSE it is loaded into the database partition from the DB2 Server for VSE library.

The **application server** is the facility that responds to requests for information from and updates to the database. It is composed of the database and the database manager.

The **application requester** is the facility that transforms a request from an application into a form suitable for communication with an application server.

Prerequisites

Knowledge

This manual assumes the following:

- You have read the manuals listed under the heading “Publications” that follows and understand the way the database manager works.
- You have a working knowledge of the IBM VSE/ESA environment and are acquainted with job control language (JCL).
- You have a working knowledge of the VM Conversational Monitor System and are acquainted with CMS commands.
- You know basic terms and concepts used in the *DB2 Server for VSE System Administration* and *DB2 Server for VM System Administration* manuals.
- You have access to the manuals listed in the Bibliography.

Publications

This manual assumes that you are familiar with the information in the following manuals:
DB2 Server for VSE & VM Interactive SQL Guide and Reference, SC09-2990
DB2 Server for VSE & VM Overview, GC09-2995
DB2 Server for VSE System Administration, SC09-2981
DB2 Server for VM System Administration, SC09-2980.

Highlighting Conventions

This manual observes the following text highlighting conventions:

Convention	**Meaning**
Italics	Italic type denotes command variables, parameter values and their symbolic equivalents, titles of stand-alone documents, and strings of characters referred to as such.
Boldface	Bold type is used for emphasis or for an important term that is being defined.
`Monospace Type`	Monospace type indicates material that is entered at a display station, displayed on a screen, coded, or printed on a computer printing device.
ALL CAPS	Capital letters indicate keytop nomenclature, for example, PF*n*, ENTER, CLEAR, INSERT, and DELETE. In addition, the following situations call for all caps: • Acronyms and other all-cap abbreviations • Names of programs and other coded entities • Names of files, tables, libraries, logs, and so forth • Command, statement, and parameter names or constants • Keyword and option names • Data area and storage names.
“Quotation Marks”	Quotation marks (double) enclose the headings of parts, chapters, and lesser sections of stand-alone documents when they are referenced; to designate specific, lengthy passages of text (at least a sentence in length); and to denote figurative and other special usage, such as jargon.
As Displayed	Panel names, menu titles, and other display headers are shown in uppercase or mixed case, as displayed.

Syntax Notation Conventions

Throughout this manual, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right and from top to bottom, following the path of the line.

  The ►►── symbol indicates the beginning of a statement or command.

  The ──► symbol indicates that the statement syntax is continued on the next line.

  The ►── symbol indicates that a statement is continued from the previous line.

  The ──►◄ symbol indicates the end of a statement.

  Diagrams of syntactical units that are not complete statements start with the ►── symbol and end with the ──► symbol.
- Some SQL statements, Interactive SQL (ISQL) commands, or database services utility (DBS Utility) commands can stand alone. For example:

```
►►──SAVE──────────────────────────────────────────────────────────►◄
```

  Others must be followed by one or more keywords or variables. For example:

```
►►──SET AUTOCOMMIT OFF────────────────────────────────────────────►◄
```

- Keywords may have parameters associated with them which represent user-supplied names or values. These names or values can be specified as either constants or as user-defined variables called *host_variables* (*host_variables* can only be used in programs).

```
►►──DROP SYNONYM──synonym─────────────────────────────────────────►◄
```

- Keywords appear in either uppercase (for example, SAVE) or mixed case (for example, CHARacter). All uppercase characters in keywords must be present; you can omit those in lowercase.
- Parameters appear in lowercase and in italics (for example, *synonym*).
- If such symbols as punctuation marks, parentheses, or arithmetic operators are shown, you must use them as indicated by the syntax diagram.
- All items (parameters and keywords) must be separated by one or more blanks.
- Required items appear on the same horizontal line (the main path). For example, the parameter *integer* is a required item in the following command:

```
►►──SHOW DBSPACE──integer──────────────────────────────────────────────►◄
```

This command might appear as:

```
SHOW DBSPACE 1
```

- Optional items appear below the main path. For example:

```
►►──CREATE──┬────────┬──INDEX──────────────────────────────────────────►◄
            └─UNIQUE─┘
```

This statement could appear as either:

```
CREATE INDEX
```

or

```
CREATE UNIQUE INDEX
```

- If you can choose from two or more items, they appear vertically in a stack.

  If you must choose one of the items, one item appears on the main path. For example:

```
►►──SHOW LOCK DBSPACE──┬─ALL─────┬─────────────────────────────────────►◄
                       └─integer─┘
```

Here, the command could be either:

```
SHOW LOCK DBSPACE ALL
```

or

```
SHOW LOCK DBSPACE 1
```

If choosing one of the items is optional, the entire stack appears below the main path. For example:

```
►►──BACKWARD──┬─────────┬──────────────────────────────────────────────►◄
              ├─integer─┤
              └─MAX─────┘
```

Here, the command could be:

```
BACKWARD
```

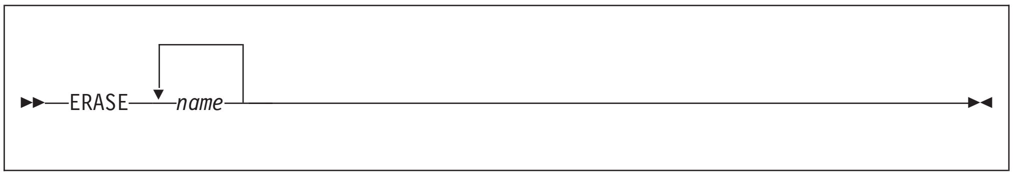
or

```
BACKWARD 2
```

or

```
BACKWARD MAX
```

- The repeat symbol indicates that an item can be repeated. For example:



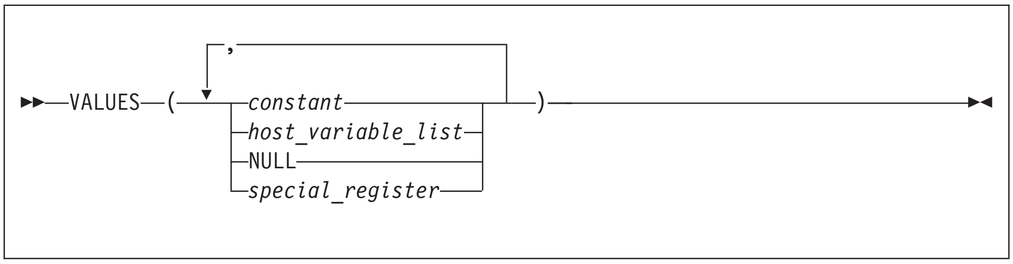
  This statement could appear as:

  ```
  ERASE NAME1
  ```

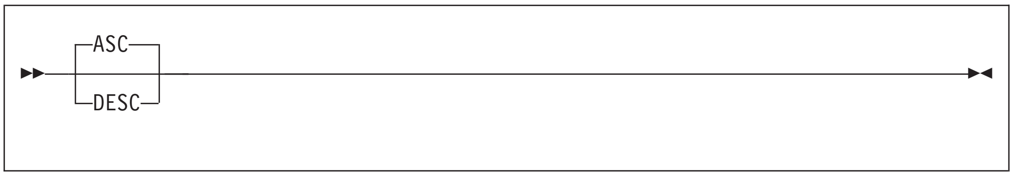
  or

  ```
  ERASE NAME1 NAME2
  ```

  A repeat symbol above a stack indicates that you can make more than one choice from the stacked items, or repeat a choice. For example:

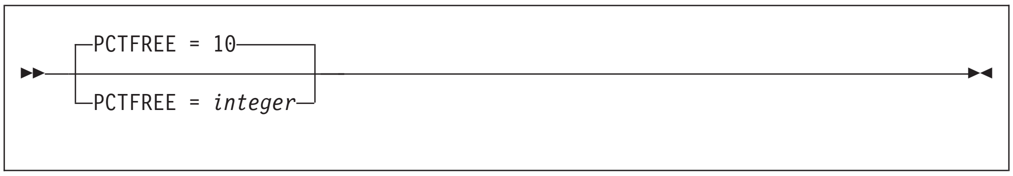


- If an item is above the main line, it represents a default, which means that it will be used if no other item is specified. In the following example, the ASC keyword appears above the line in a stack with DESC. If neither of these values is specified, the command would be processed with option ASC.

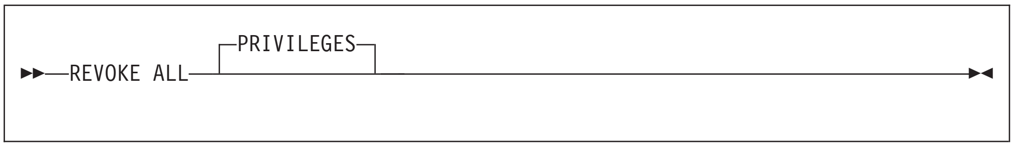


- When an optional keyword is followed on the same path by an optional default parameter, the default parameter is assumed if the keyword is not entered. However, if this keyword is entered, one of its associated optional parameters must also be specified.

  In the following example, if you enter the optional keyword PCTFREE =, you also have to specify one of its associated optional parameters. If you do not enter PCTFREE =, the database manager will set it to the default value of 10.

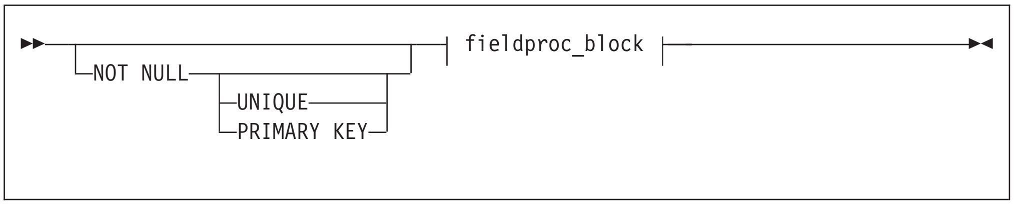


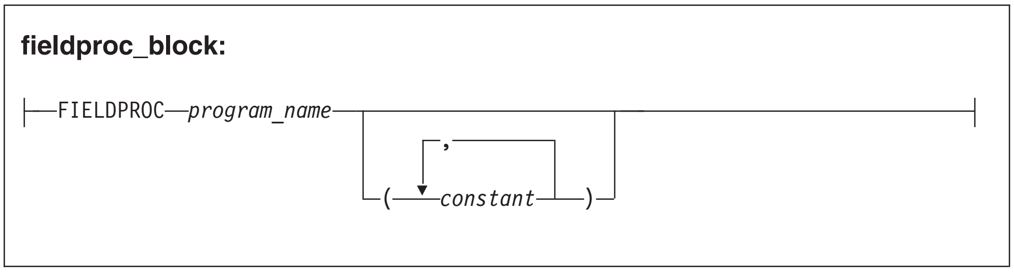
- Words that are only used for readability and have no effect on the execution of the statement are shown as a single uppercase default. For example:



Here, specifying either REVOKE ALL or REVOKE ALL PRIVILEGES means the same thing.

- Sometimes a single parameter represents a fragment of syntax that is expanded below. In the following example, **fieldproc_block** is such a fragment and it is expanded following the syntax diagram containing it.





SQL Reserved Words

The following words are reserved in the SQL language. They cannot be used in SQL statements except for their defined meaning in the SQL syntax or as host variables, preceded by a colon.

In particular, they cannot be used as names for tables, indexes, columns, views, or dbspaces unless they are enclosed in double quotation marks (").

ACQUIRE
ADD
ALL
ALTER
AND
ANY
AS
ASC
AVG

BETWEEN
BY

CALL
CHAR
CHARACTER
COLUMN
COMMENT
COMMIT
CONCAT
CONNECT
COUNT
CREATE
CURRENT

DBA
DBSPACE
DELETE
DESC
DISTINCT
DOUBLE
DROP

EXCLUSIVE
EXECUTE
EXISTS
EXPLAIN

FIELDPROC
FOR
FROM

GRANT
GRAPHIC
GROUP

HAVING

IDENTIFIED
IN
INDEX
INSERT
INTO
IS

LIKE
LOCK
LONG

MAX
MIN
MODE

NAMED
NHEADER
NOT
NULL

OF
ON
OPTION
OR
ORDER

PACKAGE
PAGE
PAGES
PCTFREE
PCTINDEX
PRIVATE
PRIVILEGES
PROGRAM
PUBLIC

RESOURCE
REVOKE
ROLLBACK
ROW
RUN

SCHEDULE
SELECT
SET
SHARE
SOME
STATISTICS
STORPOOL
SUM
SYNONYM

TABLE
TO

UNION
UNIQUE
UPDATE
USER

VALUES
VIEW

WHERE
WITH
WORK

Summary of Changes

This is a summary of the technical changes to the DB2 Server for VSE & VM database management system for this edition of the book. Several manuals are affected by some or all of the changes discussed here. For your convenience, the changes made in this edition are identified in the text by a vertical bar (|) in the left margin. This edition may also include minor corrections and editorial changes that are not identified.

This summary does not list incompatibilities between releases of the DB2 Server for VSE & VM product; see either the *DB2 Server for VSE & VM SQL Reference*, *DB2 Server for VM System Administration*, or the *DB2 Server for VSE System Administration* manuals for a discussion of incompatibilities.

Summary of Changes for DB2 Version 7 Release 5

Version 7 Release 5 of the DB2 Server for VSE & VM database management system is intended to run on the Z/VM Version 5 Release 2 or later environment and on the Z/VSE(®) Version 3 Release 1 or later environment.

Enhancements, New Functions, and New Capabilities

The following have been added to DB2 Version 7 Release 5:

Explain Option on DBSU REBIND PACKAGE Command

This new functionality allows the EXPLAIN(YES/NO) option on REBIND PACKAGE command. If EXPLAIN(YES) is issued, then all four update tables (structure, plan, cost, reference) will be updated. If EXPLAIN(NO) is issued, then none of the four update tables will be updated.

For more information, see the following DB2 Server for VSE & VM documentation:

- *DB2 Server for VSE & VM Database Services Utility*
- *DB2 Server for VSE & VM Performance Tuning Handbook*
- *DB2 Server for VSE & VM Quick Reference*
- *DB2 Server for VSE & VM SQL Reference*

For Fetch only

This new functionality accepts the ″FOR FETCH ONLY″ clause after a cursor select statement. It causes a cursor to become read-only (no UPDATEs or DELETEs are permitted using this cursor). If a read-only cursor is referenced in an UPDATE or DELETE statement, SQLCODE -510 will be issued and the statement is not processed. In addition, under the SBLOCK preprocessor option, ″FOR FETCH ONLY″ forces blocking to be used on the read-only cursor regardless of whether there is a COMMIT. If there is no ″FOR FETCH ONLY″ clause, under SBLOCK, blocking would only be done if a COMMIT was absent.

For more information, see the following DB2 Server for VSE & VM documentation:

- *DB2 Server for VM Messages and Codes*
- *DB2 Server for VSE & VM Application Programming*
- *DB2 Server for VSE & VM Performance Tuning Handbook*
- *DB2 Server for VSE & VM Quick Reference*

- *DB2 Server for VSE & VM SQL Reference*

Application Message Formatter

This functionality provides an Application Programming Interface (API) that retrieves the descriptive text for an SQLCODE, given an SQLCA input parameter. The API will be available for Assembly, COBOL, C, PL/I and FORTRAN.

In DB2 for VM and DB2 for VSE Online, the user may specify the language of the returned text. The languages supported by DB2 for VSE/VM are American English (AMENG), uppercase English (UCENG), German (GER), French (FRANC) and Japanese (KANJI). VSE Batch does not support switching to another language. Therefore the default will be used regardless of the user's specification. The values of SQLCODE, SQLSTATE, SQLERRD1 and SQLERRD2 will be automatically appended to the returned text. The user may also specify to have the entire SQLCA included. If the SQLCODE could not be found in the repository, the entire SQLCA will be returned in the buffer.

If the SQLCA was set by another product (such as DB2 UBD), the descriptive text is retrieved if the SQLCODE exists in the DB2 for VM/VSE repositories. However, the token substitutions may not be correct.

For more information, see *DB2 Server for VSE & VM Application Programming*.

Convert buffer read/write to compiler macro

The DRDA code has over 100 small modules. Each call to an external module has a certain amount of overhead associated with it. Certain modules are called very frequently and this can add up to a significant amount of time. This functionality improves the performance by converting few modules to macros or internal procedures, to reduce this overhead.

Modify Build Tree Creation

This functionality modifies Build Tree creation used by DRDA parsing and generation. It is built in such a way that every code point that is used to search through the tree must be converted to a different format before the search can be done. If modified build tree was created with the converted point, then the code point would not have to be converted every time the tree must be searched. This improves the performance of the DRDA code path length with the minimal search.

Split code point search routines

When parsing a data stream within each parser action routine, a binary search is done to find the specific code point. Some action specific routines are quite large, so the binary search can be long. Splitting and spreading the code point evenly among other modules would reduce the overheads and improves the performance of the DRDA code path length.

DRDA Multi-Row Insert

Multi Row insert is a means of caching homogenous insert statements and sending them as a block to the server for processing. This reduces the overhead of sending a large number of singular inserts and receiving as many responses.

Buffering of homogenous inserts eliminates the need to send an SQL statement to the DB2 server every time an insert is made, thereby improving performance over DRDA.

For more information, see the following DB2 Server for VSE & VM documentation:

- *DB2 Server for VSE & VM Application Programming*

- *DB2 Server for VSE & VM Database Administration*
- *DB2 Server for VM System Administration*
- *DB2 Server for VSE & VM Performance Tuning Handbook*
- *DB2 Server for VSE & VM Quick Reference*
- *DB2 Server for VSE & VM SQL Reference*

Connection Pooling for DRDA TCP/IP in Online Resource Adapter

Connection pooling is a technique that allows multiple users to share a cached set of pre-established connections that provide access to a database. Establishing a connection between a user and a server takes a sizeable time. Users who have validated their entry to a database once need not establish a connection every time a request is submitted. Instead, they can use a pre-established connection from a pool of such connections and get their results much faster.

From the user's point of view, there is a considerable improvement in response time after this line item is implemented.

For more information, see the following documentation on DB2 Server for VSE & VM:

- *DB2 Server for VSE System Administration*
- *DB2 Server for VSE & VM Application Programming*
- *DB2 Server for VSE & VM Operation*
- *DB2 Server for VSE & VM Performance Tuning Handbook*

IBM DB2 Server for VSE, Client Edition

This feature allows the customer the flexibility to install and use only the client (run-time support) component of DB2 Server for VSE without the requirement to buy and install the server component during the installation process of DB2 server for VSE product. The client-only installation enables customers to reduce the total cost of ownership when they have their databases residing on a non-local platform (like VM, z/OS, LUW) and have a large number of their DB2 applications on VSE (like ISQL on CICS, DBSU on VSE, other online/batch applications on VSE).

For more information, see the following DB2 Server for VSE & VM documentation:

- *DB2 Server for VSE System Administration*
- *DB2 Server for VSE Program Directory*

IBM DB2 Server for VM, Client Edition

This feature allows the customer the flexibility to install and use only the client (run-time support) component of DB2 Server for VM without the requirement to buy and install the server component during the installation process of DB2 server for VM product. The client-only installation enables our customers to reduce the total cost of ownership when they have their databases residing on a non-local platform (like VM, z/OS, LUW) and have a large number of their DB2 applications on VM (like ISQL, DBSU, other user applications on VM).

For more information, see the following DB2 Server for VSE & VM documentation:

- *DB2 Server for VM System Administration*
- *DB2 Server for VM Program Directory*

Handling Commit Responses from DB2 UDB Stored Procedures

This feature will allow DB2 Resource Manager on VSE/VM to accept and process results of a stored procedure running in a UDB server with a COMMIT statement in the stored procedure.

Currently, DB2 for VM/VSE client does not handle responses from 'COMMIT' statements coded in DB2 UDB stored procedures. Implementation of this feature will enable handling responses of COMMIT statements in DB2 UDB stored procedures and thus allow users to have COMMIT statements in their stored procedures, while using DB2 for VM/VSE client.

COMMIT statements, however, are not allowed in stored procedures on the DB2 Server for VM/VSE.

For more information, see *DB2 Server for VSE & VM Application Programming*.

Make on-line programs AMODE 31 RMODE ANY

This feature converts DB2 server for VSE online program which presently operate under 24 bit addressing mode from AMODE 24, to AMODE 31 RMODE ANY. Presently, all the online programs are loaded below 16M line. Implementation of this line item ensures that all the online program will be loaded above the 16M line, which results in more virtual storage below the line, which can be utilized by other applications.

For more information, see the following DB2 Server for VSE & VM documentation:

- *DB2 Server for VSE System Administration*
- *DB2 Server for VSE Program Directory*

Provide BIND File Support in VM and in VSE Batch Environments

This feature provides the facility of binding packages across servers. The process of binding is achieved by dividing the program preparation method into two steps. The first step does the precompilation of the embedded SQL programs with the prep parameter 'BIND'. Invocation of VSE/VM preprocessor creates a 'bindfile'. The bindfile can be bound against any DB2 server using VSE/VM binder. During this process, the access path is generated, SQL statements are verified, authorization checks are performed, and package on the target server is created. This line item eliminates the need of re-prepping the source code or porting of packages across DB2 servers.

For more information, see the following DB2 Server for VSE & VM documentation:

- *DB2 REXX SQL for VM/ESA Installation and Reference*
- *DB2 Server for VM Messages and Codes*
- *DB2 Server for VSE & VM Application Programming*
- *DB2 Server for VSE & VM Database Administration*
- *DB2 Server for VM Program Directory*
- *DB2 Server for VSE Program Directory*

Convert TCP/IP LE/C interface to EZASMI API

The feature of converting TCP/IP LE/C interface to EZASMI API intends to replace the current LE/C interface and implement the EZA Assembler Interface (EZASMI)to enhance performance in DB2 Client/Server for VSE over DRDA. Currently, either LE/C interface or CSI Assembler Interface is used for TCP/IP functions. The EZASMI interface makes the code all Assembler.

For more information, see *DB2 Server for VSE Program Directory*

Part 1. User's Guide

This part of the manual presents procedures for performing the tasks provided by the Database Services Utility, which is a part of the DB2 Server for VSE & VM product. The major task areas covered are as follows:

- Familiarizing yourself with the DBS Utility
- Loading data into a DB2 Server for VSE & VM database
- Unloading data stored in a DB2 Server for VSE & VM database
- Reloading data into a DB2 Server for VSE & VM database in a format provided by the DBS Utility
- Interpreting the output of the DBS Utility
- Unloading and reloading packages.

Examples and reference material necessary to perform these tasks are framed in boxes with the procedures themselves. For additional reference information, see Part 2, "Reference."

Usual operation of the Database Services Utility is accessing the database in multiple user mode; therefore, the task descriptions and procedures in this part of the manual mainly address operation of the Database Services Utility with multiple user mode.

Chapter 1. Getting Started

This chapter gives a brief overview of the Database Services (DBS) Utility and explains how to start it. The fundamentals of using the utility are also described, such as defining input and output requirements, working with a report in VSE, a message file in VM, and using SQL statements within the utility. Finally, this chapter describes how to exit from the utility.

Introducing the Database Services Utility

The Database Services Utility is an application program that supplies a user interface to the IBM DB2 Server for VSE & VM product and that, with some limitations, also works with other relational databases that use DRDA flow. Consider using it to load or reload data into, or unload data from, a database. If the amount of data to be processed is large, or if exact sequences of database commands are to be used on a periodic basis, consider using the utility.

You usually employ the Database Services Utility for DB2 Server for VSE & VM for large-scale processing of relational databases in a batch environment. Input to the utility, as well as its reported output, is in the form of sequential files. In VM, you have another way of using the Database Services Utility. Although DB2 Server for VM batch processing is the utility's usual operating mode, you can also use it interactively by specifying a terminal as its input file. You can also direct its output to a terminal instead of storing the output as a physical file.

In addition to loading data into and unloading it from a database, you can use the utility to process SQL statements and to transfer packages into or out of databases. You can do these operations in either single or multiple user mode.

The four primary Database Services Utility control commands are DATALOAD, DATAUNLOAD, UNLOAD, and RELOAD. The UNLOAD and RELOAD commands are qualified by the object they manipulate:

UNLOAD	**RELOAD**
UNLOAD DBSPACE	RELOAD DBSPACE
UNLOAD TABLE	RELOAD TABLE
UNLOAD PACKAGE	RELOAD PACKAGE

The DATALOAD command inserts data from a sequential file into a DB2 Server for VSE & VM table. You specify the format of the sequential file.

The DATAUNLOAD command selects data from tables and copies it to a sequential file. You specify the format of the sequential file.

The UNLOAD commands provide a backup function for existing tables, dbspaces, and packages. These control commands are also useful for distributing copies of data to other sites that use the database manager. The output of each UNLOAD command is a sequential file formatted for the use of its corresponding RELOAD command.

The RELOAD commands restore information previously backed up with UNLOAD commands. The RELOAD commands are also useful for reorganizing database tables or dbspaces and for receiving tables and packages from other sites. With the

RELOAD TABLE command, you can create new tables from logical views previously unloaded from existing tables. You can then build an index for each newly created table. The RELOAD package can be used to distribute packages to other sites that use the DB2 Server for VSE & VM application server or other application servers that support DRDA flow.

The Database Services Utility provides other commands for your convenience:

- A COMMENT command documents your Database Services Utility command input.
- A REBIND PACKAGE command preprocesses existing packages.
- A REORGANIZE INDEX command efficiently reorganizes a table's index in one step.
- A SCHEMA command executes the SQL statements CREATE TABLE, CREATE VIEW, and GRANT in a schema file. A schema file contains an authorization ID and a list of table, view, and privilege definitions.
- A number of SET commands control various processing, environmental, and formatting characteristics. The SET commands turn on or regulate certain SQL statements.

Loading Data into a Database

You can use the Database Services Utility DATALOAD command to load or add rows from a user-defined sequential file. The input to DATALOAD processing consists of a set of Database Services Utility commands and input data records. The utility commands identify:

- tables to be loaded
- Location of tabular column data in an input record
- Format of input-record fields
- Sequential file containing the input records.

You can specify how often DATALOAD processing commits insertions to the database. Specify a number of input data records, and the insertions are committed each time the DATALOAD command processes the specified number of records. If a subsequent error occurs, the database manager only has to undo the database changes made since the last commit point. The committing and restarting capabilities of the database manager are useful when you are loading large amounts of data with the utility.

Referential constraints (rules that require all values in dependent tables to match corresponding values in parent tables) are enforced during DATALOAD processing. This means that primary key rows must be loaded before their foreign key rows. You can improve the utility's performance by deactivating the constraints before loading the data and activating them again afterwards. For descriptions and instructions on DATALOAD processing, see Chapter 2, "Loading Data with the Database Services Utility."

Unloading Data from a Database

The DATAUNLOAD command allows you to selectively unload data from a database to a sequential access method (SAM) output file. You can:

- Create a file for transporting data from a DB2 Server for VSE & VM to a non-DB2 Server for VSE & VM processing environment
- Create a sequential file, modify it, and reload it into tables with the DATALOAD command.

You can use the other Database Services Utility unload-data commands (UNLOAD DBSPACE and UNLOAD TABLE) to:
- Create a backup for specific data
- Move data to another DB2 Server for VSE & VM database manager.

You can also use these UNLOAD commands, immediately followed by their RELOAD counterparts, to:
- Reclaim fragmented disk space
- Reorder data records to match indexes.

The main difference between the DATAUNLOAD and UNLOAD commands is that DATAUNLOAD allows you to specify more about the data you unload than the UNLOAD commands allow. Consequently, the UNLOAD commands are simpler, but it is easier to work with output data from a DATAUNLOAD command. For descriptions and instructions on DATAUNLOAD and UNLOAD processing, see Chapter 3, “Unloading Data with the Database Services Utility.”

Reloading Data into a Database

DATALOAD and RELOAD are essentially the same kind of operation: they both insert data into databases; however, RELOAD inserts data that was previously unloaded using the UNLOAD command while DATALOAD uses a user-defined file of data, or the output file of a DATAUNLOAD command.

RELOAD processing can purge existing tables before reloading them (from previously *unloaded* information). Similarly, you can unload a view as if it were a table and reload it as a new table. When RELOAD creates a new table, it does not automatically re-create all the entities associated with the old table; you must specify views, indexes, keys, and access privileges. For descriptions and instructions on RELOAD processing, see Chapter 4, “Reloading Data with the Database Services Utility.”

Unloading Packages from a Database

You can use the Database Services Utility to unload a *package* from a DB2 Server for VSE & VM database to a *portable file*. A package consists of the internally optimized application SQL statements stored in (bound to) the database at preprocessing time and used by the database with the application at execution time. A portable file is one that contains an unloaded DB2 Server for VSE & VM package that is ready for distribution to another application server. You can unload a package to a file to:
- Create a backup of the package before making changes to it
- Reload a package to another application server.

The UNLOAD PACKAGE command unloads the package, along with information about the way it was created, to a portable file. You can then send the file to the application server that requires it. It is unnecessary to distribute source programs or to preprocess and compile source code at the receiving location. For descriptions and instructions on unloading packages, see Chapter 5, “Unloading and Reloading Packages with the Database Services Utility,” on page 83.

Reloading Packages into a Database

You can use the Database Services Utility to load a package from a file into a DB2 Server for VSE & VM database. You can do this to achieve the following:
- Restore a previous version of a package
- Install an application that is distributed in a portable file.

The database manager preprocesses reloaded packages to ensure that all dependencies are satisfied on the installing system.

When a RELOAD PACKAGE command loads a package into an application server, the module can replace another package with the same name. The new package can carry over the run-privileges previously granted to users of the replaced version. You can reload a package created and unloaded on a VM system, and use it on a VSE system; or you can reload a package created and unloaded on a VSE system, and use it on a VM system. For descriptions and instructions on reloading packages, see Chapter 5, "Unloading and Reloading Packages with the Database Services Utility."

Processing SQL Statements with the Database Services Utility

The Database Services Utility executes SQL statements against the database. You can use most SQL statements in a VM utility control file or a VSE utility input control card file. SQL statements not supported by the Database Services Utility are those used only in application programs (SELECT statements with INTO clauses, cursor management commands, DESCRIBE, EXECUTE, INCLUDE, PREPARE, and WHENEVER).

A Database Services Utility Job

DB2 Server for VM Components

A basic job has five components that control the input and output of data. All five are discussed in more detail later in this chapter:

Control File
The control file contains Database Services Utility commands and SQL statements that the utility processes. The control file must have a fixed format and a record length of 80 characters.

Message File
This output file contains a list of all commands executed, as well as the results of these commands. These results can be messages to indicate whether the command was executed successfully, as well as data that was obtained by a SELECT statement.

Input/Output File
Either this file contains data to be loaded or copied to a database, or it is the file to which data is written. Its use depends on the Database Services Utility command you are using.

File Definitions
File definitions specify input and output requirements for the above three files.

SQLDBSU EXEC
This EXEC starts a Database Services Utility job. You can also use the SQLDBSU EXEC to specify the input and output requirements for the control and message files.

DB2 Server for VSE Files

A basic job has three components that control the input and output of data. All three are discussed in more detail later in this chapter:

Input Control Card File
The input control card file contains Database Services Utility commands and SQL statements that the utility processes. The input control card file must have a fixed format and a record length of 80 characters.

Report
The report contains a list of all commands executed, as well as the

results of these commands. These results can be messages to indicate whether the command was executed successfully, as well as data that was obtained by a SELECT statement.

Input/Output File
Either this file contains data to be loaded or copied to a database, or it is the file to which data is written. Its use depends on the Database Services Utility command you are using.

Starting and Using the Database Services Utility

The Database Services Utility can be started to access the database in either *multiple user mode* or *single user mode*.

Multiple User Mode

Multiple user mode is the usual way of running the application server. It permits multiple users to access a DB2 Server for VSE & VM application server simultaneously. Unless you have database maintenance to perform or another task requiring a dedicated database, run the utility with multiple user mode.

DB2 Server for VSE:

For more information about starting the Database Services Utility with multiple user mode, see "Multiple User Mode Job Control" on page 108.

DB2 Server for VM:

The DB2 Server for VM Database Services Utility with multiple user mode cannot run either in the CMS/DOS environment, or in CMS subset.

In preparation for running the Database Services Utility with multiple user mode, initialize the user machine by specifying defaults using the SQLINIT EXEC. On the CMS command line, type:

```
SQLINIT DBNAME(server-name)
```

where *server-name* is the name of the application server to be accessed. Press ENTER.

For more information on the SQLINIT EXEC in multiple user mode, see "Running the DB2 Server for VM Database Services Utility with Multiple User Mode" on page 128.

Single User Mode

Run the Database Services Utility with single user mode to prevent concurrent access of a DB2 Server for VSE & VM application server by other users. Unless you have database maintenance to perform, are the sole user of an application server, or are performing a task requiring a dedicated database, run the utility with multiple user mode.

DB2 Server for VSE:

For more information on running the Database Services Utility with single user mode, see "Single User Mode Job Control" on page 106.

DB2 Server for VM:

In DB2 Server for VM single user mode, the SQLINIT EXEC is unnecessary; the Database Services Utility and the application server are executed in the same virtual machine, and you specify the desired application server with the DBNAME parameter of the SQLDBSU EXEC.

For more information on using the SQLDBSU EXEC with single user mode, see Chapter 7, "Using the Database Services Utility from Application Programs," on page 105 and "SQLDBSU EXEC Format" on page 129.

Note: Because usual operation of the Database Services Utility is with multiple user mode, the task descriptions and procedures in this part of the manual largely address operation of the utility with multiple user mode.

Overview of Database Services Utility Files

The Database Services Utility is a general purpose utility that requires two or three files to run: one for Database Services Utility command or SQL statement input, one for message output, and one for data output or input.

The required DB2 Server for VSE input file is the *input control card file*, and it is assigned to SYSIPT. The input control card file contains utility control commands, which are described in the following section.

The Database Services Utility creates a *report*; it is assigned to SYSLST. The report lists the input control card file records, messages, and results.

The DB2 Server for VM control file and message file are usually CMS files. You can define the control file to any sequential tape or DASD file supported by CMS OS/QSAM, to a virtual reader file, or to the terminal. You can define the message file to any sequential tape file supported by CMS OS/QSAM, to a virtual print file, or to the terminal.

Often you require an additional file for Database Services Utility input or output. The Database Services Utility commands that use additional files for input or output contain a data definition name (*ddname*) parameter that you must specify to identify the additional input or output file. In DB2 Server for VSE, the *ddname* refers to the file name specified in the applicable DLBL or TLBL system control statement. In DB2 Server for VM, this parameter refers to the *ddname* defined in a CMS FILEDEF command. A *ddname* can be from one to eight characters. The first character of a *ddname* must be alphabetic (or a national character). For more information about file definition, see the relevant command description or refer to Appendix B, "FILEDEF Command Syntax and Notes," on page 249.

The required DB2 Server for VM input file is the control file (or command file), and it is assigned to the *ddname* SYSIN. The control file contains utility control commands, which are described in the following section.

The required DB2 Server for VM output file is the message file; it is assigned to the *ddname* SYSPRINT. The utility lists the control file records, writes messages, and prints results in the message file.

Note: You do not need an additional file when you use the DATALOAD command if you place the data input information in the command file. You do require an additional input or output file with all other commands that have a *ddname* parameter.

The Database Services Utility supports the use of multiple-volume tape files and variable-length, spanned records in either environment. For additional information on tape support, refer to the *DB2 Server for VM System Administration* or the *DB2 Server for VSE System Administration* manual.

Working with an Input Control Card File in DB2 Server for VSE

Creating a Control Card File

Create a control file as follows.

1. Provide the following commands and statements shown in Figure 3 with the JCL statements needed to run the job.

```
// JOB DBS UTILITY EXAMPLE VSE MULTIPLE USER MODE JOB CONTROL
// EXEC PROC=ARIS62PL           >——DB2 Server for VSE Production Library Definition
// EXEC ARIDBS, SIZE=AUTO       >——invoke DBS Utility
 CONNECT your user ID IDENTIFIED BY your password;
 SELECT * FROM SQLDBA.DEPARTMENT;
 SELECT * FROM SQLDBA.PROJECT;
/*
/&
```

Figure 3. Example of a Simplified Input Control Card File

The following statement runs the Database Services Utility:

```
// EXEC ARIDBS,SIZE=AUTO
```

2. Ensure that the input control card file has a fixed record length.
3. Store the input control card file.

Working with a Report

The Database Services Utility creates a report on the device that your installation assigned to SYSLST.

After you submit the Database Services Utility job that you created in Figure 3, and it finishes processing the input control card file, look at the results in the report shown in Figure 4 on page 10.

Note: The report may contain error messages if errors occurred when the Database Services Utility was processing the commands in the input control card file.

```
ARI0801I DBS Utility started: 07/18/89 16:10:31.          ◄──────────── 1
         AUTOCOMMIT = OFF ERRORMODE = OFF                 ◄─────────┬─ 2
         ISOLATION LEVEL = REPEATABLE READ                ◄─────────┘
───────► CONNECT "SQLDBA  " IDENTIFIED BY ********;
ARI8004I User SQLDBA connected to database SQLDBA.
ARI0500I SQL processing was successful.
ARI0505I SQLCODE = 0  SQLSTATE = 00000  ROWCOUNT = 0
───────►
───────► SELECT * FROM SQLDBA.DEPARTMENT;                 ◄──────────── 3
SELECT * FROM SQLDBA.DEPARTMENT                                               PAGE      1
DEPTNO DEPTNAME                      MGRNO  ADMRDEPT      ───┐
────── ───────────────────────────── ────── ────────         │
                                                             │
A00    SPIFFY COMPUTER SERVICE DIV.  000010 A00              │
B01    PLANNING                      000020 A00              │
C01    INFORMATION CENTER            000030 A00              │
D01    DEVELOPMENT CENTER                   A00              ├───── 4
D11    MANUFACTURING SYSTEMS         000060 D01              │
D21    ADMINISTRATION SYSTEMS        000070 D01              │
E01    SUPPORT SERVICES              000050 A00              │
E11    OPERATIONS                    000090 E01              │
E21    SOFTWARE SUPPORT              000100 E01              │
ARI0850I SQL SELECT processing successful: Rowcount = 9   ───┘
───────► SELECT * FROM SQLDBA.PROJECT;                ◄──────────── 5
```

Figure 4. Database Services Utility: Example Report Output (Part 1 of 2)

```
SELECT * FROM SQLDBA.PROJECT                                                     PAGE       2
PROJNO PROJNAME            DEPTNO RESPEMP PRSTAFF PRSTDATE   PRENDATE   ─┐
────── ─────────────────── ────── ─────── ─────── ────────── ──────────  │
                                                                         │
AD3100 ADMIN SERVICES      A00    000010     6.50 1982-01-01 1983-02-01  │
MA2100 WELD LINE AUTOMATIO D01    000010    12.00 1982-01-01 1983-02-01  │
AD3111 PAYROLL PROGRAMMING B01    000020     2.00 1982-01-01 1983-02-01  │
PL2100 WELD LINE PLANNING  B01    000020     1.00 1982-01-01 1982-09-15  │
IF1000 QUERY SERVICES      C01    000030     2.00 1982-01-01 1983-02-01  │
IF2000 USER EDUCATION      C01    000030     1.00 1982-01-01 1983-02-01  │
OP1000 OPERATION SUPPORT   E01    000050     6.00 1982-01-01 1983-02-01  │
OP2000 GEN SYSTEMS SERVICE E01    000050     5.00 1982-01-01 1983-02-01  │
MA2110 W L PROGRAMMING     D11    000060     9.00 1982-01-01 1983-02-01  │
AD3110 GENERAL AD SYSTEMS  E11    000090     6.00 1982-01-01 1983-02-01  │
OP1010 OPERATION           E11    000090     5.00 1982-01-01 1983-02-01  ├── 6
OP2010 SYSTEMS SUPPORT     E21    000100     4.00 1982-01-01 1983-02-01  │
MA2112 W L ROBOT DESIGN    D11    000150     3.00 1982-01-01 1982-12-01  │
MA2113 W L PROD CONT PROGS D11    000160     3.00 1982-02-15 1982-12-01  │
MA2111 W L PROGRAM DESIGN  D11    000220     2.00 1982-01-01 1982-12-01  │
AD3112 PERSONNEL PROGRAMMG D21    000250     1.00 1982-01-01 1983-02-01  │
AD3113 ACCOUNT.PROGRAMMING D21    000270     2.00 1982-01-01 1983-02-01  │
OP2011 SCP SYSTEMS SUPPORT E21    000320     1.00 1982-01-01 1983-02-01  │
OP2012 APPLICATIONS SUPPOR E21    000330     1.00 1982-01-01 1983-02-01  │
OP2013 DB/DC SUPPORT       E21    000340     1.00 1982-01-01 1983-02-01  │
ARI0850I SQL SELECT processing successful: Rowcount = 20                ─┘
ARI0802I End of command file input.                             ◄─────────── 7
ARI8997I ...Begin COMMIT processing.                                 ─xxxxx┐
ARI0811I ...COMMIT of any database changes successful.                     │
ARI0809I ...No error(s) occurred during command processing.                ├── 8
ARI0808I DBS processing completed: 07/18/89 16:10:33.                ─xxxxx┘
```

Figure 4. Database Services Utility: Example Report Output (Part 2 of 2)

Notes for Figure 4 on page 10:

1	The Database Services Utility start message.
2	The Database Services Utility default values. See "Set-Item Commands" on page 214 in Chapter 8, "Command Reference," on page 135 for details on changing these defaults.
3	The first SELECT statement that the Database Services Utility is to process.
4	Results of the Database Services Utility processing the SELECT statement show the rows retrieved from the table, a message to indicate that the SELECT statement was successful, and the number of rows retrieved.
5	The next SELECT statement that the Database Services Utility is to process.
6	Results of the Database Services Utility processing the SELECT statement show the rows retrieved from the table, a message to indicate that the SELECT statement was successful, and the number of rows retrieved.
7	Database Services Utility has processed all commands in the input control card file.
8	Database Services Utility completion messages.

If you encounter the following message, look at the report to find the error or errors:

```
ARI0807E ...Error(s) occurred during command processing.
```

Error types are listed and discussed in Chapter 9, "Error Handling and Debugging," on page 223. Item **6** in Figure 5 shows an example of an error found in a report.

```
ARI0801I DBS Utility started: 07/18/89 16:10:47.       ◄──────────── 1
         AUTOCOMMIT = OFF ERRORMODE = OFF               ◄──────┬──── 2
         ISOLATION LEVEL = REPEATABLE READ              ◄──────┘
───────► CONNECT "SQLDBA  " IDENTIFIED BY ********;
ARI8004I User SQLDBA connected to database  SQLDBA.
ARI0500I SQL processing was successful.
ARI0505I SQLCODE = 0  SQLSTATE = 00000  ROWCOUNT = 0
───────►
───────► SELECT * FROM SQLDBA.DEPARTMENT;               ◄──────────── 3
SELECT * FROM SQLDBA.DEPARTMENT                                               PAGE       1
DEPTNO DEPTNAME                     MGRNO  ADMRDEPT     ───┐
────── ──────────────────────────── ────── ────────        │
                                                            │
A00    SPIFFY COMPUTER SERVICE DIV.  000010 A00              │
B01    PLANNING                     000020 A00              │
C01    INFORMATION CENTER           000030 A00              │
D01    DEVELOPMENT CENTER                  A00              ├──── 4
D11    MANUFACTURING SYSTEMS        000060 D01              │
D21    ADMINISTRATION SYSTEMS       000070 D01              │
E01    SUPPORT SERVICES             000050 A00              │
E11    OPERATIONS                   000090 E01              │
E21    SOFTWARE SUPPORT             000100 E01              │
ARI0850I SQL SELECT processing successful: Rowcount = 9  ───┘
───────► SELECT * FROM SQLDBA.PROJJECT;                 ◄──────────── 5
ARI0503E An SQL error has occurred.                        ──┐
         SQLDBA.PROJJECT was not found in the system catalogs.│
ARI0505I SQLCODE = -204  SQLSTATE = 52004  ROWCOUNT = 0      ├── 6
ARI0504I SQLERRP: ARIXOCA SQLERRD1: -100 SQLERRD2: 0          │
ARI0851E SQL SELECT processing unsuccessful: Rowcount = 0  ──┘
ARI8998I ...Begin ROLLBACK processing.                   ───┐
ARI0811I ...ROLLBACK of any database changes successful.    │
ARI0813I ...Suspend command execution:                      ├──── 7
         AUTOCOMMIT = OFF ERRORMODE = ON                    │
ARI0802I End of command file input.                      ───┘
ARI0807E ...Error(s) occurred during command processing. ◄──┬──── 8
ARI0808I DBS processing completed: 07/18/89 16:10:47.    ◄──┘
```

Figure 5. Example of a Database Services Utility Error

Notes for Figure 5:

1 The Database Services Utility start message.

2 The Database Services Utility default values. See "Set-Item Commands" on page 214 in Chapter 8, "Command Reference," on page 135 for details on changing these defaults.

3 The first SELECT statement that the Database Services Utility is to process.

4 Results of the Database Services Utility processing the SELECT statement shows the rows retrieved from the table, a message to indicate that the SELECT statement was successful, and the number of rows retrieved.

5 The next SELECT statement that the Database Services Utility is to process.

6 Messages indicating that the SELECT statement could not be successfully processed. The message indicates that the SQLDBA.PROJJECT table could not be found in the database; *PROJJECT* is misspelled. You should now correct the spelling in the input control card file and run the job again.

7 Indicates that the Database Services Utility encountered an error and cannot process any commands that follow. This message is not relevant to the present example because no more commands follow. If commands followed this one in error, they would **not** be processed.

Note: This command suspension can be controlled by the user; see "SET ERRORMODE" in Chapter 8, "Command Reference," on page 135.

8 Completion messages.

Working with a Control File in DB2 Server for VM

Using a Control File

The control file contains a group of SQL statements and Database Services Utility commands to be executed. Grouping these commands in one file gives you the option of saving the file for periodic execution of the sequence of commands in it; you do not have to retype these commands. Use a control file when running a batch job, testing statements or commands, or when you expect to use the same or similar utility commands again.

Creating a Control File

Create a control file by using an editor program as follows.

1. Give your control file a file name, file type, and file mode and start the editor. If you are doing this exercise to learn about the Database Services Utility, call your control file COMMANDS DBSU A (if you are using your A-disk), and set the width of the file to 80.
2. Type the desired utility commands and SQL statements. You must use uppercase; for example, you can type:
   ```
   SELECT * FROM SQLDBA.DEPARTMENT;
   SELECT * FROM SQLDBA.PROJECT;
   ```
   Note: Always end SQL statements with a semicolon.
3. If your editor program is set to variable length record format, set it to a fixed length record format.

   Note: This step sets the record length of the control file to a fixed length. If the default of the editor is set to variable length record format, you must repeat this step each time you edit the file.
4. Store the control file and leave the editor.

For more information on Database Services Utility commands, see Chapter 8, "Command Reference," on page 135. For more information on SQL statements, see the *DB2 Server for VSE & VM SQL Reference*.

Defining Input and Output Requirements

You must define I/O requirements to the Database Services Utility for the control file, the message file, and any input or output data files.

You define your input and output requirements to the Database Services Utility by using FILEDEF commands. The SQLDBSU EXEC generates standard FILEDEF statements for the control and message files; if you are using an additional file for data input or output, or need parameters not supplied by the SQLDBSU EXEC, you must use a FILEDEF statement to supplement the EXEC. When you specify options other than the SQLDBSU EXEC default options, the EXEC defaults are overridden.

Because you use a data file for input or output with the RELOAD, DATAUNLOAD, UNLOAD and SCHEMA commands, you must write a FILEDEF statement for these commands. The DATALOAD command does not require a FILEDEF statement when the input data is in the command file. For further details about command specific FILEDEF information, see the section about using file definitions for the particular command.

You should use a FILEDEF statement as an *addition* to the FILEDEF statements issued by the SQLDBSU EXEC, not as a replacement. When you do use customized FILEDEF statements in addition to the SQLDBSU EXEC, the FILEDEFs precede the SQLDBSU EXEC.

Using File Definitions

Use a FILEDEF command to identify a CMS file, a virtual reader file, a virtual printer file, or any sequential tape or DASD file supported by CMS/QSAM. The FILEDEF command assigns a name to the file and specifies the file's device type and file options.

Figure 6 illustrates the syntax of a FILEDEF statement:

Format:

```
►►──FIledef──ddname──┬─Terminal─────────┬──┬──────────────────┬──────────────────────────────────►◄
                     ├─PRinter──────────┤  └─(──Options──┬───┬┘
                     ├─Reader───────────┤                └─)─┘
                     ├─DISK──fn_ft_fm───┤
                     └─TAPn─────────────┘
```

Figure 6. FILEDEF Statement Syntax

ddname (data definition name)

Identifies the name of the input or output file that you are defining.

Device type can be one of the following parameters:

Terminal	Your workstation
PRinter	The spooled printer available to you
Reader	The spooled reader available to you
DISK fn ft fm	Virtual direct access storage device (DASD) CMS file
TAPn	Magnetic tape drive, where *n* can be *1*, *2*, *3*, or *4*, representing virtual units 181, 182, 183, and 184, respectively.

Options: To avoid error messages, specify only those options that are valid for a particular device. Table 19 on page 251 shows valid options for each device type.

The message ARI0868I (in the message file) identifies the file characteristics used by Database Services Utility processing.

The following shows a FILEDEF statement that defines an input data file. In this example, DBSFILE is the name of the input file as it is referred to in your Database Services Utility command.

```
FILEDEF DBSFILE DISK DBSFILE DATA A (RECFM F LRECL 800
```

DBSFILE is a file on DASD called DBSFILE DATA A. It has a fixed record length of 800.

For an explanation of FILEDEF parameters and options, see Appendix B, “FILEDEF Command Syntax and Notes,” on page 249.

Using the SQLDBSU EXEC

If you have simple, straightforward I/O needs for the control and message files, the SQLDBSU EXEC, without supplementary FILEDEF commands, is probably all you need. You can choose only one of the following input control file options:

- **A named CMS file** for which SQLDBSU issues this FILEDEF:
  ```
  FILEDEF SYSIN DISK file-name file-type file-mode
  (RECFM FB LRECL 80 BLOCK 800
  ```
- **A virtual reader file** for which SQLDBSU issues this FILEDEF:
  ```
  FILEDEF SYSIN READER (RECFM F LRECL 80
  ```
- **A workstation as control file** for which SQLDBSU issues this FILEDEF:
  ```
  FILEDEF SYSIN TERMINAL (RECFM F LRECL 80
  ```

You can choose only one of the following output message file options with the SQLDBSU EXEC.

- **A named CMS file** for which SQLDBSU issues this FILEDEF:
  ```
  FILEDEF SYSPRINT DISK file-name file-type file-mode
  (RECFM FBA LRECL 121 BLOCK 1210
  ```
- **A virtual printer** for which SQLDBSU issues this FILEDEF:
  ```
  FILEDEF SYSPRINT PRINTER (RECFM FA LRECL 121
  ```
- **A workstation as message file** for which SQLDBSU issues this FILEDEF:
  ```
  FILEDEF SYSPRINT TERMINAL (RECFM F LRECL 120
  ```

If, for example, your control file is COMMANDS DBSU and you want to have the message file displayed on your terminal, your SQLDBSU EXEC statement is:

```
SQLDBSU SYSIN (COMMANDS DBSU A) SYSPRINT (T)
```

Note: When the control file is assigned as TERMINAL, do the following:

- Use the same character positions and same command syntax as if entering commands or data into a CMS file; end all commands with a semicolon.
- Use uppercase or lowercase because CMS converts your input to uppercase. If your input entered from the terminal must contain lowercase values, you must issue the following FILEDEF before issuing the SQLDBSU EXEC without the SYSIN parameter specification:
  ```
  FILEDEF SYSIN TERMINAL (RECFM F LRECL 80 LOWCASE
  ```
  If this FILEDEF is issued, all the Database Services Utility command and SQL statement keywords must be entered in uppercase.

- Do not submit command records with sequence numbers in positions 73–80 when you are using the READ FILE command. (When SYSIN=TERMINAL, positions 73–80 are used for command information.)

Note: With single user mode, the SQLDBSU statement has additional parameters.

For detailed information on the SQLDBSU EXEC and startup of the Database Services Utility, see Chapter 7, "Using the Database Services Utility from Application Programs," on page 105. For usage notes and syntax of the CMS FILEDEF command, see Appendix B, "FILEDEF Command Syntax and Notes," on page 249.

Sample Startup

This procedure uses the control file you created in "Using the SQLDBSU EXEC" on page 15 to startup the Database Services Utility with the SQLDBSU EXEC.

Note: Consider using the same file name for both the control and message files to identify the input and output as belonging to the same job. Use different file types for the control and message files to prevent the output data and messages from overwriting the control file contents.

On the CMS command line, type:

```
SQLDBSU SYSIN (COMMANDS DBSU A) SYSPRINT (COMMANDS RESULT A)
```

Press ENTER to start the Database Services Utility. The commands in your COMMANDS DBSU A file are now executed. The utility processes the commands and displays the results as shown in Figure 7.

```
ARI0717I Start SQLDBSU EXEC: 07/18/89 16:09:52 EST◄──────────  1
ARI0662I EMSG function value reset to: ON.
ARI0659I Line—edit symbols reset:
         LINEND=# LINEDEL=OFF CHARDEL=OFF ESCAPE=OFF TABCHAR=OFF
ARI0655I Input file (SYSIN): COMMANDS DBSU A        ◄───────┐
ARI0656I Message file (SYSPRINT): COMMANDS RESULT A         │
ARI0320I The default database name is SQLDBA.               │
ARI0663I FILEDEFS in effect are:                            ├──── 2
ARISQLLD DISK     ARISQLLD LOADLIB  Q1                      │
SYSIN    DISK     COMMANDS DBSU     A1                      │
SYSPRINT DISK     COMMANDS RESULT   A1      ◄───────────────┘
ARI0809I ...No errors occurred during command processing.◄──── 3
ARI0808I DBS processing completed: 07/18/89 16:09:55.◄────┐
ARI0660I Line—edit symbols restored:                       │
         LINEND=# LINEDEL=OFF CHARDEL=OFF ESCAPE=ó TABCHAR=ON
ARI0657I EMSG function value restored to: TEXT.            │
ARI0796I End SQLDBSU EXEC: 07/18/89 16:09:56 EST◄─────────┴─── 4
```

Figure 7. Messages Displayed during Processing

Notes for Figure 7:

1 Informs the user that the Database Services Utility started.

2 Shows the input (or control) file name, the message file name, the database being accessed, and the FILEDEFs in effect.

3 Identifies any errors that occur when Database Services Utility processes the commands in the control file.

4 Indicates that the Database Services Utility is finished processing.

You may receive the following message instead of the message displayed at **3**:

```
ARI0807E ...Error(s) occurred during command processing.
```

This message indicates that an error occurred when the Database Services Utility was processing the commands in the control file. Error types are listed and discussed in Chapter 9, "Error Handling and Debugging," on page 223.

Working with a Message File

The Database Services Utility automatically creates a message file with the name you supplied in the SQLDBSU EXEC parameter. If a file already exists with the same name, it is overwritten.

After the utility is run and finishes its processing, view the message file to see the results of processing the control file commands. Figure 8 on page 18 shows the contents of message file COMMANDS RESULT A.

```
1ARI0801I DBS Utility started: 07/18/89 16:10:31.          ◄──────── 1
          AUTOCOMMIT = OFF ERRORMODE = OFF                 ◄─────┐ 2
          ISOLATION LEVEL = REPEATABLE READ                ◄─────┘
0───────► SELECT * FROM SQLDBA.DEPARTMENT;                 ◄──────── 3
1SELECT * FROM SQLDBA.DEPARTMENT                                              PAGE       1
0DEPTNO DEPTNAME                      MGRNO  ADMRDEPT      ──┐
 ------ ----------------------------- ------ --------        │
                                                             │
 A00    SPIFFY COMPUTER SERVICE DIV.  000010 A00             │
 B01    PLANNING                      000020 A00             │
 C01    INFORMATION CENTER            000030 A00             │
 D01    DEVELOPMENT CENTER                   A00             ├──── 4
 D11    MANUFACTURING SYSTEMS         000060 D01             │
 D21    ADMINISTRATION SYSTEMS        000070 D01             │
 E01    SUPPORT SERVICES              000050 A00             │
 E11    OPERATIONS                    000090 E01             │
 E21    SOFTWARE SUPPORT              000100 E01             │
0ARI0850I SQL SELECT processing successful: Rowcount = 9   ──┘
1───────► SELECT * FROM SQLDBA.PROJECT;                ◄──────── 5
1SELECT * FROM SQLDBA.PROJECT                                                 PAGE       2
0PROJNO PROJNAME             DEPTNO RESPEMP PRSTAFF PRSTDATE   PRENDATE   ──┐
 ------ -------------------- ------ ------- ------- ---------- ----------   │
                                                                            │
 AD3100 ADMIN SERVICES       A00    000010     6.50 1982-01-01 1983-02-01   │
 MA2100 WELD LINE AUTOMATIO  D01    000010    12.00 1982-01-01 1983-02-01   │
 AD3111 PAYROLL PROGRAMMING  B01    000020     2.00 1982-01-01 1983-02-01   │
 PL2100 WELD LINE PLANNING   B01    000020     1.00 1982-01-01 1982-09-15   │
 IF1000 QUERY SERVICES       C01    000030     2.00 1982-01-01 1983-02-01   │
 IF2000 USER EDUCATION       C01    000030     1.00 1982-01-01 1983-02-01   │
 OP1000 OPERATION SUPPORT    E01    000050     6.00 1982-01-01 1983-02-01   │
 OP2000 GEN SYSTEMS SERVICE  E01    000050     5.00 1982-01-01 1983-02-01   │
 MA2110 W L PROGRAMMING      D11    000060     9.00 1982-01-01 1983-02-01   │
 AD3110 GENERAL AD SYSTEMS   E11    000090     6.00 1982-01-01 1983-02-01   │
 OP1010 OPERATION            E11    000090     5.00 1982-01-01 1983-02-01   ├── 6
 OP2010 SYSTEMS SUPPORT      E21    000100     4.00 1982-01-01 1983-02-01   │
 MA2112 W L ROBOT DESIGN     D11    000150     3.00 1982-01-01 1982-12-01   │
 MA2113 W L PROD CONT PROGS  D11    000160     3.00 1982-02-15 1982-12-01   │
 MA2111 W L PROGRAM DESIGN   D11    000220     2.00 1982-01-01 1982-12-01   │
 AD3112 PERSONNEL PROGRAMMG  D21    000250     1.00 1982-01-01 1983-02-01   │
 AD3113 ACCOUNT.PROGRAMMING  D21    000270     2.00 1982-01-01 1983-02-01   │
 OP2011 SCP SYSTEMS SUPPORT  E21    000320     1.00 1982-01-01 1983-02-01   │
 OP2012 APPLICATIONS SUPPOR  E21    000330     1.00 1982-01-01 1983-02-01   │
 OP2013 DB/DC SUPPORT        E21    000340     1.00 1982-01-01 1983-02-01   │
0ARI0850I SQL SELECT processing successful: Rowcount = 20                 ──┘
1ARI0802I End of command file input.                          ◄───────────── 7
 ARI8997I ...Begin COMMIT processing.                         ──────┐
 ARI0811I ...COMMIT of any database changes successful.             │
 ARI0809I ...No error(s) occurred during command processing.        ├─────── 8
 ARI0808I DBS processing completed: 07/18/89 16:10:33.        ──────┘
```

Figure 8. Database Services Utility: Sample Message File Output

Notes for Figure 8:

1 The Database Services Utility start message.

2 The Database Services Utility default values. See Set-Item Commands in Chapter 8, "Command Reference" for details on changing these defaults.

3 The first SELECT statement that the Database Services Utility processes.

4 Results of the Database Services Utility processing the SELECT statement show the rows retrieved from the table, a message indicating that the SELECT statement was successful, and the number of rows retrieved.

5 The next SELECT statement that the Database Services Utility processes.

6 Results of the Database Services Utility processing the SELECT statement show the rows retrieved from the table, a message indicating that the SELECT statement was successful, and the number of rows retrieved.

7 Database Services Utility has processed all commands in the control file.

8 Database Services Utility completion messages.

Using the Database Services Utility on Remote Application Servers Which Support DRDA Flow

With the implementation of the Distributed Relational Database Architecture (DRDA), you can use the Database Services Utility on remote application servers which support the DRDA flow. Before you can use the Database Services Utility on an unlike application server, the Utility must be preprocessed on the application server using the ERROR preprocessing option, and the table SQLDBA.DBSOPTIONS must also exist on the unlike application server. Refer to the *DB2 Server for VM System Administration* or the *DB2 Server for VSE System Administration* manual for more information on using the Database Services Utility on an unlike application server using DRDA flow.

To access a non-DB2 Server for VM application server, a VM DBSU user can use the SQLINIT EXEC with the PROTOCOL options set to AUTO or DRDA. (You can also access a DB2 Server for VM application server with the protocol option set to DRDA). In VSE, the Database Services Utility can only access a remote DRDA application server if the remote DRDA application server is specified in the DBNAME directory as a remote server.

Only the following Database Services Utility commands are supported when DRDA flow is used:
- DATAUNLOAD
- DATALOAD
- RELOAD PACKAGE
- SET commands except SET ISOLATION and SET UPDATE STATISTICS.

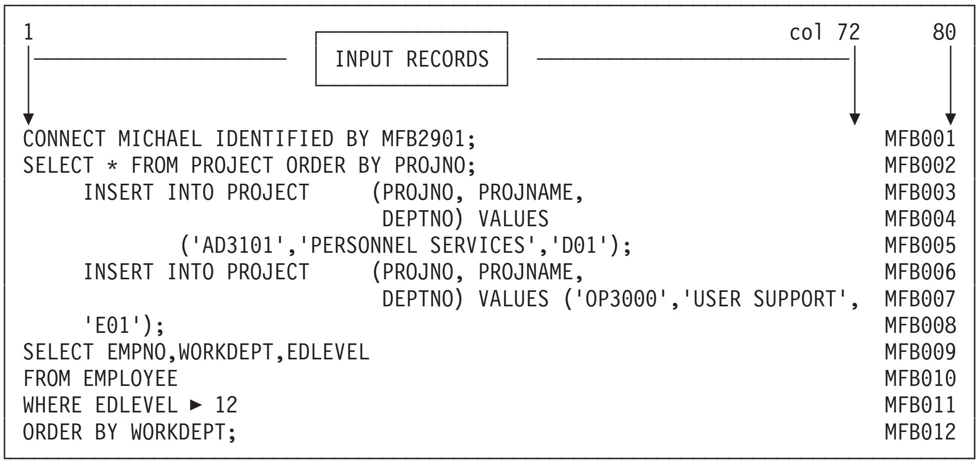
The ISOLATION level is always assumed to be **CS** when DRDA flow is used. If an ISOLATION level other than CS is requested, the command will have no effect and the following message will be displayed:

```
ARI2906I - The only valid isolation level is CS when the DRDA
protocol is used.  Isolation level CS is now in effect.
```

Using SQL Statements within the Database Services Utility

Figure 9 on page 20 is a sample Database Services Utility file that executes SQL statements. It illustrates some of the principles described so far.

Note: You must always end SQL statements with a semicolon.



```
CONNECT MICHAEL IDENTIFIED BY MFB2901;                                  MFB001
SELECT * FROM PROJECT ORDER BY PROJNO;                                  MFB002
     INSERT INTO PROJECT     (PROJNO, PROJNAME,                         MFB003
                              DEPTNO) VALUES                            MFB004
             ('AD3101','PERSONNEL SERVICES','D01');                     MFB005
     INSERT INTO PROJECT     (PROJNO, PROJNAME,                         MFB006
                              DEPTNO) VALUES ('OP3000','USER SUPPORT',  MFB007
     'E01');                                                            MFB008
SELECT EMPNO,WORKDEPT,EDLEVEL                                           MFB009
FROM EMPLOYEE                                                           MFB010
WHERE EDLEVEL ► 12                                                      MFB011
ORDER BY WORKDEPT;                                                      MFB012
```

Figure 9. Database Services Utility Example File

CONNECT

The Database Services Utility supports the SQL CONNECT statement so that you can:

- Identify yourself as an SQL user
- Identify and switch to another application server

Identifying Yourself as a Particular SQL User

You can use the CONNECT statement to identify yourself as a particular DB2 Server for VSE & VM user for the current application server. To identify yourself to the database manager, enter the following:

```
CONNECT authorization-id IDENTIFIED BY password;
```

where *authorization-id* is either your SQL identifier (if you have one) or your user ID, and *password* is your database-access password.

The ID specified in the last CONNECT statement processed by the database manager is the user ID on which the database manager bases its authorization checking for all subsequent Database Services Utility processing.

You can use the CONNECT statement to identify yourself as a user of another application server. To do so, use:

```
CONNECT authorization-id IDENTIFIED BY password TO server-name;
```

where *authorization-id* is your SQL identifier, *password* is your database-access password, and *server-name* is the name of the target application server.

CONNECT Information Shown in Message Files

The authorization ID specified in the last CONNECT statement processed by the database manager is the authorization ID on which the database manager bases its authorization checking for subsequent Database Services Utility processing. When the Database Services Utility displays the CONNECT statement in the message file, the password is suppressed. The authorization ID is shown in double quotation marks ("); for example:

```
CONNECT "ANNETTE" IDENTIFIED BY ********
```

If the Database Services Utility detects an error in the CONNECT statement, the original input line is not displayed. Instead, the Database Services Utility displays the following in the message file:

```
CONNECT ? IDENTIFIED BY ?
```

Note that you must supply an SQL CONNECT statement in the DB2 Server for VSE input control card file before any other SQL or Database Services Utility command, unless you invoke the Database Services Utility from an application program that has already executed an SQL CONNECT. (“Using the Database Services Utility from Programming Languages” on page 112 describes how to invoke the Database Services Utility from an application program.) Refer to the *DB2 Server for VSE & VM Database Administration* manual for additional information on SQL CONNECT processing.

Suppose you have ACTIVITY tables in two databases, RDB1 and RDB2. To query both of them, type in your control file:

```
CONNECT TO RDB1;
SELECT * FROM ACTIVITY;
COMMIT;
CONNECT authorization-id IDENTIFIED BY
password TO RDB2;
SELECT * FROM ACTIVITY;
```

You would replace *authorization-id* with your SQL identifier and replace *password* with your database-access password.

Identifying and Switching to Another Application Server

To use the SQL CONNECT statement to switch to another application server, type:

```
CONNECT TO server-name;
```

where *server-name* is the name of the application server to which you want to connect.

DB2 Server for VSE

If the CONNECT statement is issued without the identify clause (for example, CONNECT TO RDB1), and the previous LUW ends with a COMMIT or ROLLBACK statement, you are connected to the application server with the same user ID and password that was used in the previous LUW.

DB2 Server for VM

If a CONNECT is not explicitly issued, or is issued without the identify clause (for example, CONNECT TO RDB1), the DB2 Server for VM application requester does an implicit connection when you execute your first SQL statement. The database manager uses the entry in the CMS communications directory file (COMDIR) to give you connect authorization to the application server. If the authorization ID is not resolved from the CMS COMDIR, the database manager uses the VM user ID. In some situations, the user ID received at the target application server is different from your VM user ID. For example, an entry in the CMS COMDIR might change the user ID, or the target system might change it. Refer to the *DB2 Server for VSE & VM Database Administration* for additional information on SQL CONNECT processing.

Identifying the Current User ID and Application Server

When you do not specify options with the CONNECT statement, the system displays the current SQL user ID and application server name. This is a null CONNECT. To enter a null CONNECT, use:

```
CONNECT;
```

If a null CONNECT is issued before a server connection is established by a previous CONNECT statement, a blank user ID and a blank application server-name is returned. If a null CONNECT is issued before a valid user ID is established by a previous CONNECT statement, a blank user ID and the connected server-name is returned.

For further information on the CONNECT statement, refer to the *DB2 Server for VSE & VM SQL Reference*.

SELECT

Output of Query Results

The Database Services Utility writes the results of an SQL SELECT statement (an SQL query) to the DB2 Server for VSE report (SYSLST) or the DB2 Server for VM message file (SYSPRINT).

Specifying a Multiple-Row Query

The use of the SQL SELECT statement is often called a *query* because SELECT statements are the means of extracting information from a database.

The Database Services Utility automatically handles multiple-row query results; you do not have to declare a cursor. Figure 10 on page 23 is an example of pseudocode showing how a query is coded in an application program to return many rows using a cursor. Figure 11 on page 23 shows how the Database Services Utility handles the same multiple-row query.

```
EXEC SQL DECLARE C1 CURSOR FOR
     SELECT PROJNO,PROJNAME
     FROM PROJECT WHERE DEPTNO = 'E21'
     ORDER BY PROJNO
EXEC SQL OPEN C1
EXEC SQL FETCH C1 INTO :NUMBER, :NAME
DO WHILE (SQLCODE=0)
     DISPLAY (NUMBER, NAME)
     EXEC SQL FETCH C1 INTO :NUMBER, :NAME
END-DO
EXEC SQL CLOSE C1
```

Figure 10. Sample Multiple-Row Query in an Application Program

```
SELECT PROJNO,PROJNAME
FROM PROJECT WHERE PROJNO = 'E21'
ORDER BY PROJNO;
               ^
               |
               Required delimiter for
               the Database Services Utility
```

Figure 11. Multiple-Row Query

Specifying a Single Value Query

The Database Services Utility does not support INTO clauses. Figure 12 shows how an application program uses the INTO clause to return a single value. Figure 13 shows how to specify the same single value query in the Database Services Utility.

```
SELECT AVG(BONUS)
INTO :EXTRA
FROM EMPLOYEE
WHERE JOB = 'MANAGER'
```

Figure 12. Sample Single Value Query in an Application Program

```
SELECT AVG(BONUS)
FROM EMPLOYEE
WHERE JOB = 'MANAGER';
```

Figure 13. Single Value Query

SELECT Output Is Identified by Column Name

Column data appearing in SELECT output produced by Database Services Utility processing is identified by column name. Column labels are ignored by Database Services Utility processing.

COMMIT

The Database Services Utility handles logical units of work in almost the same way as application programs. Most Database Services Utility commands or SQL statements implicitly begin a logical unit of work. The following commands and statements, however, do not:

```
   Database Services Utility Commands        SQL Statements

COMMENT                      CONNECT
SET AUTOCOMMIT               COMMIT [RELEASE]
SET ERRORMODE                ROLLBACK [RELEASE]
SET FORMAT
SET ISOLATION
SET LINECOUNT
SET LINEWIDTH
SET UPDATE STATISTICS
```

A logical unit of work continues until you issue an SQL COMMIT statement or a ROLLBACK statement. After the COMMIT or ROLLBACK is processed, another Database Services Utility command or SQL statement begins a new logical unit of work.

If you do not include a COMMIT statement or a ROLLBACK statement, the Database Services Utility treats all the control commands as a single logical unit of work. If all processing is error-free during this single logical unit of work, the changes are committed to the database; if errors occurred, no changes are committed to the database.

Committing Logical Units of Work

The Database Services Utility provides the command SET AUTOCOMMIT ON or OFF. When AUTOCOMMIT is on, the utility performs a COMMIT operation after the successful execution of each command that accesses the database. When AUTOCOMMIT is off (the default mode of processing) or is not specified in your (input) control file, logical units of work are processed as described previously in "COMMIT".

If you want the utility to commit logical units of work, include the following in your (input) control file:

```
SET AUTOCOMMIT ON;
```

Using SQL Comments

SQL comments can be included within SQL statements used in the Database Services Utility and within Database Services Utility commands wherever a separator is valid, as long as the existing Database Services Utility syntax rules are followed. SQL comments are identified by two consecutive hyphens (--) on the same line. The hyphens must not be separated by a space. SQL comments must not be part of a literal, double-byte character set (DBCS) string or quoted identifier. Each SQL comment must be contained on a single line. In the Database Services Utility, an SQL statement must be terminated by a semicolon (;). If a semicolon appears in an SQL comment, however, it does not end the SQL statement. For example,

```
SELECT * FROM T1 --this does not end the SQL statement;

SELECT * FROM T1; -- this ends the SQL statement
```

The following restrictions apply when using SQL comments within Database Services Utility commands:

- SQL comments are not supported in the data portion of a Database Services Utility command. SQL comments that are used improperly are treated as part of the data.
- SQL comments are not allowed in the Database Services Utility COMMENT command or in the ENDDATA subcommand of the DATALOAD command.

Note: The ENDDATA subcommand of the DATALOAD command should not contain any other information.

Querying the Current Status in DB2 Server for VM

To query the current status of a job that is running, use the CMS immediate command SQLQRY by typing #SQLQRY directly from the terminal. The SQLQRY command cannot be used from a Database Services Utility control file.

You can use SQLQRY to determine the application server you are currently connected to. The output you receive from SQLQRY will be similar to that shown in Figure 14. For more information on the SQLQRY command, see the *DB2 Server for VSE & VM Database Administration* manual.

Note: The output from SQLQRY varies depending on the operating environment.

```
15:54:48 * MSG FROM SQLUSER6: Status of Database Conversations on 1999-06-30
15:54:48 * MSG FROM SQLUSER6: EXTNAME = SQLUSER6.1
15:54:48 * MSG FROM SQLUSER6: RDBMS  = SQLRDB1           SQLDS/VM V3.3.0
15:54:48 * MSG FROM SQLUSER6: STATUS = COMM   TIME = 1999-06-30.15:54:38
15:54:48 * MSG FROM SQLUSER6: LUWID  = IBMNET01.*IDENT.45F2ABCD236D42
15:54:48 * MSG FROM SQLUSER6:
15:54:48 * MSG FROM SQLUSER6: EXTNAME = SQLUSER6.2
15:54:48 * MSG FROM SQLUSER6: RDBMS  = IBMSTLDB2         DB2      V2.3.0
15:54:48 * MSG FROM SQLUSER6: STATUS = APPL   TIME = 1999-06-30.15:30:25
15:54:48 * MSG FROM SQLUSER6: LUWID  = IBMNET01.TORLU001.45F2ABCD236DFE
15:54:48 * MSG FROM SQLUSER6: LU     = STLMVS04
15:54:48 * MSG FROM SQLUSER6: TPN    = "6DB     (X'07F6C4C2')
15:54:48 * MSG FROM SQLUSER6:
15:54:48 * MSG FROM SQLUSER6: EXTNAME = SQLUSER6.2
15:54:48 * MSG FROM SQLUSER6: RDBMS  = SQLMACGM          DB2/VM   n/a
15:54:48 * MSG FROM SQLUSER6: STATUS = VRA    TIME = 1999-06-30.15:30:25
15:54:48 * MSG FROM SQLUSER6: TCP/IP = 9.21.4.194       PORT 6100
```

Figure 14. Sample Output from SQLQRY

Canceling a DB2 Server for VM Command

In interactive mode, you can cancel a command before it is completed by typing:

```
SQLHX
```

For example, you can cancel a query that is running and has not returned a query result. The SQLHX command drops the connection to the application server, thus canceling the current command and rolling back the current logical unit of work. If you issued an explicit CONNECT before the SQLHX command, the authorization ID and the name of the application server revert back to those defined by an implicit CONNECT: your VM ID and default application server.

Note: If you are using synchronous APPC/VM communications with the application server (by specifying the SYNCHRONOUS (YES) option when invoking the SQLINIT EXEC), the SQLHX command does **not** cancel the Database Services Utility command that is running.

Exiting from the Database Services Utility

In DB2 Server for VSE, you exit from the utility automatically after all the commands in the input control card file have been processed.

In DB2 Server for VM, if you use the Database Services Utility and supply a control file with the SYSIN option, you exit from the utility automatically after all the commands in the control file have been processed. If you use the utility by typing `SQLDBSU` and do not supply a control file, you are using the utility interactively. To exit, type the command `EXIT`. Any uncommitted work is committed, and you exit from the utility. You then return to CMS.

Chapter 2. Loading Data with the Database Services Utility

This chapter explains how to load data into a DB2 Server for VSE & VM table with the DATALOAD command. The data you are loading can be separate from the DATALOAD command or embedded in the command. You can also load data of different formats, such as DECIMAL, GRAPHIC, or CHARACTER.

This chapter also describes general loading procedures, such as how to load null values or special register values. Moreover, you can work more efficiently by loading data into multiple tables, combining records to load several table rows, or committing work while loading data. If an error stops the DATALOAD processing, you can restart the loading process. Finally, this chapter describes how statistics are collected during or after the utility has loaded the data.

DATALOAD Command Components

The DATALOAD command allows you to load rows into existing DB2 Server for VSE & VM tables from data contained in a sequential input file that was created by processing external to the database manager or by the Database Services Utility's DATAUNLOAD processing. You can load data into DB2 Server for VSE & VM application servers, as well as into other application servers that support DRDA flow.

In general, each input data record used for DATALOAD processing contains data for a row of a table. Input data records can reside in a sequential file or can be embedded within the (input) control file. The DB2 Server for VM input data file is typically a CMS file, but it can be a virtual reader file or any tape or DASD file supported by CMS OS/QSAM.

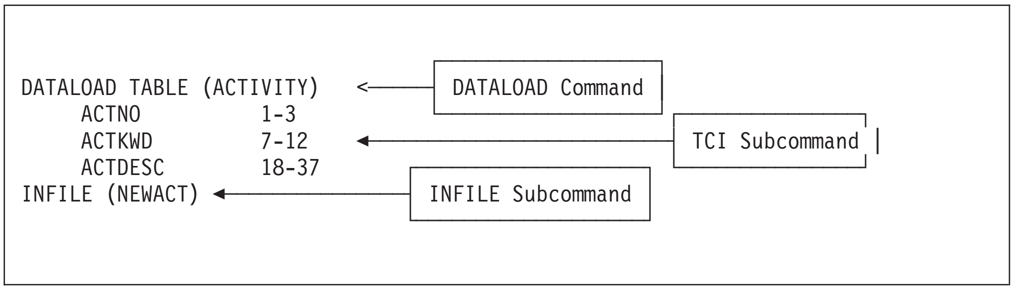
Note: You should not use a sequential access method (SAM) file produced by Database Services Utility UNLOAD processing as input to DATALOAD processing. An error condition can result. Use the RELOAD command instead to load a file produced by the UNLOAD command.

The DATALOAD command and its subcommands can:
- Identify the tables to be loaded
- Describe the data fields in the input records
- Relate table column names to the input record data fields
- Identify the source of the input records.

You must complete the DATALOAD command on a single record; do not continue it on a second input record. The record immediately following a DATALOAD command must contain a Table Column Identification (TCI) subcommand. If, for example, you want to load data into 10 columns of a table, the first input record would contain the DATALOAD command, and the next 10 input records would contain TCI subcommands.

The other subcommands used with the DATALOAD command are INFILE and ENDDATA. The INFILE subcommand identifies the input data file or, when followed by an asterisk (*), identifies that the data is in the (input) control file and immediately follows the subcommand. You use the ENDDATA command to signal the end of user-supplied data; you do not need it if the input data is in a separate file.

Figure 15 illustrates a DATALOAD command followed by three TCI subcommands and an INFILE subcommand. Because the input data is contained in the file NEWACT, the ENDDATA command is not used in this example.



```
DATALOAD TABLE (ACTIVITY)
     ACTNO          1-3
     ACTKWD         7-12
     ACTDESC        18-37
INFILE (NEWACT)
```

Figure 15. DATALOAD Command with Subcommands

The DATALOAD command identifies the table that you want to load the data into (ACTIVITY). This table is sometimes referred to as the target table.

The next three records identify the names of the columns in the ACTIVITY table into which you want to insert data. These records are TCI subcommands. The numbers in the subcommands represent the positions where the data exists on the input records (that is, they identify the input data fields).

Note: If you are not loading data into the column to the extreme right of the table, add a TCI subcommand for that column and set it to null. By identifying the last table column, you avoid space problems in the future when you update the rows loaded into the table. Enough space is allocated in the table to include the column farthest to the right that you specified with the TCI subcommands.

INFILE identifies the file where the input data is located (NEWACT). Figure 16 on page 29 and Figure 17 on page 30 illustrate how the above DATALOAD command sequence relates to the target table and the input file.

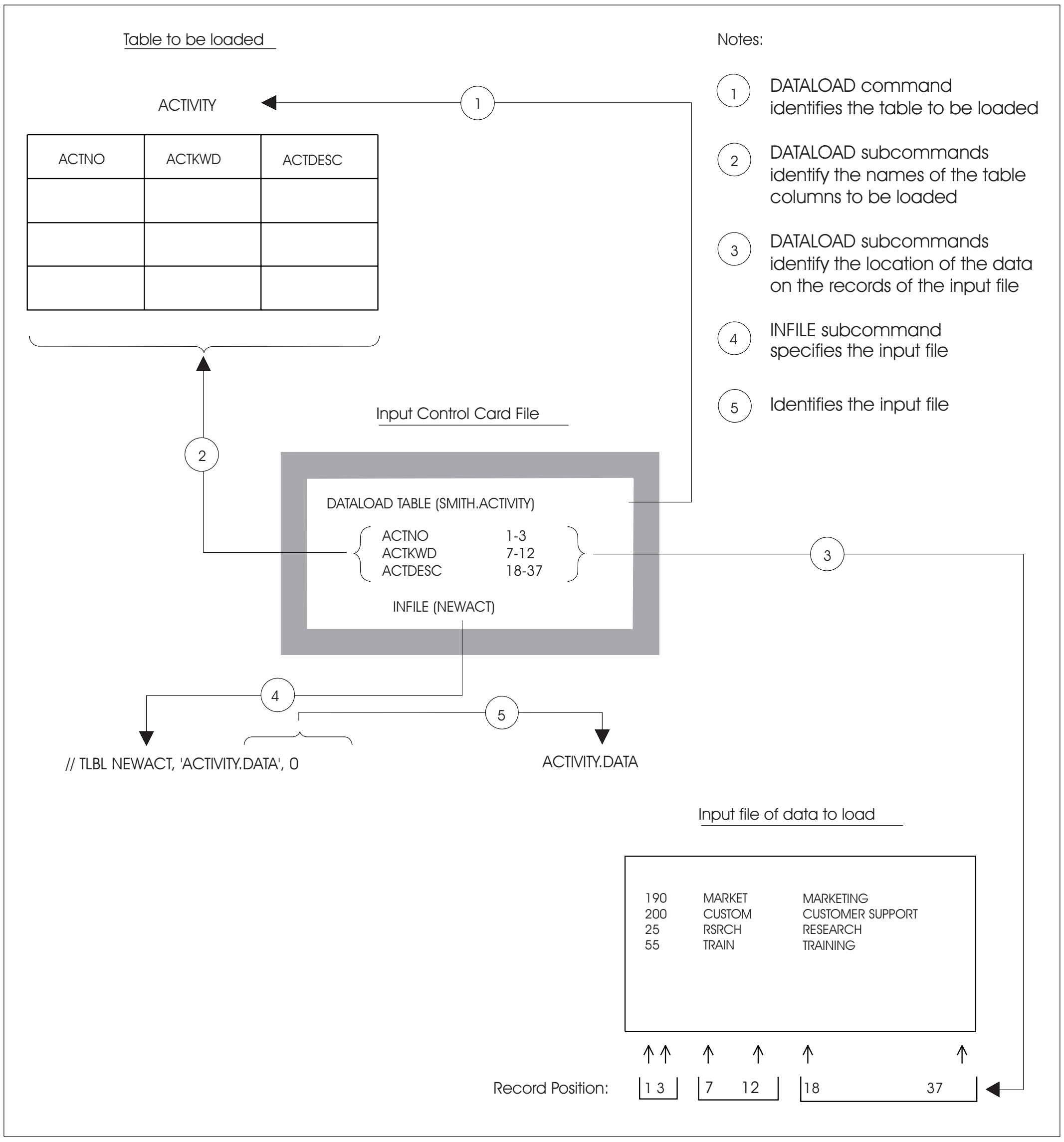


Table to be loaded
ACTIVITY
ACTNO
ACTKWD
ACTDESC
Notes:
1 DATALOAD command identifies the table to be loaded
2 DATALOAD subcommands identify the names of the table columns to be loaded
3 DATALOAD subcommands identify the location of the data on the records of the input file
4 INFILE subcommand specifies the input file
5 Identifies the input file
Input Control Card File
DATALOAD TABLE (SMITH.ACTIVITY)
ACTNO 1-3
ACTKWD 7-12
ACTDESC 18-37
INFILE (NEWACT)
// TLBL NEWACT, 'ACTIVITY.DATA', 0
ACTIVITY.DATA
Input file of data to load
190 MARKET MARKETING
200 CUSTOM CUSTOMER SUPPORT
25 RSRCH RESEARCH
55 TRAIN TRAINING
Record Position:
1 3
7 12
18 37

Figure 16. Schematic Representation of the DB2 Server for VSE DATALOAD Command

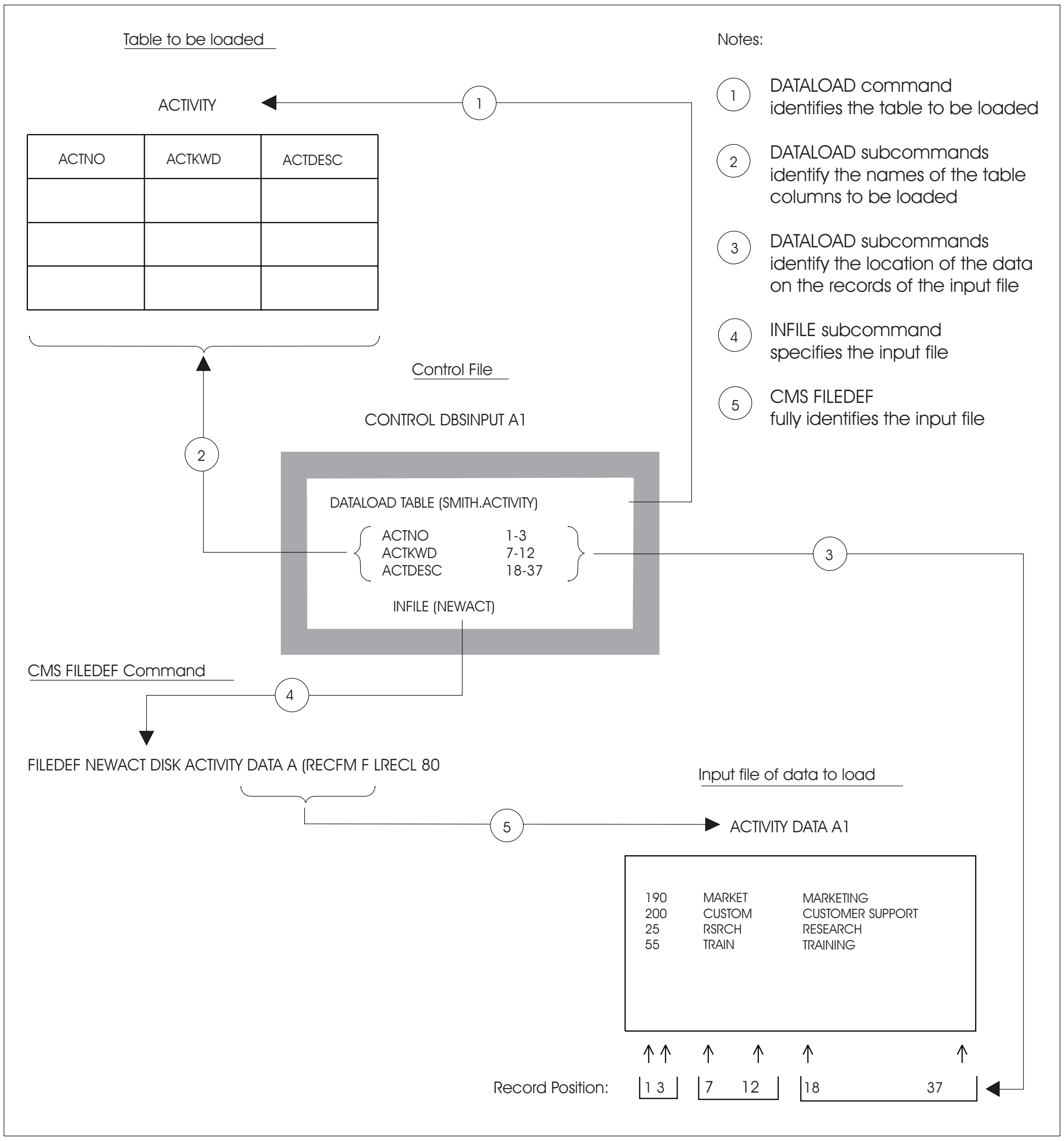


Table to be loaded
ACTIVITY
ACTNO
ACTKWD
ACTDESC
Notes:
1 DATALOAD command identifies the table to be loaded
2 DATALOAD subcommands identify the names of the table columns to be loaded
3 DATALOAD subcommands identify the location of the data on the records of the input file
4 INFILE subcommand specifies the input file
5 CMS FILEDEF fully identifies the input file
Control File
CONTROL DBSINPUT A1
DATALOAD TABLE (SMITH.ACTIVITY)
ACTNO 1-3
ACTKWD 7-12
ACTDESC 18-37
INFILE (NEWACT)
CMS FILEDEF Command
FILEDEF NEWACT DISK ACTIVITY DATA A (RECFM F LRECL 80
Input file of data to load
ACTIVITY DATA A1
190 MARKET MARKETING
200 CUSTOM CUSTOMER SUPPORT
25 RSRCH RESEARCH
55 TRAIN TRAINING
Record Position: 1 3 7 12 18 37

Figure 17. Schematic Representation of the DB2 Server for VM DATALOAD Command

As with SQL INSERT statements, all columns of a table do not have to be specified for DATALOAD processing. Specifying the last column of a table is recommended to avoid problems when updating the rows in the future. If a table column is omitted, however, the column must be defined to permit nulls. If this rule is violated, SQL and Database Services Utility error messages are generated, and DATALOAD processing is not performed.

DB2 Server for VSE

NEWACT is a tape file, because that is the default device type. You must specify INFILE (NEWACT PDEV(DASD) BLKSZ(2048)) if NEWACT is located on a direct access storage device (DASD).

DATALOAD Procedures

You cannot mix other Database Services Utility commands or SQL statements within the DATALOAD command and its subcommands. The input data file for this utility is a general-use programming interface. See "Programming Interface Information" on page 257 for a definition of general-use programming interfaces.

Authorization

DB2 Server for VSE & VM authorization checking prevents you from loading a table if you do not have proper authority. You must have INSERT and SELECT privileges on the tables affected by the DATALOAD command.

Using the DATALOAD Command with a Separate Data Input File

(Input) Control File and Separate Data File

Use the following procedure as a standard method of constructing and implementing the DATALOAD command. Variations on this procedure appear throughout this chapter.

Assume that you have a separate (input) control file and data file. Your sequential access method (SAM) data file exists already, but you want to issue a DATALOAD command to insert rows into a certain table.

Proceed as follows for DB2 Server for VSE:

1. Provide the following Database Services Utility command:
   ```
   DATALOAD TABLE (table-name)
   ```
   where *table-name* is the name of the table that you want to load with data.
2. Put a TCI subcommand on the next record:
   ```
   column-name startpos-endpos data-type
   ```
   where *column-name* is the name of the table column, *startpos* is the first character position in the input record, *endpos* is the last position in the input record, and *data-type* is the data format of the input values. The default data type is character (CHAR).
3. Repeat the preceding step for each table column into which data is to be inserted. Any table column that you are not loading data into must allow null values.
4. On the next record, put:
   ```
   INFILE (ddname)
   ```
   where *ddname* identifies the input file. Use the same *ddname* in a TLBL or DLBL statement, depending on whether the data is stored on tape or in a DASD file.
5. Submit the job to run.

Proceed as follows for DB2 Server for VM:

1. Issue the SQLINIT command to initialize the user machine. If you have already done this, proceed to Step 2.

2. Create a control file to contain the DATALOAD command, which you construct in the following steps. See "Working with a Control File in DB2 Server for VM" on page 13 for detailed information on creating a control file.
3. Type the command name:
   ```
   DATALOAD TABLE (table-name)
   ```
   where *table-name* is the name of the table.
4. Enter the first TCI subcommand. On a new line, type:
   ```
   column-name startpos-endpos data-type
   ```
   where *column-name* is the name of the table column, *startpos* is the first character position in the input record, *endpos* is the last position in the input record, and *data-type* is the data format of the input values. The default data type is character (CHAR).
5. Repeat the preceding step for each table column into which data is to be inserted. Any table column that you are not loading data into must allow null values.
6. On a new line, type:
   ```
   INFILE (ddname)
   ```
   where *ddname* identifies the input file.
7. Store the control file.
8. In CMS, specify the necessary FILEDEF statements. When you specify the FILEDEF statement for the input data file, use the same *ddname* that you assign to the INFILE in this procedure. For general information about FILEDEFs, see "Using File Definitions" on page 14. For command-specific information, see "Using File Definitions with the DB2 Server for VM DATALOAD Command" on page 35.
9. Issue the SQLDBSU EXEC command to run the Database Services Utility. If you did not specify FILEDEFs for the control and message files, use the default values in the SQLDBSU EXEC. For more information on the SQLDBSU EXEC, see "Using the SQLDBSU EXEC" on page 15.

Using a Workstation as a DB2 Server for VM Control File: You can also insert rows into a table by using your workstation as a control file. To do so, follow the standard procedure given in "(Input) Control File and Separate Data File" on page 31 for constructing a DATALOAD command, but enter the information in the following order:
1. In CMS, specify the necessary FILEDEFs.
2. Specify an SQLDBSU EXEC statement that defines SYSIN as T.
3. When the DB2 Server for VSE & VM command entry panel appears, enter the DATALOAD command, TCI subcommands, and INFILE subcommand.

Using the DATALOAD Command with Embedded Data

(Input) Control File with Embedded Data

The data to load need not be in a separate file; you can include it with the Database Services Utility commands in the (input) control file. Figure 18 on page 33 shows a DATALOAD command with data following the INFILE subcommand.

```
DATALOAD TABLE (ACTIVITY)
     ACTNO        1-3
     ACTKWD       5-10
     ACTDESC      12-31
INFILE (*)
190 MARKET MARKETING
200 CUSTOM CUSTOMER SUPPORT
25  RSRCH  RESEARCH
55  TRAIN  TRAINING
ENDDATA
```

Figure 18. DATALOAD Command with Embedded Data

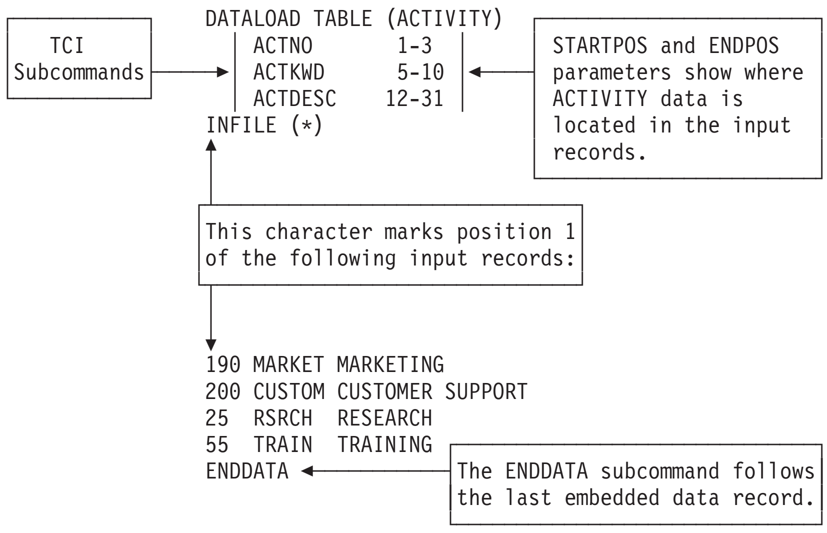
The asterisk parameter (*) of the INFILE subcommand indicates that input data immediately follows. When such input data is included with a DATALOAD command, mark the end of it with an ENDDATA subcommand.

The ENDDATA subcommand is valid only when the previous Database Services Utility command statement processed is INFILE (*).

Match TCI and Data Positions

Enter records following an INFILE(*) subcommand so that the character positions of the data correspond to the *startpos* and *endpos* parameter specifications of the applicable TCI subcommand.

The following example shows that data inserted in the ACTNO column is specified as occupying positions 1 through 3 of the input data records. Because *INFILE* begins at position 1, you can align position 1 of each data record under the *I* of *INFILE*. The *INFILE* does not have to begin in position 1, but because it does in this example, you can use it as a guide to position your data records.


TCI Subcommands
STARTPOS and ENDPOS parameters show where ACTIVITY data is located in the input records.
This character marks position 1 of the following input records:
The ENDDATA subcommand follows the last embedded data record.

```
DATALOAD TABLE (ACTIVITY)
    ACTNO       1-3
    ACTKWD      5-10
    ACTDESC    12-31
INFILE (*)

190 MARKET MARKETING
200 CUSTOM CUSTOMER SUPPORT
25  RSRCH  RESEARCH
55  TRAIN  TRAINING
ENDDATA
```

Use a column scale or ruler when entering DB2 Server for VM embedded data to align data fields.

To load embedded data, follow the standard procedure in "(Input) Control File and Separate Data File" on page 31, but construct the INFILE subcommand as follows:

1. On a new (VSE) record or (VM) line, type:
   ```
   INFILE (*)
   ```
   where (*) indicates that data follows immediately.
2. On the next record or line, enter the first data record. Align the character positions to match the positions of the *startpos-endpos* values. Repeat this step for each succeeding data record.
3. When you have finished providing data records, type the following on a new record or line:
   ```
   ENDDATA
   ```

Data Format Support

You can store data in sequential files in different data formats. A file created using a file editor is usually stored in CHARACTER data format. When a file is produced by a program, it is possible for the program to store data in one or more of the following data formats: DECIMAL, FIXED, FLOAT, ZONED, CHARACTER, DATE, TIME, TIMESTAMP, or GRAPHIC data formats.

The Database Services Utility supports data stored in any of the previously mentioned data formats. When loading data, the utility automatically converts the input data to the data type of the particular column of the target table.

Sometimes, rather than using character data, an application program generates fixed-point binary, floating-point binary, or packed decimal data. You still use the utility to load the data into a table, but you need to specify that the input data is no longer CHAR data type. The TCI subcommand has the optional data type parameter for this purpose. The following example illustrates the use of the TCI subcommand's data type parameter FIXED, which indicates that the input data type is fixed-point binary.

```
DATALOAD TABLE (ACTIVITY)
     ACTNO        1-3    FIXED
     ACTKWD       4-9
     ACTDESC      10-29
INFILE (NEWACT PDEV(DASD) BLKSZ)    <------DB2 Server for VSE
INFILE (NEWACT)                     <------DB2 Server for VM
```

DATALOAD converts the fixed-point binary data in columns 1-3 to SMALLINT data type in the table because the corresponding column is defined as SMALLINT. The data you are loading can be only one or 2 bytes; DATALOAD cannot convert 4-byte fixed data to SMALLINT. If you have 4-byte data, the table column you load has to be defined as INTEGER.

For further qualifying information, see "DATALOAD Data Conversion Summary" on page 165.

JCL for the DB2 Server for VSE DATALOAD Command

When you use a separate data input file with the DATALOAD command, you need to define that file through JCL statements. Use the information in this section when you construct a job for a Database Services Utility command that requires a data definition name (*ddname*). See Figure 19 on page 35 for an example of JCL statements that define an input data file.

```
// JOB DBS Utility Dataload Example
// EXEC PROC=ARIS75PL
// DLBL NEWACT, 'ACT.DATA',0
// EXTENT SYS006,SQLWK1,1,0,57,76
// ASSGN SYS006,150
// EXEC PGM=ARIDBS,SIZE=AUTO
DATALOAD TABLE (ACTIVITY)
     ACTNO       1-3    FIXED
     ACTKWD      4-9
     ACTDESC     10-29
INFILE (NEWACT PDEV(DASD) BLKSZ)
/*
/&
```

Figure 19. Example of JCL Statements to Define an Input Data File

The DATALOAD command uses the *ddname* NEWACT, which refers to the input data file ACT.DATA in the DLBL statement.

Using File Definitions with the DB2 Server for VM DATALOAD Command

When you use a separate data input file with the DATALOAD command, you need to define that file with a FILEDEF statement. Even if you want to type in a few rows of data from your terminal, you must use the FILEDEF statement to specify that the input is coming from your terminal. If the data is in a virtual reader file, you can use the FILEDEF to specify the spooled reader. The only situation where you do not need a supplementary FILEDEF statement is if the input data is in the control file.

Use the information in the following section when you construct the FILEDEF statement for the input data file.

FILEDEFs Supporting DATALOAD Command Processing

In the CMS FILEDEF command that defines the Database Services Utility's DATALOAD command input data file, all record format specifications are supported except for carriage-control characters and undefined format. (Do not use A, M, or U in your RECFM specification.) If you define CMS input files as VS or VBS, DATALOAD processing changes the record format to VB.

If you define CMS input files with variable-length spanned records (RECFM=VS or VBS), you must use the file mode number **4**. For example:

FILEDEF *ddname* DISK *filename filetype* A**4** (*options*

If the DATALOAD input data file contains records with more than 32 760 positions of data, you can do one of the following:

1. Use VS or VBS records. Specify as options only the RECFM and Block size (BLOCK or BLKSIZE) parameters in the FILEDEF command defining the data file. (The LRECL specification does not apply and would not be overridden if specified.)
2. Use F or V records if you are running under CMS 15 or later and the DATALOAD input data file contains records with less than 65 536 positions of data.

A sample FILEDEF command defining a CMS file for DATALOAD command processing is:

```
FILEDEF DBSFILE DISK DBSFILE DATA A (RECFM F LRECL 80
```

where DBSFILE is the name of the data input file as it is referred to in your program. For more information on FILEDEF parameters and options, see Appendix B, “FILEDEF Command Syntax and Notes,” on page 249.

Use the Same File Definition for DATALOAD as for DATAUNLOAD

If the input data file was created by DATAUNLOAD processing, then the CMS FILEDEF command that defines the DATALOAD input data file should be identical to the information in the FILEDEF command used when the file was created by DATAUNLOAD processing.

General Loading Procedures

Comparison Operators

In the procedures that follow, you specify a comparison operator. The following comparison operators are supported by the Database Services Utility:
Comparison Operators

```
=     Equal to
¬=    Not equal to
<>    Not equal to
>     Greater than
>=    Greater than or equal to
<     Less than
<=    Less than or equal to
```

Loading Null Values

Suppose that you are loading data to a table named DEPARTMENT in columns DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT, but you do not have a manager for every department. You still want to insert the DEPTNO, DEPTNAME, and ADMRDEPT into the database. For those departments that do not have a manager, you want to insert a null value. Use the TCI subcommand’s null-current-clause. Figure 20 illustrates one way to code the MGRNO TCI to load null values.

```
DATALOAD TABLE (DEPARTMENT)
     DEPTNO       1-3
     DEPTNAME     5-40
     MGRNO        42-47  NULL IF POS (42-47)='      '    <——6 blanks.
     ADMRDEPT     49-51
INFILE (NEWDEPT)
```

Figure 20. TCI Subcommand with a Null Clause

A translation of this clause is: make the corresponding table field null if input-record positions 42 through 47 are blank.

To specify a null condition in the TCI subcommand, do the following:

1. Leave one or more spaces after the TCI subcommand’s *endpos* (or data type) parameter, and include:

   `NULL IF POS (`*`startpos`*`-`*`endpos`*`)` *`operator constant`*

   where *startpos* is the first character position in the input record that contains the comparison string, *endpos* is the last position of the string, *constant* is the value against which the string at position *startpos-endpos* is to be compared, and *operator* is a comparison operator. (See “Comparison Operators” on page 36 for a list of comparison operators.) Do not put spaces within the brackets.

2. Proceed to the next DATALOAD subcommand.

Note: The positions checked for the null value need not be in the same positions occupied by the data field for the column. You can assign the null value to a column depending on any convention you choose. For example, to set the MGRNO column to NULL whenever a blank exists in position 11 of the DEPTNAME column, code the MGRNO TCI as follows:

```
MGRNO 42-47 NULL IF POS (11) = ' '
```

However, if the positions of the data fields and the positions specified by the *startpos* and *endpos* in the subcommand’s null-current-clause overlap, your data may be overlaid. For more information on the null-current-clause, see page 155.

Alternative Method

Another way to insert null values into the database for new rows is by omitting a TCI subcommand for that column:

```
DATALOAD TABLE (DEPARTMENT)
     DEPTNO       1-3
     DEPTNAME     5-40
     ADMRDEPT     49-51
INFILE (NEWDEPT)
```

In the above example, there is no TCI subcommand for the MGRNO column of the DEPARTMENT table. For each new row inserted, the MGRNO field is null. The columns that the utility loads null values into must permit nulls.

Loading CURRENT DATE, CURRENT TIME, and CURRENT TIMESTAMP Values

The database manager supports the following date and time formats: International Standards Organization (ISO) form, IBM Standard for Europe form (EUR), IBM Standard for the U.S. form (USA), Japanese Industrial Standard Christian Era form (JIS), and an installation-defined form (LOCAL).

Suppose that someone created the following table with the SQL CREATE TABLE statement and you want to load data into the table.

```
CREATE TABLE PAYABLE
    (COMPANY       CHAR(20),
     PAYMENT_DUE   DATE,
     AMOUNT        DEC(9,2));
```

Some of the companies in the PAYABLE table are in arrears with their payments. For these organizations, you want the payment due date to be today’s date. Use the TCI subcommand’s null-current-clause. Figure 21 on page 38 shows one way to code the TCI subcommand PAYMENT_DUE to load the current date. (This example does not use the sample tables; therefore, do not attempt to process it.)

```
DATALOAD TABLE (PAYABLE)
  COMPANY      1-20
  PAYMENT_DUE  22-31 CURRENT DATE IF POS(22-30) = 'IMMEDIATE'
  AMOUNT       35-45
INFILE(*)
VESUVIUS, INC.       2000-05-01   5000.00
ATLANTIS CO.         28.05.1999   3820.00
TITANIC LTD.         IMMEDIATE    7250.00
SKY INC.             05/22/1999    300.00
ENDDATA
```

Figure 21. TCI Subcommand with a Current-Date Clause

A translation of the current date clause is: load the corresponding table field with the current date if input-record positions 22 through 30 contain the string *IMMEDIATE*.

To specify a current date in the TCI subcommand, proceed as follows:

1. Leave one or more spaces after the TCI subcommand's ENDPOS (or DATATYPE) parameter, and include:

   `CURRENT DATE IF POS (`*`startpos-endpos`*`)` *`operator constant`*

   where *startpos* is the first character position in the input record that contains the comparison string, *endpos* is the last position of that string, *constant* is the value against which the string *startpos-endpos* is to be compared, and *operator* is a comparison operator. See "Comparison Operators" on page 36 for a list of comparison operators.
2. Proceed to the next DATALOAD subcommand.

To load current times and timestamps, replace CURRENT DATE with CURRENT TIME or CURRENT TIMESTAMP in Step 1 of this procedure. For example, in Figure 21 on page 38, you could replace CURRENT DATE with CURRENT TIME if the PAYMENT_DUE column were TIME data type.

Note: The current date, current time, and current timestamp value is acquired by Database Services Utility at the start of the DATALOAD command processing, and will not change throughout the DATALOAD command processing.

Loading Data into Multiple Tables

You can load the same data records into more than one table, or load different data records in the same input file into their respective tables. When you are loading data into more than one table, the Database Services Utility automatically performs an UPDATE STATISTICS (unless a SET UPDATE STATISTICS OFF command has been issued) **after** the DATALOAD command processing is completed successfully.

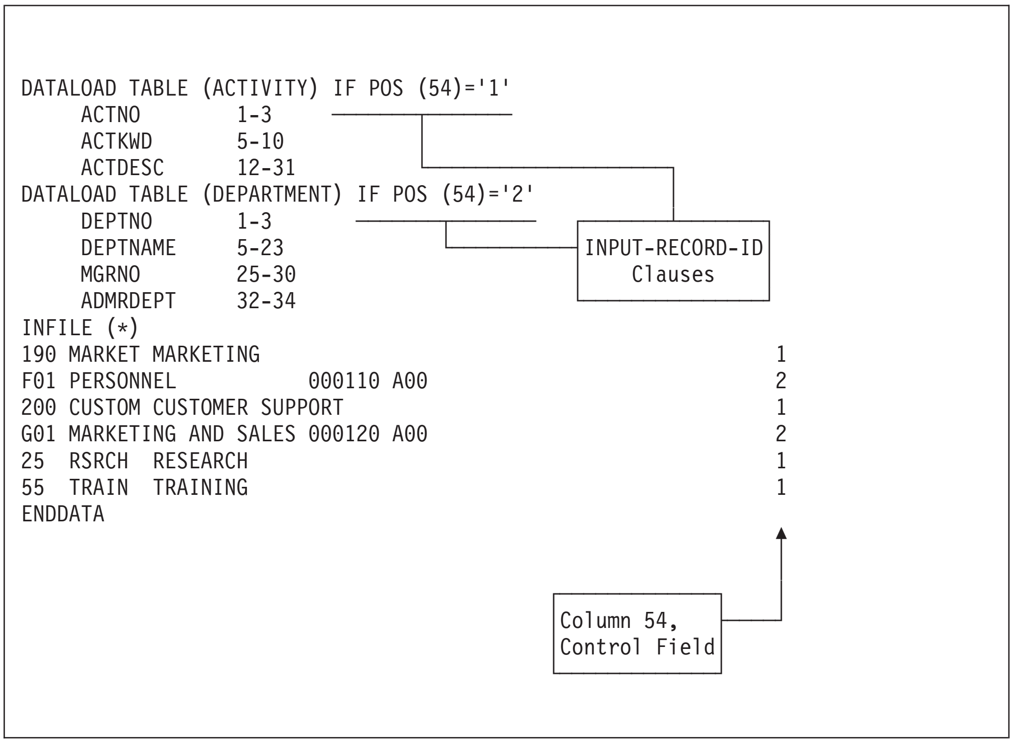
Loading Mixed INFILE Records into the Correct Tables

If you had to load data into two tables, you would probably prepare two DATALOAD commands that could be run either separately or consecutively in the same (input) control file. This is shown in Figure 22 on page 39.

```
DATALOAD TABLE (ACTIVITY)
     ACTNO        1-3
     ACTKWD       5-10
     ACTDESC      12-31
INFILE (*)
190 MARKET MARKETING
200 CUSTOM CUSTOMER SUPPORT
25  RSRCH  RESEARCH
55  TRAIN  TRAINING
ENDDATA
DATALOAD TABLE (DEPARTMENT)
     DEPTNO       1-3
     DEPTNAME     5-23
     MGRNO        25-30
     ADMRDEPT     32-34
INFILE (*)
F01 PERSONNEL           000110 A00
G01 MARKETING AND SALES 000120 A00
ENDDATA
```

Figure 22. Separate DATALOAD Commands Run Successively

If, for some reason, the input data records for two tables were mixed in one data group, you could run the single data group against multiple DATALOAD commands. This is possible with the use of the DATALOAD statement's input-record-id clause. Figure 23 shows two DATALOAD commands that share one INFILE(*) subcommand. Both DATALOAD commands have input-record-id clauses that specify the records that belong to each table.



```
DATALOAD TABLE (ACTIVITY) IF POS (54)='1'
     ACTNO        1-3
     ACTKWD       5-10
     ACTDESC      12-31
DATALOAD TABLE (DEPARTMENT) IF POS (54)='2'
     DEPTNO       1-3
     DEPTNAME     5-23
     MGRNO        25-30
     ADMRDEPT     32-34
INFILE (*)
190 MARKET MARKETING                                           1
F01 PERSONNEL           000110 A00                             2
200 CUSTOM CUSTOMER SUPPORT                                    1
G01 MARKETING AND SALES 000120 A00                             2
25  RSRCH  RESEARCH                                            1
55  TRAIN  TRAINING                                            1
ENDDATA
```

Figure 23. DATALOAD Commands Sharing a Data File

Translations of the input-record-id clauses are:

- ACTIVITY Table. If position 54 of an input data record contains a *1*, load the ACTIVITY table with that record.

- DEPARTMENT Table. If position 54 of an input data record contains a *2*, load the DEPARTMENT table with that record.

To load separate tables with mixed input data from a single file, proceed as follows:

1. Leave one or more spaces after the DATALOAD command's table-name parameter, and include:

   ```
   IF POS (startpos-endpos) operator constant
   ```

   where *startpos* is the first character position in the control field, *endpos* is the last position in that field, *constant* is the value against which the string *startpos-endpos* is to be compared, and *operator* is a comparison operator. See "Comparison Operators" on page 36 for a list of comparison operators.
2. Enter each TCI subcommand on a new line by including:

   ```
   column-name startpos-endpos data-type
   ```

   where *column-name* is the name of the table column; *startpos* is the starting position of the input record; *endpos* is the last position in the input record; and *data-type* is the data format of the column values. If the data type is character (*CHAR*), you can omit it.
3. Repeat the previous two steps for each table to be loaded (that is, for each DATALOAD command).
4. Continue with command and data entry.

Note: Ensure that the control field occupies the same position or positions in each of the data records of the input file.

Loading a Single Record into Several Tables

When the Database Services Utility loads records from a mixed input file into multiple tables, each data record is inserted into a particular table only. You can also write DATALOAD commands so that a single input data record can be a source of rows in more than one table. For example, suppose that you want to expand your activities for each project. Each activity number added to the ACTIVITY table has a corresponding activity number added to the PROJ_ACT table. To use a single input data record to make entries in these two tables, you could code utility commands as shown in Figure 24.

```
DATALOAD TABLE(ACTIVITY)
     ACTNO         1-3
     ACTKWD        5-10
     ACTDESC       12-27
DATALOAD TABLE(PROJ_ACT)
     PROJNO        29-34
     ACTNO         36-38
     ACSTAFF       40-43
     ACSTDATE      45-54
     ACENDATE      56-65
INFILE (*)
190 MARKET MARKETING        AD3100 190 0.50 1999-01-02 1999-04-30
200 CUSTOM CUSTOMER SUPPORT OP2000 200 1.50 1998-03-01 1999-12-31
55  TRAIN  TRAINING         IF2000 55  1.00 1999-02-01 1999-09-05
ENDDATA
```

Figure 24. Individual Records Supplying the Same Activity Number to Two Tables

In Figure 24, each DATALOAD command has its own set of TCI subcommands that point to unique positions on the same input record.

Sixteen DATALOADs Sharing One INFILE

When more than one DATALOAD command (without input-record-id clauses) precedes an INFILE subcommand, the input data identified by the INFILE subcommand is used for all tables identified in the DATALOAD commands. A maximum of 16 DATALOAD commands can precede an INFILE subcommand. You can load 16 tables at the same time.

In Figure 24, the activity number is repeated twice (once for the ACTIVITY table and once for the PROJ_ACT table) in each input data record. Eliminate the need for duplicate fields by using TCI subcommands for each of the tables that point to the same physical location in the input record, as shown in Figure 25.

```
DATALOAD TABLE(ACTIVITY)
     ACTNO         1-3    ◄────────
     ACTKWD        5-10            These two subcommands
     ACTDESC       12-27           point to the same
DATALOAD TABLE(PROJ_ACT)           physical location in
     PROJNO        29-34           the input data records.
     ACTNO         1-3    ◄────────
     ACSTAFF       40-43
     ACSTDATE      45-54
     ACENDATE      56-65
INFILE (*)
190 MARKET MARKETING        AD3100     0.50 1989-01-02 1989-04-30
200 CUSTOM CUSTOMER SUPPORT OP2000     1.50 1989-03-01 1995-12-31
55  TRAIN  TRAINING         IF2000     1.00 1989-02-01 1989-09-05
ENDDATA
```

Figure 25. TCIs in Individual DATALOADs That Point to the Same Location

To use similar TCI subcommands in multiple DATALOAD TABLE commands, proceed as follows:

1. Find the correct *startpos-endpos* value for each TCI subcommand that is common to more than one table.
2. Include the TCI statement in each affected DATALOAD command, and give the same *startpos-endpos* value for each.

Combining Records to Load Multiple Table Rows

Usually, one input data record provides information for one table row. This is illustrated in Figure 26 on page 42.

```
DATALOAD TABLE(ACTIVITY)
     ACTNO       1-3
     ACTKWD      5-10
     ACTDESC     12-27
INFILE (*)
190 MARKET MARKETING
200 CUSTOM CUSTOMER SUPPORT
25  RSRCH  RESEARCH
55  TRAIN  TRAINING
ENDDATA
```

Figure 26. Normal Relationship: One Record for One Row

Using multiple DATALOAD commands, however, you can load more than one table row with each input data record. Figure 27 presents a way of using multiple DATALOAD statements to load combined records into the same table.

```
DATALOAD TABLE(ACTIVITY)
     ACTNO       1-3
     ACTKWD      5-10
     ACTDESC     12-27
DATALOAD TABLE(ACTIVITY)
     ACTNO       29-31
     ACTKWD      33-38
     ACTDESC     40-55
INFILE(*)
190 MARKET MARKETING       200 CUSTOM CUSTOMER SUPPORT
25  RSRCH  RESEARCH        55  TRAIN  TRAINING
ENDDATA
```

Figure 27. Combined Records: Each for Two Rows

In the above example, two rows are inserted into the ACTIVITY table for each data record that is read.

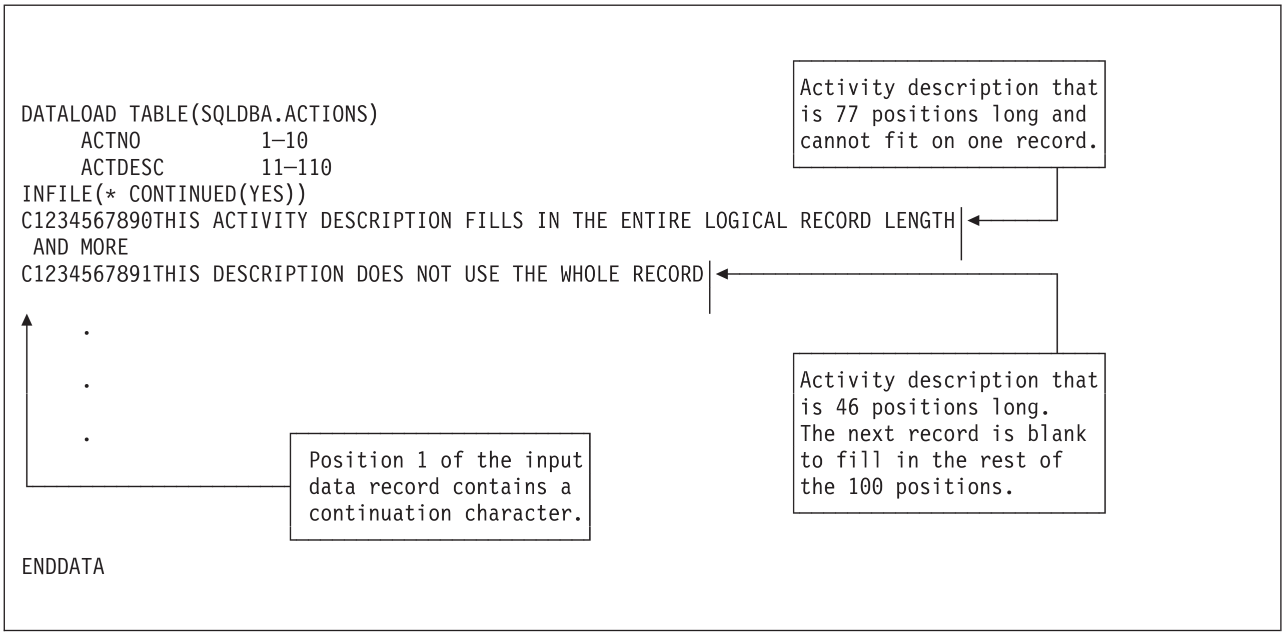
To load combined records into the same table, proceed as follows:

1. Decide how many simple data records the combined record should contain; this is the number of DATALOAD statements needed. Provide the required number of DATALOAD TABLE commands, each with the same table name.
2. Provide each DATALOAD TABLE command with a set of identical TCI subcommands.
3. Determine the correct *startpos-endpos* values for each DATALOAD command set; provide the appropriate value to each TCI statement.
4. In VSE, provide the INFILE(*) data as combined records. In VM, enter the INFILE(*) data as combined records—with each record on a separate line. Ensure that this data is positioned to correspond to the *startpos-endpos* values in the TCI subcommands.

Processing Data That Spans More Than One Input Record

You can load data that is continued onto the next physical record. Figure 28 on page 43 illustrates the command sequence necessary to load data from 80-byte input data records into the columns ACTNO and ACTDESC in the table named SQLDBA.ACTIONS where:

- The column ACTNO is defined with the data type CHAR(10).
- The column ACTDESC is defined with the data type VARCHAR(100).



```
DATALOAD TABLE(SQLDBA.ACTIONS)
     ACTNO          1–10
     ACTDESC        11–110
INFILE(* CONTINUED(YES))
C1234567890THIS ACTIVITY DESCRIPTION FILLS IN THE ENTIRE LOGICAL RECORD LENGTH
 AND MORE
C1234567891THIS DESCRIPTION DOES NOT USE THE WHOLE RECORD

     .

     .

     .

ENDDATA
```

Activity description that is 77 positions long and cannot fit on one record.

Activity description that is 46 positions long. The next record is blank to fill in the rest of the 100 positions.

Position 1 of the input data record contains a continuation character.

Figure 28. Specifying Continued Input Records

The CONTINUED parameter of the INFILE subcommand indicates that continued records are in the input data. If position 1 of an input data record contains a character, the input data is continued onto the next physical input data record. The above example uses a *C* as a continuation character, but you can use any character. The first position of each input file data record is not included in the actual input data, so character positions begin numbering from the second position. Therefore, the physical position 2 of an input data record is referred to as position 1 in the TCI subcommand.

If position 1 of the input data record is blank, the input data is contained on a single logical input record, or it is the terminating row of a continued record. All actual input data records must be at least as long as the highest end-position value specified in a TCI subcommand. In this situation, the ACTDESC field must be 100 characters in length. Therefore, a second physical input line containing all blanks is necessary to extend the ACTDESC field of the second logical input record (beginning with ACTNO=1234567891) to the maximum length value of 100.

If the highest TCI *endpos* value exceeds the input record length, you need to use continued input records. To construct a Database Services Utility command to load continued physical input records into a table, proceed as follows:

For DB2 Server for VSE

1. Define the DATALOAD TABLE and TCI statements in the usual way.
2. Put the INFILE statement with the CONTINUED parameter on the next record:
   ```
   INFILE (* CONTINUED(YES))
   ```
   The CONTINUED parameter must be on the same record as INFILE.
3. Provide the data records:
   ```
   C ...data...
   ```
   where *C* can be the character C or any other character, and *...data...* is the actual input-record data **to a maximum of 79 characters**.

4. Depending on the length of the physical input record, do one of the following:
   - If the physical record length exceeds the logical record length (LRECL=80), continue entering the data, beginning in position 2 of the following input record. Do not put a continuation character on the second record.
   - If the physical record length is less than the logical record length (LRECL=80), leave the following input record blank.

   If you have to enter more than two records for a physical input record, put a continuation character at the beginning of each record except the last one. The blank in position 1 terminates that input record.
5. For each data record, repeat the preceding two steps.
6. Indicate the end of input data. On a new record, put:
   ```
   ENDDATA
   ```

For DB2 Server for VM
1. Define the DATALOAD TABLE and TCI statements in the usual way.
2. Type the INFILE statement as far as the asterisk:
   ```
   INFILE (*
   ```
3. Enter the CONTINUED parameter. On the same line, leave one space; then type:
   ```
   CONTINUED(YES))
   ```
4. Enter the data records. Type:
   ```
   C ...data...
   ```

   where *C* can be the character C or any other character, and *...data...* is the actual input-record data **to a maximum of 79 characters**.
5. Depending on the length of the physical input record, do one of the following:
   - If the physical record length exceeds the logical record length (LRECL=80), continue entering the data, beginning in position 2 of the following input line. Do not put a continuation character on the second line.
   - If the physical record length is less than the logical record length (LRECL=80), leave the following input line blank.

   If you have to enter more than two lines for a physical input record, put a continuation character at the beginning of each line except the last one. The blank in position 1 terminates that input record.
6. For each data record, repeat the preceding two steps.
7. Indicate the end of input data. On a new line, type:
   ```
   ENDDATA
   ```

Committing Work While Loading Data

If the SET AUTOCOMMIT ON command has been issued, the DATALOAD command can specify that the utility issue SQL COMMIT statements periodically during processing. The interval is specified in terms of a specific number of input records processed by DATALOAD. A record is considered to be processed by DATALOAD when it is read and appropriate action is taken. The action is one of the following:
- Skip the record because input record selection criteria are not met
- Insert data from the record into one or more tables.

You indicate the number of records by specifying the COMMITCOUNT(*ccount*) parameter on the INFILE subcommand. Each time that COMMIT processing is to begin, a message (ARI0800I) is written to the message file containing the number of input records processed up to this point. The message is written as a result of COMMITCOUNT processing. You also receive a message (ARI0811I) to inform you that the changes were committed successfully.

When to Use the COMMITCOUNT Parameter

You can use COMMITCOUNT to minimize lock interface with other users of a dbspace or a table. If you set the value of *ccount* low enough, escalation is avoided.

The COMMITCOUNT parameter also helps reduce log space requirements during execution with multiple user mode.

To cause the Database Services Utility to commit work during processing, proceed as follows:

1. On the same record as the INFILE subcommand, leave one or more spaces, then include:
   ```
   COMMITCOUNT(ccount)
   ```
   where *ccount* is a number from 1 to 2,147,483,647.
2. Specify the rest of the DATALOAD TABLE command set.

Specifying a COMMITCOUNT value commits that number of input records to the database as soon as the Database Services Utility has processed them.

Error-Processing Example

Assume that an error occurs in a job for which a COMMITCOUNT value of 1000 has been specified. If the error occurs during the processing of record 99 501 in a 100 000-record file, a ROLLBACK (implicit) command is processed only for the database row inserts performed for the last 500 records (records 99 001 to 99 500).

If any INSERT commands are processed during DATALOAD processing of the first 99 000 records, they have already been committed to the database. Specifying a COMMITCOUNT of 1000 causes COMMIT processing to be done after every 1000 input data records are processed. The last messages in the (VSE) report or (VM) message file are:

```
ARI0800I  ...Begin COMMIT. Input Record Count = 99000
ARI0811I  ...COMMIT of any database changes was successful.
```

Determining the Number of Records Processed

During DATALOAD processing of files containing more than 15,000 data records, a message (ARI8995I) is written to either the VSE operator console or your VM terminal after every 15,000 records to let you know that the job is running normally and that *n* input records have been processed. These messages appear unless:

- You used the INFILE subcommand's COMMITCOUNT parameter and assigned the report or message file to your terminal.
- The number of records is fewer than 15,000.

Skipping Bad Records

A bad data record is one that:

- Contains a data field that cannot be converted to the data type of its target column

- Contains a data value that causes an SQL INSERT data-conversion or a nonunique-column-value error.

If the COMMITCOUNT parameter is specified with AUTOCOMMIT ON, and ERRORMODE CONTINUE processing is in effect, DATALOAD processing skips bad data records. Processing continues under the following circumstances:
- An error identified by ARI0866E occurs.
- An SQL INSERT error identified by SQLCODE -405, -424, -530, -802, or -803 occurs followed by message ARI0862E, and insert blocking is not in effect.

Insert blocking is not in effect under the following conditions:
- Database Services Utility is running with single user mode.
- Database Services Utility is running with multiple user mode but was preprocessed with the NOBLOCK option.
- Insert blocking is suppressed by the database manager.

A bad data record is not skipped, and DATALOAD processing is terminated under the following conditions:
- The first 256 records of data are bad.
- Multiple DATALOAD commands are used preceding an INFILE subcommand when an insert error occurs, and the record or a portion of the record has already been used for a successful insert by any of the DATALOAD commands.
- An error, other than an SQL INSERT error identified by SQLCODE -405, -424, -530, -802, or -803, occurs.

Tables in Nonrecoverable Storage Pools

A *nonrecoverable storage pool* is a pooled storage area for which there is no automatic recovery action to restore data to the condition it was in before a system failure or a failed operation. The message:

```
ARI8990I The table tablename is in a
         nonrecoverable storage pool.
```

is written before DATALOAD table insert processing begins if one of the tables you are loading resides in a nonrecoverable storage pool. This message indicates that changes made to this table by the DATALOAD command are **not** deleted by a ROLLBACK statement if an error occurs.

Restarting the Loading Process

The Database Services Utility is designed to run despite minor errors. If, however, an error is serious enough to halt the utility, you must rerun your particular Database Services Utility command and reprocess all your files; you cannot simply restart the Database Services Utility from the point of failure.

The COMMITCOUNT parameter, introduced in the preceding section, saves processed data at intervals that you specify. This saves you processing time because, although you must rerun jobs from the beginning, you do not have to reprocess data that has already been committed to the database. The part of the DATALOAD TABLE command that lets you bypass records is the RESTARTCOUNT parameter.

In general, you want to skip any records that have been successfully processed and also any bad input records. To run a job that has errors, proceed as follows:

1. Prepare the DB2 Server for VSE job and commands for the Database Services Utility and submit the job to run; or prepare the necessary DB2 Server for VM files and invoke the Database Services Utility.
2. When the job cancels (in VSE) or halts (in VM) with an error, determine (from the messages sent) the number of records, if any, committed and the number of records processed up to the start of the error condition.

   Note: If you were running the Database Services Utility without SET AUTOCOMMIT protection, or with too high a COMMITCOUNT value, you could rerun the job with SET AUTOCOMMIT ON and an appropriately low COMMITCOUNT value to save the successfully processed records. If you choose this course of action, return to step 1.
3. Before rerunning the job, add the following parameter to the DATALOAD TABLE command:

   ```
   RESTARTCOUNT(rcount)
   ```

   where *rcount* specifies the number of input records to be skipped. In general, start processing from the last COMMIT action.
4. Remove the error condition, if possible.
5. Rerun the job.

If a DATALOAD job is canceled or halted repeatedly by errors, or if bad records are causing the errors, consider using the following:

1. Specify as the RESTARTCOUNT value the input record count of the last ARI0800I message. A sample ARI0800I message looks like this:

   ```
   ARI0800I  ...Begin COMMIT. Input Record Count = 100
   ```
2. Specify the COMMITCOUNT value again to equal the number of records between the value found in the preceding step and the bad record.
3. Rerun DATALOAD up to the point of failure. This skips the previously committed records and commits the remainder up to the bad record.
4. Specify as a new RESTARTCOUNT value the input record count of the latest ARI0800I message **plus** *n*, where *n* is the number of bad records. (You can also specify the COMMITCOUNT parameter again to its original value.)
5. Rerun the job. DATALOAD processing begins at a point beyond the bad records and the previously committed work.

If you were running a DATALOAD job with the COMMITCOUNT set to 100, but the job was unsuccessful at record 151, you could run DATALOAD again with a new COMMITCOUNT value and restart after the number of records that were committed:

```
DATALOAD TABLE(SMITH.DEPARTMENT)
IF POS (50) = 'X'
INFILE(SOMEDEPT) COMMITCOUNT(50) RESTARTCOUNT(100)
```

Now you have committed all the records up to the bad record. To skip the bad record, change the RESTARTCOUNT value, and restore the COMMITCOUNT parameter to its original value:

```
DATALOAD TABLE(SMITH.DEPARTMENT)
IF POS (50) = 'X'
INFILE(SOMEDEPT) COMMITCOUNT(100) RESTARTCOUNT(151)
```

Alternative Method

You can specify that the Database Services Utility ignore certain error conditions and continue processing records:

1. Immediately before the DATALOAD command set in the (input) control file, include:
   ```
   SET ERRORMODE CONTINUE
   ```
2. On the same line as INFILE, add:
   ```
   COMMITCOUNT(ccount)
   ```
   where *ccount* is the number of input records to be processed before a COMMIT action is taken.
3. Run the job.

Although this procedure skips bad records, it does not pinpoint them. After a job is finished, compare the loaded table with source input documents to locate missing table rows.

For more information on Database Services Utility's error handling, see Chapter 9, "Error Handling and Debugging," on page 223.

Statistics Collection

The database manager generates table statistics while loading the data and calculates index statistics while creating an index. This method of creating statistics avoids doing a dbspace scan and a separate scan of the index pages, which are done when you issue an UPDATE STATISTICS statement.

The database manager generates table statistics while the Database Services Utility DATALOAD, RELOAD TABLE, and RELOAD DBSPACE commands are loading data only if the SET UPDATE STATISTICS command is set to ON. Other rules that must be met if statistics are to be collected for DATALOAD processing are:

- The DATALOAD command is loading data into only one table.
- No indexes exist on the table. If indexes do exist, the Database Services Utility issues an UPDATE STATISTICS after the load is complete to generate index and table statistics.
- The table being loaded currently contains no data; statistics are accumulated only for rows that are being loaded. If the table already contains data and statistics are generated when more rows are loaded, the statistics would not accurately describe the entire table. For example, to load 5000 rows into a 500000-row table, and have the table's statistics describe only the 5000 rows that were loaded, would not be accurate. For this reason, statistics are not generated when you specify the RESTARTCOUNT option. The use of RESTARTCOUNT implies that a DATALOAD with the COMMITCOUNT option had already loaded rows into the table, a failure occurred, and the DATALOAD is being restarted at the point of the most recent COMMIT. Clearly, there are already rows in the table.

  The database manager determines that rows are already in the table when the ROWCOUNT column of the SYSCATALOG table is a positive number for the table you are loading. Generally, a positive number in the ROWCOUNT column indicates that the table contains rows, but if you delete all the data from the table without updating the statistics, ROWCOUNT still contains a positive number. You must update the statistics to set the ROWCOUNT to zero before loading data into that table.

If, for any of the above reasons, table statistics were not generated while data was being loaded, the Database Services Utility executes an SQL UPDATE STATISTICS statement for each table loaded after DATALOAD or RELOAD command processing successfully ends. Statistics are neither updated automatically nor is an UPDATE STATISTICS statement executed under either one of the following conditions:

- A SET UPDATE STATISTICS OFF command was issued before the DATALOAD or RELOAD command.
- A view name was specified instead of a table name.

Chapter 3. Unloading Data with the Database Services Utility

This chapter describes the DATAUNLOAD command first, and then describes the UNLOAD commands, beginning with "UNLOAD Procedures" on page 63.

To selectively unload data from tables and views, use the DATAUNLOAD command. The DATAUNLOAD command creates a sequential file of data that you can modify and reload into a table with the DATALOAD command.

If you want to create a backup for specific dbspaces or tables, use the UNLOAD command. The UNLOAD command also allows you to move data in units of tables (one table or all the tables in a dbspace) to another database manager. If you want to reclaim fragmented disk space or reorder data records to match indexes, use the UNLOAD command followed by the RELOAD command.

Refer to the appropriate sections of the earlier chapters for details about invoking the Database Services Utility and defining files.

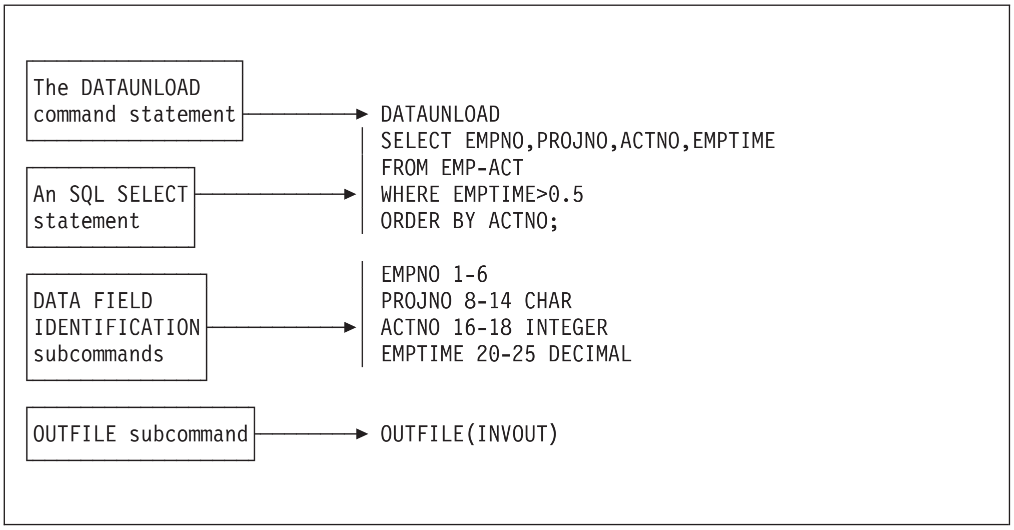
DATAUNLOAD Procedures

Database Services Utility DATAUNLOAD processing enables you to unload data from tables and views to a user-defined sequential access method (SAM) file record format. You can also unload data from remote application servers that support the DRDA flow. The data to be unloaded is selected from the database with an SQL SELECT statement that you supply. The output data file for this utility is a general-use programming interface. See "Programming Interface Information" on page 257 for a definition of general-use programming interfaces.

In general, each output record resulting from DATAUNLOAD processing contains data for a row of a table. These output data records reside in a sequential file. In a VM system, you must define this file using the CMS FILEDEF command.

The sequential output file can contain fixed, variable-length, or variable-length-spanned records. The records can be blocked or unblocked. A standard SELECT statement in its SQL syntax is used in the DATAUNLOAD command set as a mandatory subcommand. The DATAUNLOAD command and its subcommands can:
- Identify the tables to be unloaded
- Describe the data fields in the output records
- Relate table column names to output record data fields
- Identify the source of the output records.

A DATAUNLOAD command contains four elements, as shown in Figure 29 on page 52.


The DATAUNLOAD command statement
An SQL SELECT statement
DATA FIELD IDENTIFICATION subcommands
OUTFILE subcommand

```
DATAUNLOAD
SELECT EMPNO,PROJNO,ACTNO,EMPTIME
FROM EMP-ACT
WHERE EMPTIME>0.5
ORDER BY ACTNO;

EMPNO 1-6
PROJNO 8-14 CHAR
ACTNO 16-18 INTEGER
EMPTIME 20-25 DECIMAL

OUTFILE(INVOUT)
```

Figure 29. DATAUNLOAD Command Sequence

The DATAUNLOAD command statement consists of a single word and occupies the first record of the command set in the Database Services Utility (input) control file.

The SQL statement, the second element in the command set, occupies one or more control file input records. Its syntax and sequence of keywords are the same as they would be if used outside the Database Services Utility.

Entering Commands from a Workstation

Most Database Services Utility commands and all SQL statements must end with a semicolon (;) when the control file is assigned to TERMINAL. In general, use a semicolon to terminate all commands entered through your workstation.

Data Field Identification (DFI) subcommands, element three of DATAUNLOAD commands, identify the location in the output records for the data of columns specified in the select-list parameter. DFI subcommands are optional; if they are omitted, the resulting output data fields are sequenced according to system defaults. See “Unloading Data in System-Defined Format” for more information. If DFI subcommands **are** included, each must occupy a single record of the DB2 Server for VSE input control card file or DB2 Server for VM command-file input record.

The OUTFILE subcommand, the final element in the DATAUNLOAD command set, identifies the sequential output file that is to contain the data unloaded by the preceding DATAUNLOAD command sequence. It tells the Database Services Utility to start unloading data to the file identified by the corresponding *ddname* parameter. In a VM system, this parameter is defined in the FILEDEF command.

Unloading Data in System-Defined Format

The DATAUNLOAD command and subcommands are contained on more than one input record. If you want to unload all the data from a table in a

system-determined sequence and with the default output data field format, the three parts of the Database Services Utility command are:
- The DATAUNLOAD command
- An SQL SELECT statement ended with a semicolon
- An OUTFILE subcommand.

Default Output Data Field Formats

If you do not supply DFI subcommands in a DATAUNLOAD command set and if the source table column contains double-byte character set (DBCS) data, the default data type for the output data fields is CHARACTER or GRAPHIC data type. The overall format of the output data depends on the data type and length (actual or maximum) of the column from which the data is taken. Figure 30 and Figure 31 show the default output data field sequence.

If you do not supply DFI subcommands, the data fields appear in the output records in the order of occurrence of columns in the SELECT statement's select-list parameter. Each field is separated from the next by a blank position (hex 40).

For fixed-length output records, the data field associated with the first select-list column starts in position 1 of the record, as shown in Figure 30.

```
┌─────────────┬───────┬─────────────┬───────┬───────┬─────────────┐
│ data from   │ blank │ data from   │ blank │  ...  │ data from   │
│ select-list │       │ select-list │       │  ...  │ select-list │
│ column 1    │       │ column 2    │       │  ...  │ column n    │
└─────────────┴───────┴─────────────┴───────┴───────┴─────────────┘
  .
  .
  .
  Position 1
```

Figure 30. Default Data Field Sequence—Fixed-Length Output Records

For variable-length output records, the data field associated with the select-list column starts in position 5 of the record because the first 4 bytes are the record length control field. Figure 31 on page 54 shows the data fields for variable-length output records.

```
 ___________________________________________________________________________________
| Record    | data from    | blank | data from    | blank    | ... | blank | data from    |
| Length    | select-list  |       | select-list  |          | ... |       | select-list  |
| Control   | column 1     |       | column 2     |          | ... |       | column n     |
| Field     |              |       |              |          |     |       |              |
|___________|______________|_______|______________|__________|_____|_______|______________|
  .           .
  .           .
  .           .
  .           Position 5
  .
  Position 1
```

Figure 31. Default Data Field Sequence—Variable-Length Output Records

In **DB2 Server for VSE**, proceed as follows to unload data in the default output-field format.

1. Provide the following Database Services Utility command:

   ```
   DATAUNLOAD
   ```

2. Put an SQL SELECT statement on the next record. (SQL statement syntax is beyond the scope of this manual. See the *DB2 Server for VSE & VM SQL Reference* for information about SQL statement syntax.) Figure 33 shows a sample DATAUNLOAD command that uses system formatting defaults.

```
DATAUNLOAD
SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME
FROM EMP_ACT,EMPLOYEE
WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
ORDER BY EMP_ACT.EMPNO;
OUTFILE(OUTPUT1)
```

Figure 32. DATAUNLOAD Command without DFI Subcommands

3. To cause default formatting of data fields, do not supply DFI subcommands.
4. On the next record, put:

   `OUTFILE (`*ddname*`)`

   where *ddname* identifies the output file. Use the same *ddname* in a TLBL or DLBL statement, depending on whether you want to store the data on tape or in a DASD file.
5. Submit the job for processing.

In **DB2 Server for VM**, proceed as follows to unload data in the default output-field format.

1. Issue the SQLINIT command to initialize the user machine. If you have already done this, proceed to Step 2.
2. Create a control file to contain the command you construct in the following steps. See “Working with a Control File in DB2 Server for VM” on page 13 for detailed information on creating a control file.
3. Enter the command name. Type:

   ```
   DATAUNLOAD
   ```

4. On a new line, enter the SQL SELECT statement. (SQL statement syntax is beyond the scope of this manual. See the *DB2 Server for VSE & VM SQL Reference* for information about SQL statement syntax.) Figure 33 shows a sample DATAUNLOAD command that uses system formatting defaults.

```
DATAUNLOAD
SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME
FROM EMP_ACT,EMPLOYEE
WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
ORDER BY EMP_ACT.EMPNO;
OUTFILE(OUTPUT1)
```

Figure 33. DATAUNLOAD Command without DFI Subcommands

5. To cause default formatting of data fields, do not supply DFI subcommands.
6. Enter the OUTFILE subcommand. On a new line, type:

   `OUTFILE(`*`ddname`*`)`

   where *ddname* identifies the output data file. You need to use the same *ddname* when you specify the FILEDEF statement for the output file.
7. Store the control file.
8. In CMS, specify the necessary FILEDEF statements. For general information about FILEDEF statements, see "Using File Definitions" on page 14. For command-specific information, see "Using File Definitions with the DB2 Server for VM DATAUNLOAD Command" on page 62.
9. Issue the SQLDBSU command to run the Database Services Utility. If you did not specify FILEDEFs for the control and message files, use the default values in the SQLDBSU EXEC. For more information on the SQLDBSU EXEC, see "Using the SQLDBSU EXEC" on page 15.

As the DATAUNLOAD command is executed, the actual command sequence, the default data-field sequence, and messages are written to the report or message file. Figure 34 on page 56 and Figure 35 on page 56 show output that results from running the DATAUNLOAD command shown in Figure 33.

```
ARI0801I DBS Utility started: 07/24/89 10:26:44.
         AUTOCOMMIT = OFF ERRORMODE = OFF
         ISOLATION LEVEL = REPEATABLE READ
-------> CONNECT "SQLDBA " IDENTIFIED BY ********;
ARI8004I User SQLDBA connected to database SQLDBA.
ARI0500I SQL processing was successful.
ARI0505I SQLCODE = 0  SQLSTATE = 00000  ROWCOUNT = 0
------->
-------> DATAUNLOAD
-------> SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME
-------> FROM EMP_ACT,EMPLOYEE
-------> WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
-------> ORDER BY EMP_ACT.EMPNO;
-------> OUTFILE(OUTPUT1)
ARI0852I DATAUNLOAD processing started.
ARI0868I DNAME=OUTPUT1 RECFM=F RECSZ=80 BLKSIZE=80  <— See Note
ARI0836I Default output record data field positions:
ARI0837I EMPNO 1-6
ARI0837I PROJNO 8-13
ARI0837I EMPTIME 15-21
ARI0835I 74 record(s) written to the output data file.
ARI0855I DATAUNLOAD processing successful.
ARI0802I End of command file input.
ARI8997I ...Begin COMMIT processing.
ARI0811I ...COMMIT of any database changes successful.
ARI0809I ...No error(s) occurred during command processing.
ARI0808I DBS processing completed: 07/24/89 10:26:45.
```

Figure 34. Database Services Utility DB2 Server for VSE Report Output: Default Data Fields

```
1ARI0801I DBS Utility started: 07/24/89 10:26:44.
         AUTOCOMMIT = OFF ERRORMODE = OFF
         ISOLATION LEVEL = REPEATABLE READ
0------> DATAUNLOAD
-------> SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME
-------> FROM EMP_ACT,EMPLOYEE
-------> WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
-------> ORDER BY EMP_ACT.EMPNO;
-------> OUTFILE(OUTPUT1)
ARI0852I DATAUNLOAD processing started.
ARI0868I DNAME=OUTPUT1 RECFM=F RECSZ=80 BLKSIZE=80  <— See Note
ARI0836I Default output record data field positions:
ARI0837I EMPNO 1-6
ARI0837I PROJNO 8-13
ARI0837I EMPTIME 15-21
ARI0835I 74 record(s) written to the output data file.
ARI0855I DATAUNLOAD processing successful.
ARI0802I End of command file input.
ARI8997I ...Begin COMMIT processing.
ARI0811I ...COMMIT of any database changes successful.
ARI0809I ...No error(s) occurred during command processing.
ARI0808I DBS processing completed: 07/24/89 10:26:45.
```

Figure 35. Database Services Utility DB2 Server for VM Message File Output: Default Data Fields

Note: The RECFM, RECSZ, and BLKSIZE information displayed in the message ARI0868I depends on either the JCL data definition statement or the CMS FILEDEF command specifications for the output file with *ddname*=OUTPUT1.

The records in the output data file are formatted according to system-determined (default) criteria. Table 1 shows the data field format resulting from the DATAUNLOAD command specification given in Figure 33 on page 55.

Table 1. Output Record Format That is System Determined

Record Position	Data Value Source (Column or Other)	Output Record Field Data Type
1-6	EMPNO	CHAR
7	(blank)	CHAR
8-13	PROJNO	CHAR
14	(blank)	CHAR
15-21	EMPTIME	CHAR

Unloading Data in User-Specified Format

If you want to unload all the data from a table in a specific sequence and with user-specified output data record field formats, the four parts of the Database Services Utility command are:

- The DATAUNLOAD command
- An SQL SELECT statement ended with a semicolon
- DFI subcommands for each column to be unloaded
- The OUTFILE subcommand.

The data for a DFI-referenced column is in the same positions in all the output data records.

Provide DFIs for All Table Columns—or None

A DFI subcommand identifies the location in the output record where you want to place the unloaded data. For example, suppose TABLE1 has five columns and you enter the following SELECT statement:

```
SELECT * FROM TABLE1
```

or

```
SELECT colname1,colname2,colname3,colname4,colname5 FROM TABLE1
```

If you supply only three DFI subcommands, the Database Services Utility only unloads the three table columns identified in the subcommands. If you want to unload data for all five columns and you want to specify your own output record format for any of the columns, you must supply five DFI subcommands.

The difference between unloading data in system-defined and user-defined format is the presence of DFI subcommands. A DFI subcommand identifies the location in the output records where the data for a column specified in the select-list parameter should be placed. The DFI also identifies the data type of its data field in the output record.

Whereas the DATAUNLOAD default output field sequence uses the order of presentation in the select-list parameter, you choose the order of fields in the output records when you supply DFI subcommands. In choosing positions for the output data-record fields, you should leave a blank position between each field for clarity, but this is not mandatory. Do not, however, use character positions 1 through 4 for output data if you have specified variable-length output records for the output file. Record positions 1–4 are reserved for the record length control field.

Figure 36 shows a DATAUNLOAD command sequence using DFI subcommands. The EMPNO field is to occupy positions 1 through 6 in the output records. The PROJNO field goes in positions 8 through 13. The EMPTIME field is to take positions 15 through 21 as data type DECIMAL.

```
DATAUNLOAD
SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME,JOB
FROM EMP_ACT,EMPLOYEE
WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
ORDER BY EMP_ACT.EMPNO;
EMPNO         1-6
PROJNO        8-13
EMPTIME       15-21    DECIMAL
OUTFILE(OUTPUT1)
```

Figure 36. DATAUNLOAD Command with DFI Subcommands

To unload data in user-specified output-file format, supply DFI subcommands. Use the standard method for unloading data in "Unloading Data in System-Defined Format" on page 63, but supply a DFI subcommand for each column that you want to unload. To construct a DFI subcommand, use the following format:

`column-reference startpos-endpos data-type set-null-clause`

- *column-reference* is usually the name of the table column in the select-list parameter, but it might be an integer. For more information, see "Data_Field_Id Subcommand" on page 170.
- *startpos-endpos* gives the first and last positions of the named table-column data in the output record. You can omit *endpos* if the output data is one character long.
- *data-type* is the data format to be used in the data field of the output record. For more information, refer to the sections starting on page 171 in Chapter 8, "Command Reference," on page 135.
- *set-null-clause* is a conditional expression that tells the Database Services Utility to provide a particular flag, control character, or string in the output record if the value of the output data is null. The *set-null-clause* also specifies the start and end positions in the output record for the flag or string.

Unloading NULL Values

You can use the set-null-clause to instruct the utility to insert a particular value whenever a null value occurs in a table that you are unloading. In the following example, you instruct the Database Services Utility to write a question mark in position 22 of the output record whenever a null field occurs in the table that is being unloaded.

```
DATAUNLOAD
SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME,JOB
FROM EMP_ACT,EMPLOYEE
WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
ORDER BY EMP_ACT.EMPNO;
EMPNO         1-6
PROJNO        8-13
EMPTIME       15-21    DECIMAL    IF NULL SET POS(22) = '?'
OUTFILE(OUTPUT1)
```

Figure 37. Unloading NULL Values with DATAUNLOAD Command

As is shown in Figure 37 on page 59, if the value of an EMPTIME column is null, the utility puts a *?* value in output record position 22. For more information about using the set-null-clause, see page 175 under "Data_Field_Id Subcommand" on page 170 in Chapter 8, "Command Reference," on page 135.

Periodic Reports during the Processing of Long Jobs

During DATAUNLOAD processing of a file containing more than 15,000 data records, the message ARI8995I is written every 15,000 records to inform you that the job is running normally and that *n* records have been unloaded. In a VM system, these messages are written to your workstation, and in a VSE system, they are written to the system operator's console. If the number of records being read from the database is less than 15,000, you do not receive message ARI8995I.

As the DATAUNLOAD command is executed, the actual command sequence and messages are written to the report or message file. Figure 38 on page 60 and Figure 39 on page 60 show report results from running the DATAUNLOAD command as shown in Figure 36 on page 58.

```
1ARI0801I DBS Utility started: 10/05/89 14:54:41.
          AUTOCOMMIT = OFF ERRORMODE = OFF
          ISOLATION LEVEL = REPEATABLE READ
1ARI0803I ...Extended DBCS (DBCS=YES) processing now in effect.
0-------> DATAUNLOAD
 -------> SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME,JOB
 -------> FROM EMP_ACT,EMPLOYEE
 -------> WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
 -------> ORDER BY EMP_ACT.EMPNO;
 -------> EMPNO        1-6
 -------> PROJNO       8-13
 -------> EMPTIME      15-21      DECIMAL    IF NULL SET POS(22) = '?'
 -------> OUTFILE(OUTPUT1)
 ARI0852I DATAUNLOAD processing started.
 ARI0831I Column JOB data will not be unloaded.
 ARI0868I DNAME=OUTPUT1 RECFM=F RECSZ=80 BLKSIZE=80  <— See Note
 ARI0835I 74 record(s) written to the output data file.
 ARI0855I DATAUNLOAD processing successful.
 ARI0802I End of command file input.
 ARI8997I ...Begin COMMIT processing.
 ARI0811I ...COMMIT of any database changes successful.
 ARI0809I ...No error(s) occurred during command processing.
 ARI0808I DBS processing completed: 10/05/89 10:54:44.
```

Figure 38. Database Services Utility Report Output with User-Specified Data Fields

```
ARI0801I DBS Utility started: 10/05/89 14:54:41.
         AUTOCOMMIT = OFF ERRORMODE = OFF
         ISOLATION LEVEL = REPEATABLE READ
-------> CONNECT "SQLDBA " IDENTIFIED BY ********;
ARI8004I User SQLDBA connected to database SQLDBA.
ARI0500I SQL processing was successful.
ARI0505I SQLCODE = 0  SQLSTATE = 00000  ROWCOUNT = 0
------->
ARI8003I ...Extended DBCS (DBCS=YES) processing now in effect.
-------> DATAUNLOAD
-------> SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME,JOB
-------> FROM EMP_ACT,EMPLOYEE
-------> WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
-------> ORDER BY EMP_ACT.EMPNO;
-------> EMPNO        1-6
-------> PROJNO       8-13
-------> EMPTIME      15-21      DECIMAL    IF NULL SET POS(22)  = '?'
-------> OUTFILE(OUTPUT1)
ARI0831I Column JOB data will not be unloaded.
ARI0868I DNAME=OUTPUT1 RECFM=F RECSZ=80 BLKSIZE=80
ARI0835I 74 record(s) written to the output data file.
ARI0855I DATAUNLOAD processing successful.
ARI0802I End of command file input.
ARI8997I ...Begin COMMIT processing.
ARI0811I ...COMMIT of any database changes successful.
ARI0809I ...No error(s) occurred during command processing.
ARI0808I DBS processing completed: 10/05/89 14:54:44.
```

Figure 39. Database Services Utility Message File Output with User-Specified Data Fields

Note: The RECFM, RECSZ, and BLKSIZE information displayed in the message ARI0868I depends on the CMS FILEDEF command specifications for the output file OUTPUT1.

The Database Services Utility does not unload data for which no DFI subcommand exists (unless **all** DFIs are omitted), but the report or message file does show you the columns that are not unloaded.

The records in the output data file are formatted according to your specifications. Table 2 shows the data field format resulting from the DATAUNLOAD command specification shown in Figure 36 on page 58.

Table 2. Output Record Format That is User Determined

Record Position	Data Value Source (Column or Other)	Output Record Field Data Type
1-6	EMPNO	CHAR
7	(blank)	CHAR
8-13	PROJNO	CHAR
14	(blank)	CHAR
15-21	EMPTIME	DECIMAL
22	EMPTIME (null indicator)	CHAR

Unloading a View

A *view* is a virtual table that is derived from one or more tables, from other views, or from combinations of views and tables. When views are processed and displayed or printed, they are indistinguishable from tables; they have rows and columns and, like tables, views have no inherent order of rows.

You can use the DATAUNLOAD command to unload views as if they were tables. Once unloaded, the output data of a view is the same as the data from a table.

To use the DATAUNLOAD command to unload a view, use the same procedure that you use to unload a table, but specify a view name instead of a table name in the SQL SELECT statement. You can select all the columns of the view (SELECT *), or use the select-list parameter. If you use this parameter, specify the names of view columns.

Suppose you create a view such as this:

```
CREATE VIEW TOSPIFFY (NUMBER,NAME,MANAGER)
AS SELECT DEPTNO,DEPTNAME,MGRNO
FROM DEPARTMENT
WHERE ADMRDEPT = 'A00'
```

To unload the view, construct a DATAUNLOAD command like this:

```
DATAUNLOAD
SELECT * FROM TOSPIFFY;
NUMBER    1-3
NAME      6-42
MANAGER   45-50 IF NULL SET POS (45-50) = '      '
OUTFILE(SUBSPIF)
```

The DATAUNLOAD command uses the view column names, not the column names of the founding table, and null entries in the MANAGER column are represented by 6 blanks in the output data file.

Using File Definitions with the DB2 Server for VM DATAUNLOAD Command

The DATAUNLOAD command uses three files: the control file, the message file, and the data output file. You must define all three files, either with the SQLDBSU EXEC or a FILEDEF command. Figure 40 on page 62 shows the relationship of the three files and the appropriate definition facility (FILEDEF or SQLDBSU) for each.

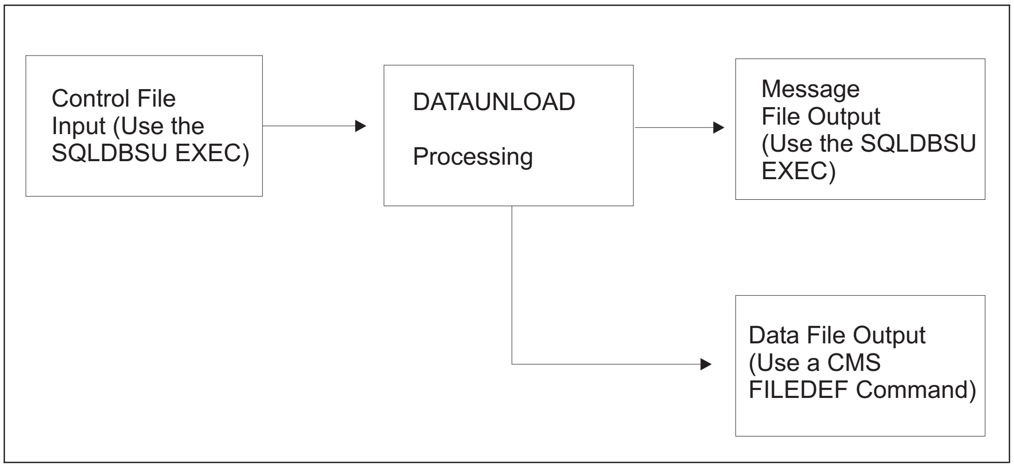

Control File Input (Use the SQLDBSU EXEC)
DATAUNLOAD Processing
Message File Output (Use the SQLDBSU EXEC)
Data File Output (Use a CMS FILEDEF Command)

Figure 40. DATAUNLOAD Files

For more information on FILEDEF parameters and options, see Appendix B, “FILEDEF Command Syntax and Notes,” on page 249.

FILEDEFs Supporting DATAUNLOAD Command Processing

In the FILEDEF command defining the Database Services Utility DATAUNLOAD command output data file, all record format (RECFM) values are supported except for undefined (U) or carriage controls (A or M). If you define CMS output files with variable-length spanned records (RECFM=VS or VBS), you must use the file-mode number **4**. In this case, DATAUNLOAD processing changes the record format to U. See Appendix B, “FILEDEF Command Syntax and Notes,” on page 249 for more information about undefined (U) record format usage.

Note: If the message ARI0868I, generated using DATAUNLOAD command processing, identified RECFM=U for an output file defined with RECFM=VBS or VS, the CMS FILEDEF command that defines the DATALOAD input data files must still specify RECFM=VBS or VS accordingly. CMS FILEDEF command information for DATALOAD command processing must be identical to the information in the FILEDEF command that was used when DATAUNLOAD command processing created the file.

A sample CMS FILEDEF command defining a CMS file for DATAUNLOAD command processing is:

```
FILEDEF DBSFILE DISK DBSFILE DATA A (RECFM FB LRECL 800 BLOCK 1600
```

where DBSFILE is the *ddname* used in the DATAUNLOAD command, and it refers to the output file DBSFILE DATA A.

If you want to print some of the data in a table, use the FILEDEF statement to specify the printer as the output device:

```
FILEDEF PRINTOUT PRINTER
```

where PRINTOUT is the *ddname* that you use in the DATAUNLOAD command.

UNLOAD Procedures

This section describes the UNLOAD commands provided by the Database Services Utility.

Unloading Data in System-Defined Format

Database Services Utility UNLOAD TABLE and UNLOAD DBSPACE processing allows you to unload tables and views to a sequential file. You can later use this file as input to Database Services Utility RELOAD TABLE and RELOAD DBSPACE processing (described in Chapter 4, “Reloading Data with the Database Services Utility,” on page 71). With the UNLOAD commands, you cannot unload data in a user-defined format.

Note: You cannot use the UNLOAD DBSPACE and UNLOAD TABLE commands if you are using DRDA flow.

The following is a brief description of the two UNLOAD commands:

- UNLOAD DBSPACE unloads all tables in a particular dbspace to an output file.
- UNLOAD TABLE is more specific than UNLOAD DBSPACE; it unloads a specific table into an output file.

Unloading a Dbspace or Table That You Do Not Own

To unload dbspaces and tables that you do not own, concatenate the owner’s user ID to the dbspace name (for example, SMITH.PERSONNEL). You must have the SELECT privilege on all tables in the dbspace you want to unload.

Figure 41 on page 64 shows how the parts of an UNLOAD DBSPACE command relate to the dbspace and output file. In the figure, the command:

```
UNLOAD DBSPACE (SMITH.PERSONNEL) OUTFILE (SAVE)
```

unloads the dbspace called SMITH.PERSONNEL to the file SAVE.

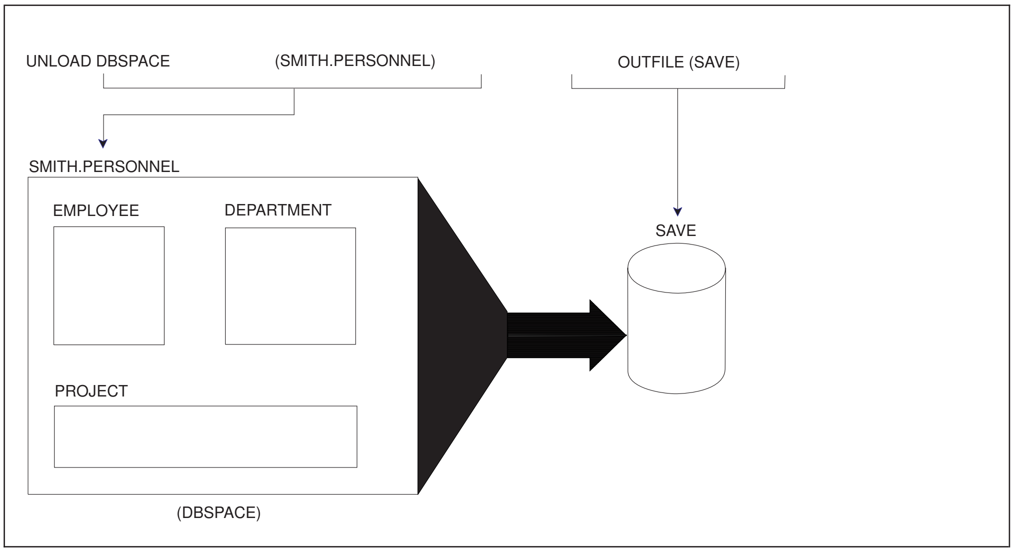


Figure 41. Diagram of the UNLOAD DBSPACE Command

The format of the UNLOAD TABLE command is the same as UNLOAD DBSPACE, but instead of a dbspace, specify a table name. In Figure 42, the UNLOAD TABLE command unloads a single table called SMITH.DEPARTMENT to the output file SAVE.

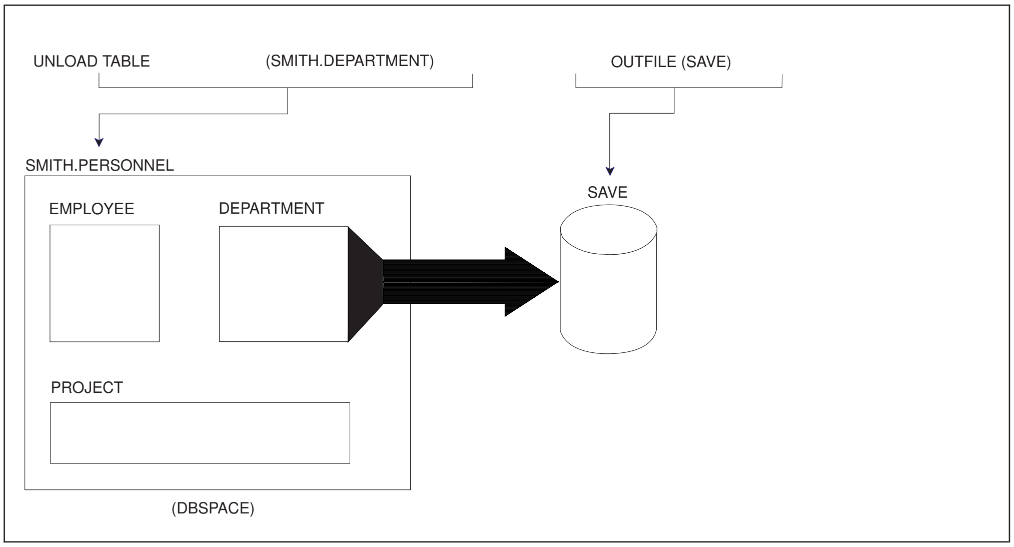


Figure 42. Diagram of the UNLOAD TABLE Command

The Database Services Utility UNLOAD processing writes all rows from a table or all rows from all tables in a dbspace as individual records to the sequential output file. Before producing these records, the utility writes records that contain information supporting the RELOAD function. The Database Services Utility RELOAD processing with the NEW parameter uses this information for creating the table(s).

Attention:

In RELOAD processing with the parameter NEW, the Database Services Utility must create the table. The table is not created if the CREATE TABLE statement used by the Database Services Utility is greater than 8192 bytes. If the statement is greater than 8192 bytes, you can use the RELOAD command only with the PURGE parameter and only if the table already exists.

The CREATE TABLE statement used by the Database Services Utility will be longer than the statement that you initially issue if:

- You do not specify the column clauses in the column definition of the table.
- You use the ALTER TABLE statement to add columns to the table.

UNLOAD Processing Uses Indexes

The Database Services Utility unloads table data in the sequence identified by the first index created for the table. This first index is also known as the *clustering index*. The CLUSTER column of the SYSINDEXES catalog table indicates the index for a given table that is the first, or clustering, index. (The CLUSTER column contains an F or W value. See *DB2 Server for VSE & VM Database Administration* for more information on SYSINDEX and clustering indexes.) The data is ordered by the clustering index before it is selected for unloading.

If a table has no indexes, the data is not put in order before it is unloaded; table rows are unloaded in a system-determined order.

The UNLOAD command does not unload any indexes, primary or foreign keys, or constraint definitions. Additionally, a *package* (preprocessed program) that depends on the unique constraint indexes is invalidated when unique constraints are dropped. All packages dependent on the table are invalidated when a unique constraint is added to the table because they might have UPDATE statements that cause multiple-row updates. Packages are also invalidated when unique constraints are activated or deactivated. When using the Database Services Utility RELOAD command with PURGE OPTION, however, you should ACTIVATE the unique constraints (in the ALTER TABLE statement) rather than re-create them because it is more efficient.

Furthermore, the Database Services Utility is sensitive to the tagging of character and graphic data (single byte, double byte, and mixed), which identify the format of the data, such as US or Kanji. The tags are maintained when you use the UNLOAD and RELOAD commands.

Attention:

Operate under isolation level repeatable read (the default Database Services Utility processing mode) when you use the UNLOAD command to ensure a consistent state of the database during backup or migration.

Using the UNLOAD DBSPACE Command

The command statements UNLOAD DBSPACE and UNLOAD TABLE are much simpler than the DATAUNLOAD statement because the UNLOAD commands do not specify the output fields. The lack of specifications leads to very long output records.

Such long output records make it difficult to locate specific data or to edit the output file. The UNLOAD commands are designed to provide output files for backup or subsequent reloading at remote sites (or locally, for purposes of reorganizing DB2 Server for VSE & VM data structures). Consequently, unloaded data should be used only for reloading purposes. The following UNLOAD DBSPACE command unloads all tables in a dbspace named PERSONNEL. The tables are placed in an output file called SAVE:

```
UNLOAD DBSPACE (PERSONNEL) OUTFILE (SAVE)
```

The syntax is simple, requiring only:
- The command name
- The name of the dbspace to be unloaded
- The name of the output file. (In VM this is specified in the FILEDEF command.)

To unload an entire dbspace for backup or subsequent reloading, follow the procedure in "Unloading Data in System-Defined Format" on page 52, but use the UNLOAD DBSPACE command instead of the DATAUNLOAD command. The UNLOAD DBSPACE command uses the following structure:

```
UNLOAD DBSPACE (dbspace-name)
```

where *dbspace-name* is the name of the dbspace.

Using the UNLOAD TABLE Command

You can use UNLOAD TABLE to specify the table you want to unload. UNLOAD DBSPACE unloads all the tables in a dbspace.

For example, if you make regular backups of the inventory table of a small company, you unload the table frequently. The table is in a dbspace with other tables that you do not need to backup as often. Use the UNLOAD TABLE command to backup only the inventory table. If your system fails, you can reload the table by using the RELOAD TABLE command, which is discussed in the next chapter.

The syntax of the UNLOAD TABLE command is similar to that of the UNLOAD DBSPACE command, requiring only:
- The command name
- The name of the table to be unloaded
- The name of the output file. (Specified in the FILEDEF command in VM.)

To unload a table for backup or subsequent reloading, use the same procedure given in "Unloading Data in System-Defined Format" on page 52, but use the UNLOAD TABLE command instead of the DATAUNLOAD command. The UNLOAD TABLE command has the following structure:

```
UNLOAD TABLE (table-name)
```

where *table-name* is the name of the table.

Unloading Views

You can unload a view (virtual table) with UNLOAD TABLE processing. The Database Services Utility processes views in the same manner as a table without indexes.

In DB2 Server for VSE

Each file of a multiple-file tape volume must be identified with the correct file name and file sequence number of the TLBL statement for each tape file. Tape rewind processing is controlled by the VSE job control statements.

In DB2 Server for VM

If you want to do a simple Database Services Utility command, such as unload one table to a DASD file, and you are not going to repeat this command regularly, use the utility interactively to avoid creating a control file and specifying a message file.

If you have to execute several UNLOAD TABLE commands or use the same command again, use message and control files. For example, to unload the organization tables that are in the same dbspace as personnel tables, you need several UNLOAD TABLE commands to specify each organization table. To unload project activity tables for managers who have different dbspaces, you again need several UNLOAD TABLE commands. Putting the Database Services Utility commands in a control file enables you to use the commands again. Also, if any command fails, you can refer to the message file repeatedly to deal with each error message and correct the mistake in the control file without retyping all of the UNLOAD commands.

Using File Definitions with the DB2 Server for VM UNLOAD DBSPACE and UNLOAD TABLE Commands

The UNLOAD DBSPACE and UNLOAD TABLE commands use at least three files: the control file, the message file, and one or more data output files.

Multiple Output-File Possibilities

You can have more than one UNLOAD TABLE or UNLOAD DBSPACE command within a single invocation of the utility; each command must unload data to a separate file, or the data is lost.

You can unload data to a multiple-volume tape file. You can also unload data to multiple files on a single tape volume in one execution of the Database Services Utility. Each UNLOAD operation must unload to a separate file. The Database Services Utility does not rewind the tape when each file is opened for output.

Figure 43 on page 68 shows the relationship between the UNLOAD files and the appropriate definition facility (FILEDEF or SQLDBSU) for each.

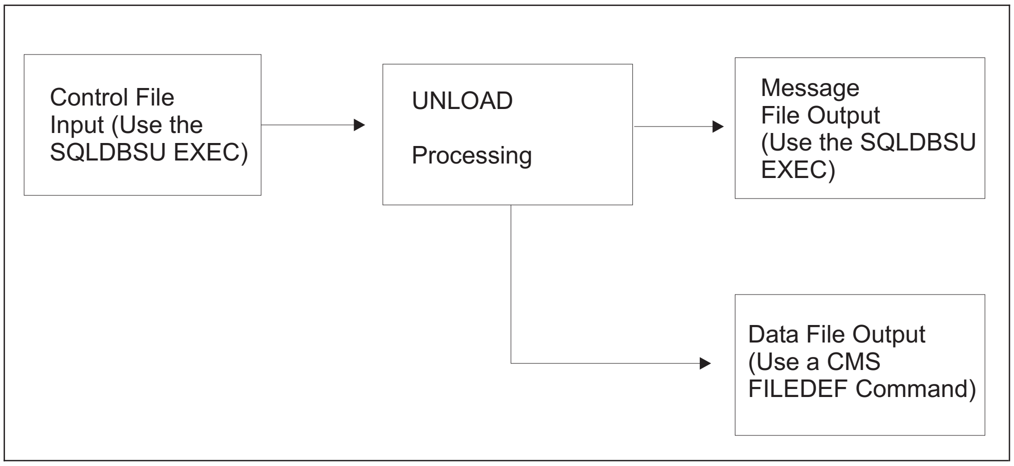

Control File Input (Use the SQLDBSU EXEC)
UNLOAD Processing
Message File Output (Use the SQLDBSU EXEC)
Data File Output (Use a CMS FILEDEF Command)

Figure 43. UNLOAD Files

Each file of a multiple-file tape volume must be identified with the correct *ddname* and label specification in the CMS FILEDEF command issued for it. Tape rewind processing is controlled by FILEDEF command specifications and performed by CMS OS/QSAM.

Specify VBS Record Format in the UNLOAD FILEDEFs

Always specify a record format (RECFM) of VBS for UNLOAD processing. UNLOAD processing changes the record format to U if the system required logical record length is greater than the specified block size (BLOCK) value minus 4. Otherwise, UNLOAD processing changes the record format to VB. See Appendix B, “FILEDEF Command Syntax and Notes,” on page 249 for more information about undefined (U) record format usage.

A block size (BLOCK) greater than 8 244 is recommended for tape output files created by UNLOAD processing.

An example of a FILEDEF statement defining a tape output file is:

```
FILEDEF SAVE2 TAP1 (RECFM VBS BLOCK 8244
```

where SAVE2 is the *ddname* that you used in the UNLOAD command.

FILEDEFs Supporting UNLOAD Command Processing

The FILEDEF command defining the UNLOAD output data file can identify a CMS file with **4** appended to the file mode letter (for example, A4) or a sequential tape file supported by CMS OS/OSAM. Always specify a record format of VBS or a block size value (or both) in the FILEDEF command defining the data file. UNLOAD processing changes the record format to U if the system-required logical record length is greater than the specified block size (BLOCK) value minus 4. Otherwise, UNLOAD processing changes the record format to VB. See Appendix B, “FILEDEF Command Syntax and Notes,” on page 249 for more information about undefined (U) record format usage.

Note: The message ARI0868I identifies the file characteristics used in the Database Services Utility’s processing. If message ARI0868I, generated during UNLOAD command processing, identified RECFM=U or VB for the UNLOAD output file, the CMS FILEDEF command that defines the

RELOAD input data file must still specify RECFM=VBS. CMS FILEDEF command information for RELOAD command processing must be identical to the information in the FILEDEF command used when UNLOAD command processing created the file. If message ARI0868I indicates RECFM=U for a tape output file, you can obtain significant performance improvements by increasing the block size value specified in the FILEDEF command that defines the *ddname*.

An example of a FILEDEF command defining a CMS file for UNLOAD processing is:

```
FILEDEF DBSFILE DISK DBSFILE DATA A4 (RECFM VBS BLOCK 2048
```

where DBSFILE is the *ddname* used in your UNLOAD command, and DBSFILE refers to the output file DBSFILE DATA A4.

For more information on FILEDEF parameters and options, see Appendix B, "FILEDEF Command Syntax and Notes," on page 249.

Release Coexistence Considerations for DB2 Server for VM

Changes were required in Version 7 Release 1 to handle file I/O correctly when using CMS 15 and later. These changes affect the format of data that is unloaded and reloaded by the UNLOAD and RELOAD commands of the DBS Utility. If you use the DBS Utility's UNLOAD and RELOAD commands with databases at different release levels, you must ensure that the code changes have been applied at all release levels. For releases prior to Version 7 Release 1, you must apply the following APARs:

Release	**APAR**
3.5	PQ28584
5.1	PQ28583
6.1	PQ27957

Chapter 4. Reloading Data with the Database Services Utility

The Database Services Utility RELOAD commands work in conjunction with the UNLOAD commands. You can reload a dbspace or a table, and you can reload the data into tables that already exist or create new tables as you reload. This chapter explains how to use RELOAD DBSPACE and RELOAD TABLE and describes the table information that is preserved after the data is reloaded.

Refer to the appropriate sections of the earlier chapters for details about invoking the Database Services Utility and defining files.

RELOAD Procedures

Reloading Data in System-Defined Format

The RELOAD commands, like their UNLOAD counterparts, do not support user-defined data formats. System-defined format is the only option. You must use the OUTFILE output of the UNLOAD DBSPACE or UNLOAD TABLE commands as input to the RELOAD DBSPACE and RELOAD TABLE commands.

Note: For a VM application requestor, the RELOAD DBSPACE and RELOAD TABLE commands can only be used if a DB2 Server for VM application server is used and the protocol is either SQLDS or AUTO. For a VSE application requestor, the RELOAD DBSPACE and RELOAD TABLE commands can only be used if a DB2 Server is running on the same VSE system as the Database Services Utility or if the DB2 Server for VM is on VM and is accessed through Guest Sharing.

Assuming that you have output from UNLOAD processing, and you want to use it as input to a DB2 Server for VSE & VM database, you must now decide whether to use the RELOAD DBSPACE or the RELOAD TABLE command.

Note: Use the UNLOAD command, not the DATAUNLOAD command. The unit of output of the DATAUNLOAD command is the *table row* whereas the UNLOAD commands have the *table* as their unit of output.

RELOAD DBSPACE is usually associated with UNLOAD DBSPACE, and RELOAD TABLE is associated with UNLOAD TABLE. In practice, this is the most frequent pairing, but all four commands use identical data formatting. Sometimes changing the object when you go from UNLOAD to RELOAD is appropriate. Briefly, the objects that each command manipulates are:
- UNLOAD DBSPACE unloads an entire dbspace.
- UNLOAD TABLE unloads just one table.
- RELOAD DBSPACE reloads an entire dbspace.
- RELOAD TABLE reloads just one table.

You might want to use RELOAD DBSPACE with UNLOAD TABLE if you unload a single table and want to reload it into a dbspace. If you do not want to specify **where** the table goes in the dbspace, the RELOAD DBSPACE command achieves the same result as the RELOAD TABLE command. RELOAD DBSPACE is even more convenient because it has fewer parameters to specify.

You might want to use RELOAD TABLE with UNLOAD DBSPACE if you unload an entire dbspace and want to reload just one of its tables into another dbspace. Using the RELOAD DBSPACE command reloads the entire UNLOAD output file, not just the desired table.

Processing Multiple Tables or Multiple Files

The Database Services Utility's RELOAD processing does not support the **concurrent** loading of multiple tables. Sequential loading, however, is supported, as long as the tables are in the same file.

During one invocation of the Database Services Utility, you can reload data from a multiple-volume tape file or from a multiple-file tape volume.

In DB2 Server for VSE, to reload data from a multiple-file tape volume, you must specify the correct file name and file sequence number on the TLBL statement for each tape file. Because the Database Services Utility rewinds the tape when each file is opened for input, this information is necessary to locate the correct file on the tape.

RELOAD DBSPACE processing loads tables serially in the order that they appear in the input file. Use the RELOAD DBSPACE command to supply four pieces of information to the Database Services Utility:
- The name of the command
- The identity of the dbspace to be loaded
- The replacement method (NEW or PURGE) to use
- The identity of the data input file.

Figure 44 shows the command flow.

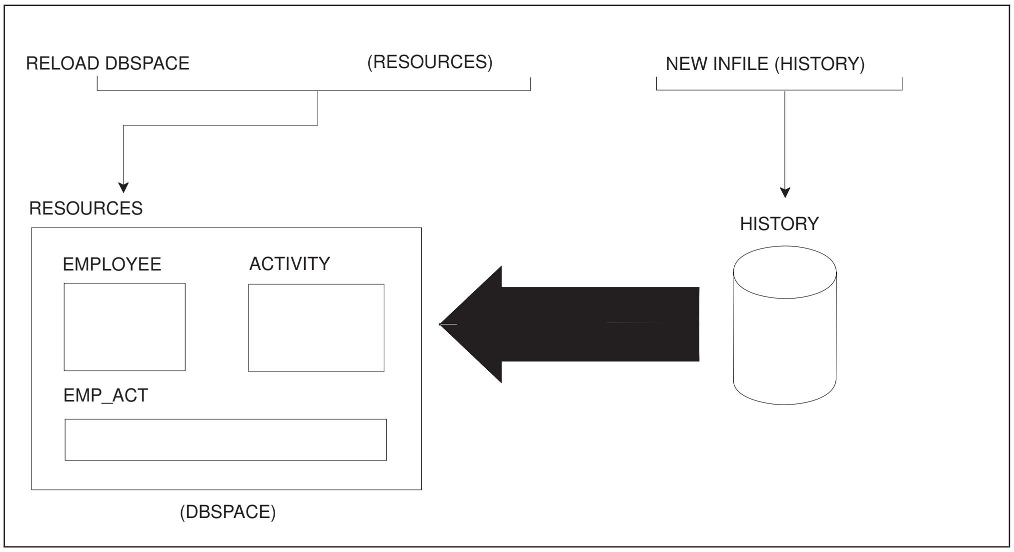


Figure 44. Diagram of the RELOAD DBSPACE Command

This figure shows a RELOAD DBSPACE operation on the RESOURCES dbspace, and the creation of new tables from the input file called HISTORY.

The RELOAD TABLE command is more precise than the RELOAD DBSPACE command. RELOAD TABLE specifies that you want to reload only one table no

matter how many exist in the input file. In the RELOAD TABLE command, you give the following five pieces of information to the Database Services Utility:
- The name of the command
- The identity of the target table to be loaded
- The replacement method (NEW or PURGE) to use
  - If NEW, the identity of the dbspace in which the table is to be created.
- Optionally, the identity of the source table being loaded
- The identity of the data input file.

Figure 45 shows the command flow.

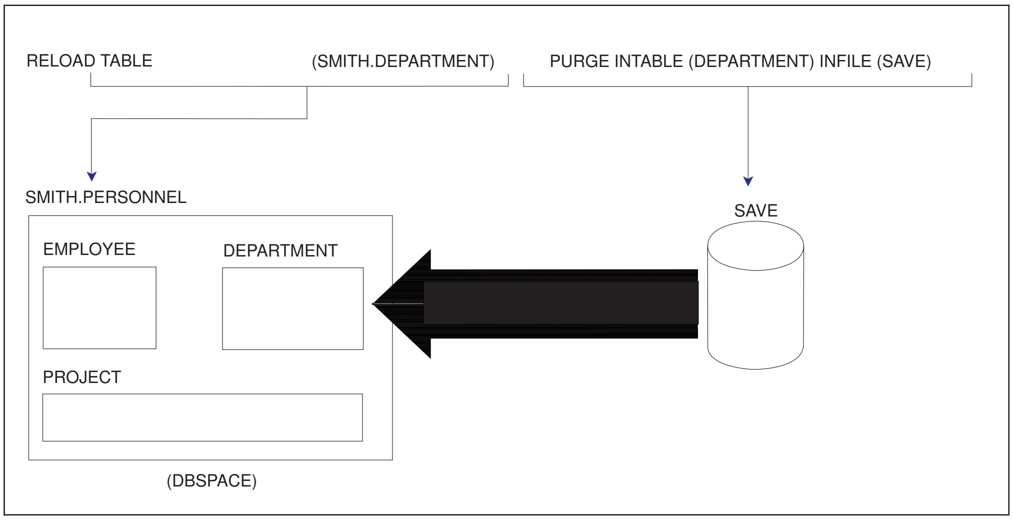


Figure 45. Diagram of the RELOAD TABLE Command

This figure shows a RELOAD TABLE operation on Smith's DEPARTMENT table, which must be purged first. The SAVE file is used as input.

If the UPDATE STATISTICS setting is ON, the RELOAD command automatically causes table statistics to be generated while the data is being reloaded.

Indexing Notes

With the PURGE Parameter:

When a table is purged, the default clustering rules are used because all indexes for the purged table are dropped.

With the NEW Parameter:

When a new table is created, the column definitions are identical to the definitions of the table contained in the input file, except the keys and unique constraints are not reproduced. The new table also does not have any indexes defined for it. You must construct them yourself by issuing subsequent SQL CREATE INDEX statements.

Using the NEW Parameter with Field Procedures:

When a new table is created, the field procedures are not reproduced. Thus, using the reload ’NEW’ parameter is not recommended for tables with field procedures. To reload tables with field procedures, use the ’PURGE’ parameter.

When reloading a dbspace or a table, you **must** either create new tables for the RELOAD input or purge old tables before reloading them. A particular table in the input file replaces the like-named table in the target dbspace if the PURGE parameter is specified, but it remains unloaded if the NEW parameter is in effect. Similarly, under the NEW option, a particular table in the input file remains unloaded when a like-named table with that name exists in **any** dbspace of the entire database; if no table with that name exists, however, the Database Services Utility creates a table into which the given input is loaded.

Referential Integrity and the RELOAD Commands

Referential integrity might affect the RELOAD commands. Specifically, if you create an unloaded file when there is no primary key, and there is a primary key on the target table at the time of RELOAD PURGE, the primary key becomes active after the RELOAD PURGE operation. If you do not want the primary key active, you must drop the primary key manually by using the ALTER TABLE statement.

If the table being reloaded has an active primary key, the Database Services Utility records this fact and issues an ALTER TABLE *table-name* DEACTIVATE PRIMARY KEY command. The Database Services Utility also saves the active foreign key names, unique constraints, and their owner’s name, before issuing an ALTER TABLE *table-name* DEACTIVATE FOREIGN KEY or DEACTIVATE UNIQUE KEY command. After the data has been loaded, the Database Services Utility reactivates the keys.

For more information about referential integrity, see the *DB2 Server for VSE & VM Database Administration* manual.

The Database Services Utility also preserves the tags for character and graphic data (single byte, double byte, and mixed) that identify the data format, such as US or Kanji. The tags are reloaded with the data when you use the RELOAD command.

A user is allowed to use a DBSU module from one release to connect to a database server containing a DBSU package at a different release. Specifically, in an UNLOAD TABLE/DBSPACE and RELOAD TABLE/DBSPACE scenario, there are 4 objects being used that may all be at different releases. For Data Capture, it is only necessary to consider whether the R750 release of the object is being used or a pre-R510 release is being used. In the chart below, ″Unload Module″ refers to the release of the DBSU module which the user is accessing when performing the unload operation. This may not be the same as the release of the database server which the user is connecting to. Similarly, ″Reload Module″ refers to the release of the DBSU module which the user is accessing when performing the reload operation. This may not be the same as the release of the database server which the user is connecting to. ″Unload P/S″ refers to the release of the database server and the release of the DBSU package contained in the database server in which the table is being unloaded. Similarly, ″Reload P/S″ refers to the release of the database server and the release of the DBSU package contained in the database server in which the table is being reloaded. The release of the DBSU package must be equal to the release of the database server where it is contained. Mixed releases are not supported.

In the most general case, a user can use a DBSU module at release A to unload a table from a database server which is at release B. Then, the user can use a DBSU module at release C to reload the table to another database server at release D. The chart below can be used to determine whether the DATA CAPTURE setting for the table will be restored.

Table 3. DATA CAPTURE settings and DBSU RELOAD and UNLOAD

Unload P/S	Unload Module	Reload P/S	Reload Module	Comments
pre-R510	n/a	n/a	n/a	Tables in a pre-R510 server do not contain a Data Capture setting.
R730	pre-R510	n/a	n/a	The pre-R510 unload module does not save the Data Capture setting so the setting will not be restored on the reload.
R730	R730	pre-R510	n/a	Data Capture setting will be saved in the unload file but pre-R510 servers do not allow a Data Capture setting for tables so the setting will not be restored on the reload.
R730	R730	R730	pre-R510	Data Capture setting will be saved in the unload file but the pre-R510 reload module does not restore the Data Capture setting so the setting will not be restored on the reload.
R730	R730	R730	R730	Data Capture setting will be saved in the unload file and will be restored by the reload module.

Using the PURGE Parameter

The PURGE keyword tells the Database Services Utility that the target table exists, and that all rows must be deleted from it before RELOAD TABLE processing begins. (If the target table does not exist, you receive an error message.) Of course, the column definitions of the target table must be identical to those of the source table.

The Database Services Utility, as part of PURGE processing, drops the clustering index, deactivates any active primary keys, active foreign keys, and active unique keys, and deletes all indexes on the target table before deleting and reloading the data. Therefore, you must have DBA authority to do a RELOAD with the PURGE option if the target table or any of its indexes are not yours. After all tables have been reloaded, the Database Services Utility reactivates the clustering index, primary key and unique keys, and re-creates the remaining indexes. It ensures that the first index that was created for the table (as recorded at PURGE time) is also the first index re-created. After all the tables are processed, the Database Services Utility reactivates all the foreign keys that it deactivated. DB2 Server for VSE & VM packages are invalidated because of table index deletions, but are automatically preprocessed the next time someone attempts to execute the package.

The following example illustrates the PURGE parameter:

```
RELOAD DBSPACE (RESOURCES) PURGE INFILE(HISTORY)
```

PURGE tells the Database Services Utility to delete all the rows of the table before loading the data. The table must, however, exist in the specified dbspace. Note also that fully qualified table names are always used internally for RELOAD DBSPACE. That is, if you unload JONES.EMP_ACT and use RELOAD DBSPACE with a PURGE option, JONES.EMP_ACT is the only table affected by the reload.

Using the NEW Parameter

The specified dbspace must already exist before you can reload tables into it. The NEW parameter causes the utility to create tables, not dbspaces. If you are using UNLOAD and RELOAD processing to duplicate an existing dbspace (as for testing application programs), first acquire an appropriate dbspace. The SQL ACQUIRE DBSPACE statement is described in the *DB2 Server for VSE & VM SQL Reference*. If the table you are reloading does not replace a table already in the dbspace, the Database Services Utility can create the target table for you. In the following example, the source table EMPTABLE is not in the target dbspace:

```
RELOAD TABLE(EMPTABLE) NEW(PRODUCTION)
       INTABLE(EMPLOYEE)
       INFILE(SAVE)
```

The NEW parameter in the above command tells the Database Services Utility that the table (EMPTABLE) to be loaded does not exist and must be created. It also identifies the dbspace (PRODUCTION) where you want the table created. The Database Services Utility creates the EMPTABLE, finds the EMPLOYEE table on the input file (SAVE), and loads the data. The new table is created in a private dbspace, PRODUCTION, that the current user owns. If the current user does not own a private dbspace with the specified name, the table is created in a public dbspace with this name. If you want to have the new table created in a particular dbspace, specify:

```
NEW (dbspace-name)
```

where *dbspace-name* is the name of the dbspace.

In another example, suppose that user ID BOB is the current Database Services Utility user. BOB issues this command:

```
RELOAD DBSPACE (RESOURCES) NEW INFILE(HISTORY)
```

Suppose, also, that one of the tables in the HISTORY file is called BOB.EMPLOYEE. If BOB already owns a table called BOB.EMPLOYEE in any other dbspace, the table cannot be created and loaded in the RESOURCES dbspace. The user ID concatenated to the table name uniquely identifies a table within the database. Thus, if BOB.EMPLOYEE already exists, it is impossible for the utility to create another BOB.EMPLOYEE anywhere else in the database.

Using the RELOAD DBSPACE Command

Percent Free Space

During RELOAD processing, the current percent free value for the dbspace being loaded, or for the dbspace where the table being loaded resides, can be critical. Before RELOAD processing begins, increase the percent free space value to reserve free space for additional rows inserted after the RELOAD process is completed. Immediately after RELOAD processing is completed, reduce the percent free value to allow the reserved free space to be used for the new rows. Refer to the *DB2 Server for VSE & VM Database Administration* for more information on the dbspace percent free specification.

Reloading Several Tables into a Dbspace Where They Are Already Defined

To reload multiple tables into a dbspace where they already exist, proceed as follows:

In VSE

1. Provide the following Database Services Utility command:

   ```
   RELOAD DBSPACE (dbspace-name)
   ```

   where *dbspace-name* is the name of the dbspace.
2. On the same record as the RELOAD DBSPACE command, leave one space and put the replacement method for existing tables:

   ```
   PURGE
   ```
3. Also on the same record, leave one space and put:

   ```
   INFILE(ddname)
   ```

   where *ddname* identifies the input file. Use the same *ddname* in a TLBL or DLBL statement, depending on whether the data is stored on tape or in a DASD file.
4. Submit the job to run.

In VM

1. Issue the SQLINIT command to initialize the user machine. If you have already done this, proceed to Step 2.
2. Create a control file to contain the command you construct in the following steps. See “Working with a Control File in DB2 Server for VM” on page 13 for detailed information on creating a control file.
3. Enter the command and dbspace name. Type:

   ```
   RELOAD DBSPACE (dbspace-name)
   ```

   where *dbspace-name* is the name of the dbspace.

4. On the same line as the RELOAD DBSPACE command, enter the replacement method for existing tables. Leave one space; then type:

   ```
   PURGE
   ```

5. On the same line, leave one space; then type:

   ```
   INFILE(ddname)
   ```

   where *ddname* identifies the input data file. You need to use the same *ddname* when you specify the FILEDEF statement for the input file.
6. Store the control file.
7. In CMS, specify the necessary FILEDEF statements. For general information about FILEDEF statements, see "Using File Definitions" on page 14. For command specific information, see "Using File Definitions with DB2 Server for VM RELOAD DBSPACE and RELOAD TABLE Commands" on page 81.
8. Specify an SQLDBSU EXEC statement; if you did not specify FILEDEFs for the control and message files, use the default values in the SQLDBSU EXEC. For more information on the SQLDBSU EXEC, see "Using the SQLDBSU EXEC" on page 15.

Reloading Several Tables into a Dbspace Where They Do Not Exist

To reload multiple tables into a database where they do not exist, follow the procedure given in "Reloading Several Tables into a Dbspace Where They Are Already Defined" on page 77, but substitute NEW for PURGE. NEW indicates the replacement method for new tables.

Loading a Single Table with the RELOAD DBSPACE Command

If you have just one table to load into a dbspace (that was unloaded with an UNLOAD TABLE command), use the RELOAD DBSPACE command. Follow the procedure in "Reloading Several Tables into a Dbspace Where They Are Already Defined" on page 77, and use the appropriate replacement method (PURGE or NEW) for the table to be loaded.

Note: If your input file contains multiple tables but you do not want to reload all of them, use the RELOAD TABLE command.

Using the RELOAD TABLE Command

The reason for using the RELOAD TABLE command rather than the RELOAD DBSPACE command is to reload one particular table into a dbspace. The RELOAD DBSPACE command loads an entire input file of table data into a dbspace (subject to the constraints imposed by the NEW or PURGE parameters). Although RELOAD processing follows the input order of the data, UNLOAD output is unpredictable: you have no way of knowing the sequence of tables in the UNLOAD DBSPACE output file. In general, if you use output from an UNLOAD DBSPACE as input to RELOAD TABLE processing (meaning that you want to reload a specific table), use the INTABLE parameter with the RELOAD TABLE command.

Reloading a Single Table into a Dbspace Where It Is Already Defined

To reload a single table into a dbspace where it already exists, proceed as follows:

In VSE

1. Provide the following Database Services Utility command:

   ```
   RELOAD TABLE (table-name)
   ```

where *table-name* is the name of the table.

2. On the same record as the RELOAD TABLE command, leave one space and put:

   ```
   PURGE
   ```
3. Also on the same record, leave one space and put:

   ```
   INFILE (ddname)
   ```

   where *ddname* identifies the input file. Use the same *ddname* in a TLBL or DLBL statement, depending on whether the data is stored on tape or in a DASD file.
4. Submit the job to run.

In VM

1. Issue the SQLINIT command to initialize the user machine to the application server where the data is to be reloaded. If you have already done this, proceed to Step 2.
2. Create a control file to contain the command you construct in the following steps. See "Working with a Control File in DB2 Server for VM" on page 13 for detailed information on creating a control file.
3. Enter the command and table name. Type:

   ```
   RELOAD TABLE (table-name)
   ```

   where *table-name* is the name of the table.
4. On the same line as the RELOAD TABLE command, enter the replacement method for existing tables. Leave one space; then type:

   ```
   PURGE
   ```
5. On the same line, leave one space; then type:

   ```
   INFILE(ddname)
   ```

   where *ddname* identifies the input data file. You need to use the same *ddname* when you specify the FILEDEF statement for the input file.
6. Store the control file.
7. In CMS, specify the necessary FILEDEF statements. When you specify the FILEDEF statement for the input data file, use the same *ddname* you assigned to the INFILE in this procedure. For general information about FILEDEF statements, see "Using File Definitions" on page 14. For command specific information, see "Using File Definitions with DB2 Server for VM RELOAD DBSPACE and RELOAD TABLE Commands" on page 81.
8. Specify an SQLDBSU EXEC statement; if you did not specify FILEDEFs for the control and message files, use the default values in the SQLDBSU EXEC. For more information on the SQLDBSU EXEC, see "Using the SQLDBSU EXEC" on page 15.

Nonrecoverable Storage Pools

Before RELOAD PURGE table insert processing begins, the message:

```
ARI8990I The table table-name is in a
         nonrecoverable storage pool.
```

is written if one of the tables you are reloading is found in a nonrecoverable storage pool. This message indicates that changes made to this table by the RELOAD command are not deleted by a ROLLBACK statement if an error occurs.

Reloading Views

You can also reload views if the view meets the restrictions defined under the RELOAD TABLE command description (see "RELOAD TABLE" on page 196). Use the PURGE parameter to reload a view that was previously unloaded. Using PURGE makes use of an existing view definition and does not violate the rule that a view is a virtual table. The only difference between reloading a table and reloading a view is that statistics are not collected for a view.

Reloading a Single Table into a Dbspace Where It Does Not Exist

To reload a single table into a dbspace where it is not defined, follow the procedure in "Reloading a Single Table into a Dbspace Where It Is Already Defined" on page 78, but use the following replacement method instead of PURGE:

```
NEW (dbspace)
```

where *dbspace* is the name of the dbspace where you want to create a new table.

Reloading a Specific Table from a Multitable Input Source

The multitable input source referred to in this section is the output file from an UNLOAD DBSPACE command. If you do regular backups of a dbspace, and the data in one table is lost or modified incorrectly, reload the one table with the RELOAD TABLE command. Use the procedure in "Reloading a Single Table into a Dbspace Where It Is Already Defined" on page 78, but with the following differences:

- Use the appropriate replacement method, NEW or PURGE.
- After providing the replacement method, leave one space, and then put:

  ```
  INTABLE(table)
  ```

  where *table* is the name of the source table.
- Leave one space and put:

  ```
  INFILE (ddname)
  ```

  where *ddname* identifies the input data file.

If you are reloading DB2 Server for VSE data from magnetic tape, identify each file of a multiple-file tape volume with the correct file name and file sequence number on the TLBL statement for each tape file. Because the Database Services Utility rewinds the tape when each file is opened for input, this information is necessary to locate the correct file on the tape.

Notification of Records Reloaded

If you are reloading more than 15,000 data records, messages (ARI8995I) are written to your terminal after every 15,000 records to inform you that a multiple of 15,000 records has been loaded.

Suppose that a dbspace was unloaded and the dbspace contained two tables named EMPLOYEE. One of these EMPLOYEE tables was originally created by SCOTT, the other by MIKE. If you want to reload the EMPLOYEE table that was created by SCOTT, you should identify the table by prefixing the table name EMPLOYEE with the owner SCOTT in the INTABLE parameter:

```
RELOAD TABLE(EMPTABLE) NEW(PRODUCTION)
       INTABLE(SCOTT.EMPLOYEE)
       INFILE(SAVE)
```

If you do not, the Database Services Utility reloads the data of the first table it finds in the input file that has the same name. If you omit the INTABLE parameter completely, the utility uses the data of the first table it finds in the input file, regardless of the table name and owner.

Using File Definitions with DB2 Server for VM RELOAD DBSPACE and RELOAD TABLE Commands

RELOAD processing requires a control file and a data file for input, and a message file for output.

Use the Same File Definition for RELOAD As for UNLOAD

CMS FILEDEF command information for RELOAD command processing should be identical to the information in the FILEDEF command you used when UNLOAD command processing created the file.

Figure 46 shows the relationship of the RELOAD files and the appropriate definition facility (FILEDEF or SQLDBSU) for each.

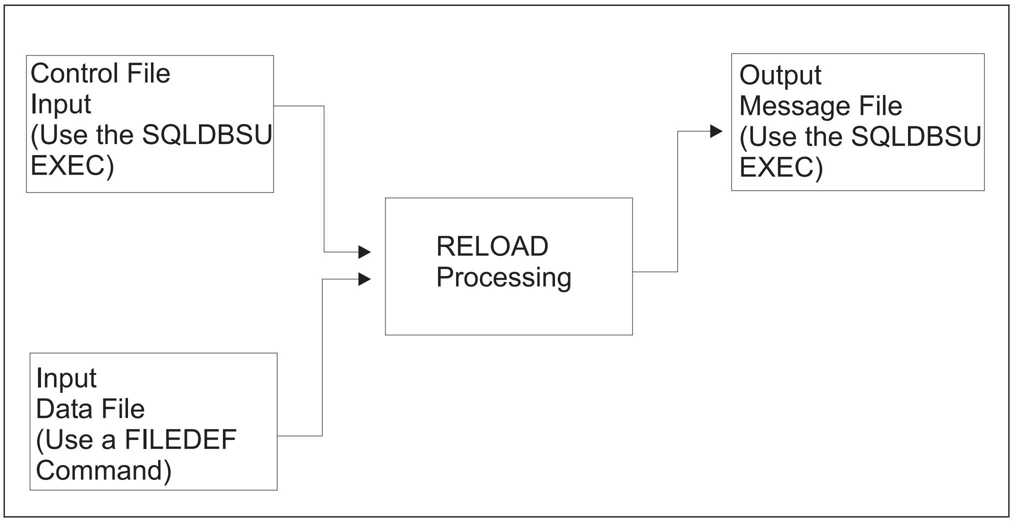

Control File Input (Use the SQLDBSU EXEC)
Input Data File (Use a FILEDEF Command)
RELOAD Processing
Output Message File (Use the SQLDBSU EXEC)

Figure 46. RELOAD DBSPACE and RELOAD TABLE Files

The default record format for RELOAD processing is variable-length blocked spanned (VBS). If you specify a RECFM value other than VBS or a LRECL value with the CMS FILEDEF command, it is ignored and overridden. RELOAD processing changes the record format from VBS to VB. The message ARI0868I identifies the file characteristics used by Database Services Utility processing.

Isolation Level for RELOAD Operations

Set the isolation level to repeatable read when you reload data to ensure a consistent state of the database during backup or migration.

Identify each file of a multiple-file tape volume with the *ddname* and label specifications in the CMS FILEDEF command that you issue for each tape file. The Database Services Utility does not perform any tape rewind processing. Tape rewind processing is controlled by FILEDEF command specifications and performed by CMS OS/QSAM.

Use the UNLOAD and RELOAD commands (RELOAD with the PURGE option) to reorder the data records to match the indexes. Use the FILEDEF to specify a DISK file for quick and easy unloading and reloading. This reordering improves the efficiency of queries performed on your tables.

FILEDEFs Supporting RELOAD Command Processing

The FILEDEF command defining the Database Services Utility RELOAD output data file identifies a CMS file with **4** appended to the file mode letter (for example, A4) or a sequential tape file supported by CMS OS/QSAM. Always specify a record format of VBS or a block size value (or both) in the FILEDEF command defining the data file. RELOAD processing changes the record format from VBS to VB.

A sample of a FILEDEF command defining a CMS file for RELOAD processing is:

```
FILEDEF DBSFILE DISK DBSFILE DATA A4 (RECFM VBS BLOCK 2048
```

where DBSFILE is the *ddname* used in your RELOAD command and DBSFILE DATA A4 is the name of the input file.

For more information on FILEDEF parameters and options, see Appendix B, “FILEDEF Command Syntax and Notes,” on page 249.

Release Coexistence Considerations for DB2 Server for VM

Changes were required in Version 7 Release 1 to handle file I/O correctly when using CMS 15 and later. These changes affect the format of data that is unloaded and reloaded by the UNLOAD and RELOAD commands of the DBS Utility. If you use the DBS Utility’s UNLOAD and RELOAD commands with databases at different release levels, you must ensure that the code changes have been applied at all release levels. For releases prior to Version 7 Release 1, you must apply the following APARs:

Release	APAR
3.5	PQ28584
5.1	PQ28583
6.1	PQ27957

Statistics Collection

If SET UPDATE STATISTICS is on, table statistics are automatically generated while the data is reloading. This method of creating statistics avoids a dbspace scan and a separate scan of the index pages, which occurs when an UPDATE STATISTICS statement is issued. If SET UPDATE STATISTICS is off, the statistics are not updated.

Note: Consider using SET UPDATE STATISTICS ON for all RELOAD processing to update the table statistics while the data is reloading.

Chapter 5. Unloading and Reloading Packages with the Database Services Utility

This chapter explains how to use the UNLOAD and RELOAD PACKAGE commands to distribute packages to connected application servers that use the Database Services Utility. The two commands work together to transport a package from one application server to another. When the package is reloaded, you have a choice of purging the old package or creating a new one in the database. Finally, the owner of the package has to authorize the people who will use the package.

Refer to the appropriate sections of the earlier chapters for details about invoking the Database Services Utility and defining files.

Package Procedures

This section describes SQL preprocessing the PACKAGE commands.

Preprocessing

SQL statements in an application program are preprocessed (that is, analyzed and converted) by the system before the program is compiled (or assembled).

DB2 Server for VSE & VM preprocessors do the following:
- Generate a modified version of the source code
- Convert SQL statements into a package and save the package in the application server
- Verify that the current user has authority to access the data and, if so, grant the user the privilege to use the package generated
- Update the database catalogs.

The preprocessor action is shown graphically below.

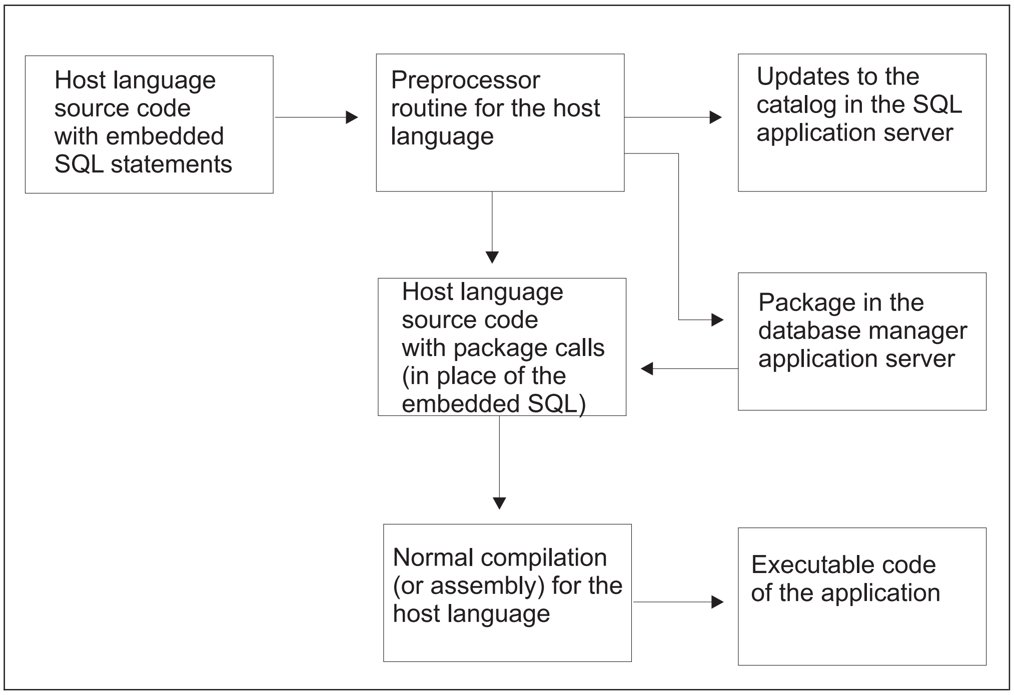

Host language source code with embedded SQL statements
Preprocessor routine for the host language
Updates to the catalog in the SQL application server
Host language source code with package calls (in place of the embedded SQL)
Package in the database manager application server
Normal compilation (or assembly) for the host language
Executable code of the application

Figure 47. Preprocessing

A package contains code for the SQL statements used in the program. The access path is based on available data statistics and applicable table indexes.

A package is available when its program needs it; moreover, because it is stored in a database, a package is monitored by database manager security mechanisms and change-management facilities.

You must preprocess an application program that switches between application servers on every application server that it accesses. To avoid distributing the program source code and preprocessing it on multiple systems, distribute packages to the connected (local or remote) application servers using the Database Services Utility.

To prevent you from unintentionally running an updated program against an old package, when you preprocess the package, a consistency token is generated and stored in both the program and the package. If the SQL request is to succeed when you run the program, the consistency token, which is based on a timestamp, must match the one in the package.

Each time that you preprocess a program, a consistency token is generated. You can choose to generate a blank consistency token. If you are running the program against multiple application servers, the package for that program, which is stored in all the application servers, must have the same consistency token as the program. If the consistency tokens do not match, the program cannot be run on the application server, or an error may occur. To ensure that the consistency tokens match, preprocess the program once in a DB2 Server for VSE & VM environment and distribute the package to other application servers using the UNLOAD PACKAGE command and the RELOAD PACKAGE command.

The PROGRAM command is a synonym for PACKAGE. The RELOAD or UNLOAD PROGRAM, and RELOAD or UNLOAD PACKAGE are therefore equivalent commands.

The UNLOAD PACKAGE and RELOAD PACKAGE commands are complementary: UNLOAD PACKAGE copies a package to a sequential file and RELOAD PACKAGE reads the package back into an application server.

To ensure that only authorized users manipulate packages in the database, only owners of programs and database administrators are entitled to unload or reload packages.

Keep Interconnected Databases at the Same Level

If you move a package between application servers at different release levels, and a facility of the database manager used by the reloaded package is not available on the new application server, an error occurs. The error occurs when the unloaded package is dynamically preprocessed again during the RELOAD.

When RELOAD processing is completed, the system updates the TIMESTAMP column of the SYSACCESS catalog table to the date and time of the RELOAD.

Using the UNLOAD PACKAGE Command

The UNLOAD PACKAGE command generates output records that contain:
- Preprocessing information
- Each SQL statement used in the associated program
- Information about its corresponding host variables.

In using the UNLOAD PACKAGE command, you must be either the owner of the program whose package you are unloading or a database administrator. Supply the following information to the Database Services Utility:
- Name of the command
- Identity of the package to be unloaded
- Optionally, the name of the application server containing the package
- Identity of the output file.

Note: The UNLOAD PACKAGE command is not supported if you are using DRDA flow.

Unloading a Package

Your system must have database switching capability to access other application servers.

To unload a package for backup or to transfer to another application server, proceed as follows:

In VSE

1. Provide the following Database Services Utility command:
   ```
   UNLOAD PACKAGE (owner.package-name)
   ```
   where *owner* is the name of the owner of the associated package, and *package-name* is the name of the package. If you omit *owner*, the database manager uses your user ID but still checks to ensure that you have the RUN privilege for the named package.
2. If the package resides in an application server other than the one you are accessing, leave one space and type:
   ```
   FROM (server-name)
   ```

where *server-name* is the name of the other application server.

3. On the same record, leave one space and type:

   ```
   OUTFILE (ddname)
   ```

   where *ddname* identifies the sequential output file on tape. Use the same *ddname* in a TLBL statement. If the output file is on DASD, specify PDEV(DASD) after the *ddname* and use the same *ddname* in a DLBL statement. Do not use SYSPCH as the *ddname*, because the output file content may be invalid and may cause the RELOAD PACKAGE command to fail.
4. Submit the job for processing.

An example of the UNLOAD PACKAGE command is:

```
UNLOAD PACKAGE(MARCY.PROG3) OUTFILE(PROGOUT3) FROM(server-name)
```

where PROG3 is the name of the package, MARCY is the owner, PROGOUT3 is the output data file, and *server-name* is the name of the other application server.

In VM

1. Issue the SQLINIT command to initialize the user machine to the application server where the package to be unloaded resides. If you have already done this, proceed to Step 2.
2. Create a control file to contain the command you construct in the following steps. See "Working with a Control File in DB2 Server for VM" on page 13 for detailed information on creating a control file.
3. Enter the command and the package name. Type:

   ```
   UNLOAD PACKAGE (owner.package-name)
   ```

   where *owner* is the name of the owner of the associated package, and *package-name* is the name of the package. If you omit *owner*, the system uses your user ID but still checks to ensure that you have the RUN privilege for the named package.
4. If the package resides in an application server other than the one you are logged on to, leave one space and type:

   ```
   FROM(server-name)
   ```

   where *server-name* is the name of the other application server.

   Note: The use of FROM always ignores any preceding CONNECT operations and uses the VM user ID as a default. In some situations, the user ID received at the target application server is different from your VM user ID. For example, an entry in the COMDIR might change the user ID, or the target system might change it. If you find this procedure unacceptable, issue the explicit CONNECTs as required, and use the UNLOAD command without the FROM parameter.
5. On the same line, leave one space and type:

   ```
   OUTFILE(ddname)
   ```

   where *ddname* identifies the output data file.
6. Store the control file.
7. In CMS, specify the necessary FILEDEF statements. When you specify the FILEDEF statement for the output data file, use the same *ddname* you assigned to the OUTFILE in this procedure. For general information about FILEDEF statements, see "Using File Definitions" on page 14. For command specific information, see "Using File Definitions with DB2 Server for VM UNLOAD and RELOAD PACKAGE Commands" on page 92.

8. Specify an SQLDBSU EXEC statement; if you did not specify FILEDEFs for the control and message files, use the default values in the SQLDBSU EXEC. For more information on the SQLDBSU EXEC, see "Using the SQLDBSU EXEC" on page 15.

An example of the UNLOAD PACKAGE command is:

```
UNLOAD PACKAGE(MARCY.PROG3) FROM(PAYROLL) OUTFILE(PROGOUT3)
```

where PROG3 is the name of the package, MARCY is the owner, the package is in the PAYROLL database, and the output data file is PROGOUT3.

Using the RELOAD PACKAGE Command

For a VM application requestor, the RELOAD PACKAGE can be used against a DB2 Server for VM application server or any non-DB2 Server for VM application server that uses DRDA flow. For a VSE application requestor, the RELOAD PACKAGE can be used against a DB2 Server for VSE running on the same VSE system as the Database Services Utility, a VM Database via Guest Sharing, or any non-DB2 Server for VSE application server that uses DRDA flow.

The following tables show the different package migration scenarios:

Table 4. Different Reload Package Scenarios in DB2 Server for VM

Unloaded From Application Server Release	Unloaded Using DBSU Release	Reloading Using DBSU Release	Reloading To		Result
			Application Server Release	Using Protocol	
2.2	2.2, 3.1 or later	2.2, 3.1 or later	2.2, 3.1 or later	SQLDS	No Error
3.1 or later	2.2, 3.1 or later	2.2, 3.1 or later	2.2	SQLDS	Error
3.1 or later	2.2	2.2	3.1 or later	SQLDS	No Error
3.1 or later	2.2	3.1 or later	3.1 or later	SQLDS	Error
3.1 or later	3.1 or later	2.2,3.1 or later	3.1 or later	SQLDS	No Error
3.1 or later	3.1 or later	3.1 or later	3.1 or later	SQLDS	No Error
2.2	2.2,3.1 or later	3.3 or later	3.3 or later	DRDA	Error
2.2	2.2,3.1 or later	3.3 or later	Non DB2 for VM	DRDA	Error
3.1 or later	2.2	3.3 or later	3.3 or later	DRDA	Error
3.1 or later	2.2	3.3 or later	Non DB2 for VM	DRDA	Error
3.1 or later	3.1 or later	3.3 or later	3.3 or later	DRDA	No Error
3.1 or later	3.1 or later	3.3 or later	Non DB2 for VM	DRDA	No Error
2.2, 3.1 or later	2.2, 3.1 or later	3.3 or later	2.2, 3.1 or 3.2	DRDA	Error

Table 5. Different Reload Package Scenarios in DB2 Server for VSE

Unloaded From Application Server Release	Unloaded Using DBSU Release	Reloading Using DBSU Release	Reloading To		Result
			Application Server Release	Using Protocol	
2.2	2.2, 3.1 or later	2.2, 3.1 or later	2.2, 3.1 or later	SQLDS	No Error
3.1 or later	2.2, 3.1 or later	2.2, 3.1 or later	2.2	SQLDS	Error
3.1 or later	2.2	2.2	3.1 or later	SQLDS	No Error
3.1 or later	2.2	3.1 or later	3.1 or later	SQLDS	Error
3.1 or later	3.1 or later	2.2,3.1 or later	3.1 or later	SQLDS	No Error

Table 5. Different Reload Package Scenarios in DB2 Server for VSE (continued)

Unloaded From Application Server Release	Unloaded Using DBSU Release	Reloading Using DBSU Release	Reloading To		Result
			Application Server Release	Using Protocol	
3.1 or later	3.1 or later	3.1 or later	3.1 or later	SQLDS	No Error
2.2	2.2,3.1 or later	7.1	7.1	DRDA	Error
2.2	2.2,3.1 or later	7.1	Non DB2 for VSE	DRDA	Error
3.1 or later	2.2	7.1	7.1	DRDA	Error
3.1 or later	2.2	7.1	Non DB2 for VSE	DRDA	Error
3.1 or later	3.1 or later	7.1	7.1	DRDA	No Error
3.1 or later	3.1 or later	7.1	Non DB2 for VSE	DRDA	No Error
2.2, 3.1 or later	2.2, 3.1 or later	7.1	7.1	DRDA	Error

Notes:

1. You cannot reload a portable package created under SQL/DS Version 2 Release 2 using the DRDA flow because it does not have the necessary information required for RELOAD PACKAGE command processing using DRDA flow.
2. Backward migration is also not possible; that is, you cannot reload DB2 Server for VSE & VM Version 7 Release 5 or later portable package with SQL/DS Version 2 Release 2.
3. Modifiable packages created using Extended dynamic statements cannot be reloaded using DRDA flow.
4. Fortran, and any other packages created using Extended dynamic statements that were originally preprocessed using SQLDS protocol, cannot be reloaded using DRDA flow.
5. Fortran, and any other packages created using Extended dynamic statements that were originally preprocessed using DRDA flow, cannot be reloaded using SQLDS protocol.

In using the RELOAD PACKAGE command, you must be either the owner of the program whose package you are trying to reload or a database administrator. DB2 Server for VSE & VM authorization checking grants the owner the RUN privilege after the following information is supplied to the Database Services Utility:

- The name of the command
- The identity of the package to be reloaded
- The replacement method (NEW or REPLACE) to use
  - If REPLACE, the disposition of previous package user privileges (KEEP or REVOKE)
- Optionally, in VSE only, the identity of additional application servers where the package is to be reloaded
- The identity of the input file.

Reloading a Package into an Application Server in Which Its Application Does Not Exist

To reload a package ported from another application server or from backup, proceed as follows:

In VSE

1. Provide the following Database Services Utility command:

`RELOAD PACKAGE (`*`owner`*`package-name)`

where *owner* is the name of the owner of the associated package, and *package-name* is the name of the package. If you omit *owner*, the database manager uses your user ID but still checks to ensure that you have the RUN privilege for the named package.

2. Specify the replacement method. Because the application associated with the package to be loaded does not exist for the application server (or application servers) being loaded, leave one space, and type:

   ```
   NEW
   ```
3. On the same record, enter the names of any additional application servers onto which the package is to be reloaded. Leave one space and put:

   `TO (`*`server-name`*`)`

   where *server-name* is the name of the other application server. If the package is to be reloaded onto several application servers, leave one space, then type:

   `TO (`*`application server1`*`,`*`application server2`*`,`*`application server3...`*`)`
4. On the same record, identify the input file. Leave one space, and type:

   `INFILE (`*`ddname`*`)`

   where *ddname* identifies the sequential input file on tape. Use the same *ddname* in a TLBL statement. If the input file is on DASD, specify PDEV(DASD) after the *ddname* and use the same *ddname* in a DLBL statement.
5. Submit the job to run.

An example of the RELOAD PACKAGE command is:

`RELOAD PACKAGE(MARCY.PROG3) NEW INFILE(PROGOUT3) TO(`*`server-name`*`)`

where PROG3 is the name of the package, MARCY is the owner, and server-name is the name of the other application server. The input data file PROGOUT3 is on tape.

In VM

1. Issue the SQLINIT command to initialize the user machine to the application server where the package is to be reloaded. If you have already done this, proceed to Step 2.
2. Create a control file to contain the command you construct in the following steps. See “Working with a Control File in DB2 Server for VM” on page 13 for detailed information on creating a control file.
3. Enter the command and package name. Type:

   `RELOAD PACKAGE (`*`owner.package-name`*`)`

   where *owner* is the name of the owner of the associated package, and *package-name* is the name of the package. If you omit *owner*, the database manager uses your user ID as the owner.
4. Indicate that you are loading a new package since the application associated with the package does not exist on the application server (or application servers) being loaded, by leaving one space and typing:

   ```
   NEW
   ```
5. On the same line, enter the names of any additional application servers to be reloaded. Leave one space; then type:

   `TO(`*`server-name`*`)`

   where *server-name* is the name of the other application server. If *several* application servers are to be reloaded, leave one space; then type:

   `TO(`*`application server1`*`,`*`application server2`*`,`*`application server3`*`...)`

Notes:

a. Your system must have database switching capability to access other application servers.

b. The use of TO means that any preceding CONNECT operations are not used, and TO uses the VM user ID as a default. In some situations, the user ID received at the target database is different from your VM user ID. For example, an entry in the COMDIR may change the user ID, or the target system may change it. If you do not want to use the TO clause procedure, issue the explicit CONNECT command as required, and use the RELOAD command without a TO clause. If the TO clause is not specified, the package is reloaded onto the currently connected application server only.

6. On the same line, enter the identity of the input file. Leave one space; then type:

   ```
   INFILE(ddname)
   ```

   where *ddname* identifies the input data file.

7. Store the control file.

8. In CMS, specify the necessary FILEDEF statements. When you specify the FILEDEF statement for the input data file, use the same *ddname* you assigned to the INFILE in this procedure. For general information about FILEDEF statements, see "Using File Definitions" on page 14. For command specific information, see "Using File Definitions with DB2 Server for VM UNLOAD and RELOAD PACKAGE Commands" on page 92.

9. Specify an SQLDBSU EXEC statement; if you did not specify FILEDEFs for the control and message files, use the default values in the SQLDBSU EXEC. For more information on the SQLDBSU EXEC, see "Using the SQLDBSU EXEC" on page 15.

An example of the RELOAD PACKAGE command is:

```
RELOAD PACKAGE(MARCY.PROG3) NEW TO(HOLIDAY) INFILE(PROGOUT3)
```

where PROG3 is the name of the package, and MARCY is the owner. The package is created in the HOLIDAY database and the input data file is PROGOUT3.

Reloading a Package into an Application Server to Update an Existing Application: Your system must have the capability to switch to other application servers.

To reload a package into an application server where a package with the same name already exists, proceed as in "Reloading a Package into an Application Server in Which Its Application Does Not Exist" on page 88, but type:

```
REPLACE
```

or

```
REPLACE REVOKE
```

where REPLACE indicates that the existing package is to be replaced by the input package with previous user privileges *intact*, and REPLACE REVOKE indicates that the existing package is to be replaced by the input package with previous user privileges *revoked*. An example of the RELOAD PACKAGE command is:

```
RELOAD PACKAGE(MARCY.PROG3) REPLACE REVOKE TO(PAYROLL) INFILE(PROGOUT3)
```

where PROG3 is the name of the package, and MARCY is the owner. The package exists in the PAYROLL database; therefore, you use REPLACE to replace it and REVOKE to revoke user privileges on the package. The input data file is PROGOUT3.

Authorizing the Use of Packages

When a package is reloaded, the owner of the package is assigned the appropriate run privilege. If NEW or REPLACE REVOKE is one of the RELOAD parameters, the only privilege for the package is the owner's run privilege. The user ID for the owner of the reloaded package must exist so that the necessary privileges can be granted. A user with DBA authority can reload a package without being the owner and can grant the RUN privileges to other users. For more information on package privileges, see the *DB2 Server for VSE & VM Application Programming* manual.

Errors with RELOAD

The Database Services Utility SET ERRORMODE CONTINUE command can be used with the RELOAD PACKAGE command. If an error occurs during reloading of the package on an application server, RELOAD processing ends on that application server. RELOAD processing then continues on subsequent application servers listed in the TO clause if ERRORMODE CONTINUE processing is in effect and the error is not severe.

Example of Authorizing the Use of Packages

Gene writes an application program GENE.TTIME to display each employee's working hours to date. The table containing this information is HOURS.TOTAL. This program is to be distributed to all the offices and installed by a user with DBA authority at each site.

When the DBA reloads this new package, only the owner (GENE) possesses the RUN privilege. This privilege is granted during the RELOAD. The user ID GENE must exist for the DBA to be able to grant the RUN privilege for the package GENE.TTIME. GENE must also have the necessary table privileges to run the package successfully.

Preprocessing and Distributing an Application

To preprocess and distribute the SQL application created by Gene in the above example, the company proceeds as follows:

1. The person with the user ID HOURS grants the SELECT privilege on HOURS.TOTAL to user ID GENE.
2. Gene creates the application program TTIME.
3. Gene unloads the package TTIME.

Setting Up to Run an Application

To set up and run the application described in "Preprocessing and Distributing an Application," the company proceeds as follows:

1. Tom, a DBA, grants the CONNECT privilege to user ID GENE.
2. Tom connects as HOURS.

3. Tom (connected as HOURS) grants the SELECT privilege on HOURS.TOTAL to user ID GENE.
4. Tom connects as GENE.
5. Tom (connected as GENE) reloads the package GENE.TTIME.
6. Tom (connected as GENE) grants the RUN privilege on GENE.TTIME to USERA, USERB, and USERC.
7. USERA, USERB, and USERC can now run the package GENE.TTIME to display each employee's total working hours to date.

Using File Definitions with DB2 Server for VM UNLOAD and RELOAD PACKAGE Commands

Use the Same File Definition for RELOAD as for UNLOAD

CMS FILEDEF command information for package RELOAD processing should be identical to the information in the FILEDEF command used when the file was created by the package's UNLOAD command processing.

Figure 48 shows the relationship of the load-program files and the appropriate definition facility (FILEDEF or SQLDBSU) for each.

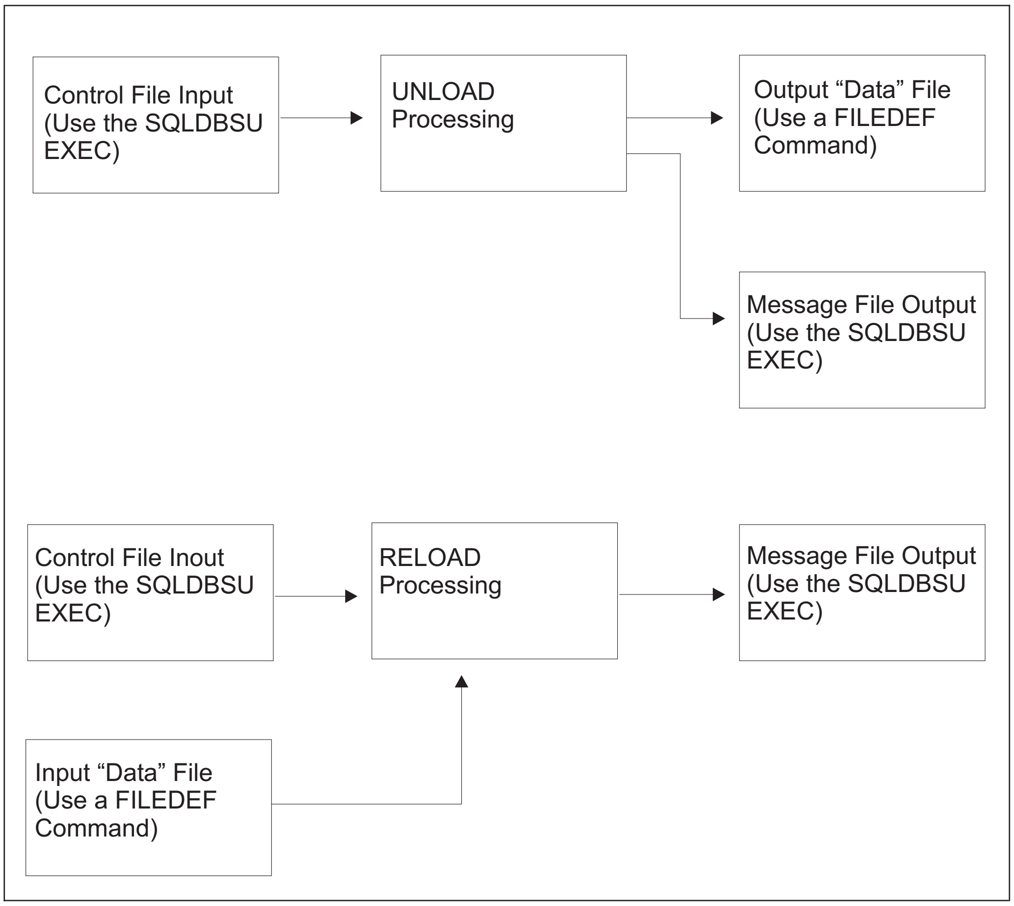

Control File Input (Use the SQLDBSU EXEC)
UNLOAD Processing
Output "Data" File (Use a FILEDEF Command)
Message File Output (Use the SQLDBSU EXEC)
Control File Inout (Use the SQLDBSU EXEC)
RELOAD Processing
Message File Output (Use the SQLDBSU EXEC)
Input "Data" File (Use a FILEDEF Command)

Figure 48. File Definition Diagram—UNLOAD PACKAGE and RELOAD PACKAGE

FILEDEFs Supporting UNLOAD and RELOAD PACKAGE

CMS FILEDEF commands must be used to define the input or output data files processed by these commands.

Except for the *ddname*, CMS FILEDEF command information for RELOAD command processing should be identical to the information in the FILEDEF command used when the file was created by the UNLOAD command processing.

If either a RECFM value other than FB or an LRECL value is specified by the CMS FILEDEF command, the value is ignored and overridden.

A sample CMS FILEDEF command defining a CMS file for UNLOAD or RELOAD command processing is:

```
FILEDEF DBSFILE DISK DBSFILE DATA A
```

For more information on FILEDEF parameters and options, see Appendix B, "FILEDEF Command Syntax and Notes," on page 249.

Chapter 6. Interpreting the Output of the Database Services Utility

You can encounter two types of Database Services Utility output: the information that the Database Services Utility feeds back to the report or message file, and the data and control information that you unload for backup or eventual reloading. Most often, you must understand the report output because it shows you what happened during the Database Services Utility processing. This chapter describes report and message file output.

This chapter builds on material presented in the foregoing chapters. Refer to the appropriate sections of the earlier chapters for basic concepts and procedures.

Understanding the Report and Message File Output

The report is a log of Database Services Utility processing activity on DB2 Server for VSE. You can use JCL to assign the output to a variety of output devices: printer, tape, or disk.

The message file is a log of Database Services Utility processing activity on DB2 Server for VM. The SQLDBSU EXEC or the CMS FILEDEF command can direct the message file to a variety of output devices; you can display or print its contents in three distinct forms.

Everything in a report or message file belongs in one of three categories:
- Command input
- System output
- Data.

Command Input (DB2 Server for VSE & VM)

All parts of a set of commands, SQL or Database Services Utility, are considered command input. Even the data embedded in a DATALOAD TABLE statement is command input. Record for record or line for line, this type of output matches the format of the (input) control file. A command-input record in the report starts with an arrow (------>).

System Output (DB2 Server for VSE & VM)

Except when suppressed by a Database Services Utility control parameter in a calling application program, all system messages, SQL and Database Services Utility, are sent to either the report or message file. A system-output record or line starts with a message identifier beginning with *ARI*.

Inclusion of Data in a Report (DB2 Server for VSE)

To include data in a report, use the LIST (YES) parameter in DATALOAD's INFILE subcommand. You can identify data in the report by the absence of arrows (------>) or message identifiers (ARI...). Figure 49 on page 96 shows a simulated report printout.

Inclusion of Data in a Message File (DB2 Server for VM)

To include data in a message file, use the LIST (YES) parameter in DATALOAD’s INFILE subcommand. Through the SELECT statement, the Database Services Utility allows a limited amount of system-user interaction. Do not use the utility as an alternative to ISQL, but if you are in a Database Services Utility session, you can enter SQL commands to query the database from your workstation (assigned as control file) without leaving the utility. You can identify data in the message file by the absence of arrows (------>) or message identifiers (ARI...). Figure 49 shows a simulated message-file printout.

```
ARI0801I DBS Utility started: 07/24/89 17:38:53.
         AUTOCOMMIT = OFF ERRORMODE = OFF
         ISOLATION LEVEL = REPEATABLE READ

------> CONNECT "TARA    " IDENTIFIED BY ********;
ARI8004I User TARA connected to database SQLDBA.
ARI0500I SQL processing was successful.
ARI0505I SQLCODE = 0  SQLSTATE = 00000  ROWCOUNT = 0

------> ACQUIRE PRIVATE DBSPACE NAMED TARASPACE;
ARI0500I SQL processing was successful.
ARI0505I SQLCODE = 0  SQLSTATE = 00000  ROWCOUNT = 0

------> CREATE TABLE DEPARTMENT (DEPTNO   CHAR(3)     NOT NULL,
------>                          DEPTNAME VARCHAR(36) NOT NULL,
------>                          MGRNO    CHAR(6)             ,
------>                          ADMRDEPT CHAR(3)     NOT NULL,
------>                          PRIMARY KEY (DEPTNO)) IN TARASPACE;
ARI0500I SQL processing was successful.
ARI0505I SQLCODE = 0  SQLSTATE = 00000  ROWCOUNT = 0

------> DATALOAD TABLE (DEPARTMENT)
------>      DEPTNO   1-3
------>      DEPTNAME 5-32
------>      MGRNO    34-39  NULL IF POS (34-39)='      '
------>      ADMRDEPT 41-43
------> INFILE(*)
ARI0852I DATALOAD TABLE processing started.
ARI8981I Dynamic statistic accumulation was disallowed
         for table 'TARA'.'DEPARTMENT',
         reason code = 01.
------> A00 SPIFFY COMPUTER SERVICE DIV. 000010 A00
------> B01 PLANNING                     000020 A00
------> C01 INFORMATION CENTER           000030 A00
------> D01 DEVELOPMENT CENTER                  A00
------> E01 SUPPORT SERVICES             000050 A00
------> D11 MANUFACTURING SYSTEMS        000060 D01
------> D21 ADMINISTRATION SYSTEMS       000070 D01
------> E11 OPERATIONS                   000090 E00
------> E21 SOFTWARE SUPPORT             000100 E00
------> ENDDATA
ARI0875I 9 row(s) loaded into table TARA.DEPARTMENT.
ARI8996I ...Begin UPDATE STATISTICS for TARA.DEPARTMENT.
ARI0855I DATALOAD processing successful.

------> SELECT * FROM DEPARTMENT;
```

Figure 49. Sample Output

In Figure 49, note that the arrows show command input. Each arrow corresponds to either a record in the input control card file or a line in the control file. Note also that the rest of the records or lines start with *ARI*, denoting messages.

Figure 50 illustrates the next part of the simulated report or message file printout.

```
SELECT * FROM DEPARTMENT                                    PAGE     1

DEPTNO DEPTNAME                             MGRNO  ADMRDEPT
------ ----------------------------------- ------ --------
A00    SPIFFY COMPUTER SERVICE DIV.         000010 A00
B01    PLANNING                             000020 A00
C01    INFORMATION CENTER                   000030 A00
D01    DEVELOPMENT CENTER                          A00
E01    SUPPORT SERVICES                     000050 A00
D11    MANUFACTURING SYSTEMS                000060 D01
D21    ADMINISTRATION SYSTEMS               000070 D01
E11    OPERATIONS                           000090 E00
E21    SOFTWARE SUPPORT                     000100 E00

ARI0850I SQL SELECT processing successful: Rowcount = 9
```

Figure 50. Sample Output Containing Data

The output is formatted such that the data in columns and rows as a normal table. Figure 50 is an example of the column (or tabular) form of output. No message- or command-input designations appear at the start of data records. Figure 51 shows the end of the simulated report printout.

```
ARI0802I End of command file input
ARI8997I ...Begin COMMIT processing.
ARI0811I ...COMMIT of any database changes successful.
ARI0809I ...No error(s) occurred during command processing.
ARI0808I DBS processing completed: 07/24/89/17:38:55.
```

Figure 51. Concluding Messages

Every Database Services Utility job ends with messages from the Database Services Utility that summarize the errors, if any, that occurred and give the completion timestamp, the system status, and a return code. For a complete listing of messages, see the *DB2 Server for VM Messages and Codes* manual.

If either command-file input or query results data has record or line lengths too wide for the page or screen, the Database Services Utility automatically switches to a block-form presentation. Figure 52 on page 98 shows a simulated printout in block form.

```
ARI0801I DBS Utility started: 11/13/89 17:42:51
         AUTOCOMMIT = OFF ERRORMODE = OFF
         ISOLATION LEVEL = REPEATABLE READ

------> COMMENT '**********************************************
------>          ***      BLOCK FORMAT PRINTOUT EXAMPLE      ***
------>          *********************************************'

------> CONNECT "MIKE    " IDENTIFIED BY ********;
ARI8004I User MIKE connected to database SQLDBA.
ARI0500I SQL processing was successful.
ARI0505I SQLCODE= 0  SQLSTATE = 00000  ROWCOUNT = 0

------> ALTER TABLE ACTIVITY ADD
------>      "FULL DESCRIPTION" VARCHAR(250);
ARI0500I SQL PROCESSING WAS SUCCESSFUL.
ARI0505I SQLCODE= 0  SQLSTATE = 00000  ROWCOUNT = 0

------> UPDATE ACTIVITY SET "FULL DESCRIPTION" = 'A FULL DESCRIPTION
------>  OF ACTIVITIES WOULD BE DISPLAYED HERE.  THE DESCRIPTION COU
------> LD OVERFLOW TO THE NEXT DISPLAY LINE FOR THE COLUMN BECAUSE
------> THE COLUMN CAN CONTAIN UP TO 250 DATA POSITIONS';
ARI0500I SQL processing was successful.
ARI0505I SQLCODE= 0  SQLSTATE = 00000  ROWCOUNT = 0

------>  SELECT * FROM ACTIVITY WHERE ACTNO < 30
           SELECT * FROM ACTIVITY WHERE ACTNO < 30             PAGE 1

*****            1   *****
ACTNO:  10     ACTKWD:  MANAGE     ACTDESC:  MANAGE/ADVISE
FULL DESCRIPTION: A FULL DESCRIPTION OF ACTIVITIES WOULD BE DISPLAY
                  ED HERE.  THE DESCRIPTION COULD OVERFLOW TO THE N
                  EXT DISPLAY LINE FOR THE COLUMN BECAUSE THE COLUM
                  N CAN CONTAIN UP TO 250 DATA POSITIONS

*****            2   *****
ACTNO:  20     ACTKWD:  ECOST      ACTDESC:  ESTIMATE COST
FULL DESCRIPTION: A FULL DESCRIPTION OF ACTIVITIES WOULD BE DISPLAY
                  ED HERE.  THE DESCRIPTION COULD OVERFLOW TO THE N
                  EXT DISPLAY LINE FOR THE COLUMN BECAUSE THE COLUM
                  N CAN CONTAIN UP TO 250 DATA POSITIONS

ARI0850I SQL SELECT processing successful: Rowcount = 2
```

Figure 52. Sample Printout of Block Output Format

As in a tabular format query result, the block format has a heading and page number; from there on, however, differences appear. Each row of the query result is presented in a separate block of lines preceded by a subheading that identifies the number of the row in the answer set. Individual fields of a row are preceded by their column names and are to be read from left to right, and top to bottom.

The list form of output is similar to block form. Figure 53 on page 99 shows an example of the list output format.

```
               SELECT * FROM DEPARTMENT                         PAGE 1


     ***ROW: 1
     DEPTNO:   A00
     DEPTNAME: SPIFFY COMPUTER SERVICES DIV.
     MGRNO:   000010
     ADMRDEPT:   A00
     FULL DESCRIPTION: A FULL DESCRIPTION OF THE DEPARTMENT WOULD BE
                       DISPLAYED HERE.  THE DESCRIPTION COULD OVERFLOW TO
                       THE NEXT DISPLAY LINE FOR THE COLUMN.
```

Figure 53. Sample Printout of List Output Format

The list output format resembles the block output format in that there is a heading and page number. Each row of the query result is presented in a separate list of records or lines preceded by a subheading that identifies the number of the row in the answer set. Individual fields of a row are preceded by their column names and are presented in separate records. The difference, however, is that with list output format, the output for each selected row begins on a new page, and the column name and data for each select-list column begins on a new output record or display line.

You can specify the format used by the Database Services Utility for SQL SELECT statement output by using either the SET FORMAT command or the FORMAT control parameter. If you do not specify the format to be used, the Database Services Utility uses column or block format as appropriate. For more information on the SET FORMAT command, refer to “SET FORMAT” on page 217. For a description of the control parameter FORMAT, see “Database Services Utility Control Parameters” on page 115.

Using the LIST Parameter on a DATALOAD Command

The example in Figure 54 shows the output from a DATALOAD command that was processed with the LIST(YES) option of the INFILE subcommand in effect.

```
---------> DATALOAD TABLE (SQLDBA.ACTIVITY) IF POS(1)='Y'
---------> ACTNO          2-11
---------> ACTDESC       12-111
---------> INFILE (* LIST(YES) CONTINUED(YES))                          Record 1.
ARI0852I DATALOAD PROCESSING STARTED.
 N2345678901THE DATA IN THIS RECORD IS NOT LOADED INTO THE TABLE. <-----
XY2345678902THIS DESCRIPTION OF AN ACTIVITY FILLS UP A WHOLE PHYSICAL INPUT
  RECORD AND CONTINUES.                                       <-----Record 2.
XY2345678903THIS ACTIVITY DESCRIPTION DOES NOT FILL THE RECORD. <-----
                                <-----------------
 --------> ENDDATA
 ARI0875I 2 row(s) loaded into table SQLDBA.ACTIVITY.  This blank line   Record 3.
 ARI0855I DATALOAD processing successful.               contains positions
                                                        80-111 of data
                                                        record 3.
```

Figure 54. Using YES in the LIST Parameter

With continued record processing, each field of the records must contain the maximum number of characters. The second and third records’ ACTDESC fields are 100 characters in length because they are to be loaded (position 1 is Y). The system does not check the length of the first record's ACTDESC field because this record is not to be loaded (position 1 is N).

The example in Figure 55 shows output for a DATALOAD command that was processed using the LIST(NO) option of the INFILE subcommand:

```
------> DATALOAD TABLE (SQLDBA.ACTIVITY) IF POS(1)='Y'
------> ACTNO          2-11
------> ACTDESC       12-111
------> INFILE (* LIST(NO) CONTINUED(YES))
ARI0852I DATALOAD PROCESSING STARTED.
------> ENDDATA
ARI0875I 2 row(s) loaded into table SQLDBA.ACTIVITY.
ARI0855I DATALOAD processing successful.
```

Figure 55. Using NO in the LIST Parameter

Reading Report and Message-File Output in Error Recovery

Reading messages, command input, and data in output reports and message files is an important task performed frequently by users of the Database Services Utility.

To read DB2 Server for VSE report output to recover from an error, proceed as follows:

1. Check the messages at the end of the report to determine whether the utility job ended without error. Message records begin with *ARI*.
2. If the job ran with errors, read all the messages, working backward from the end until you reach the point of the (initial) error message. Note this spot in the report so that you can easily return to it.
3. If data is included in the report, scan the query result to see how serious the error is.
4. If you find the cause of the error, take corrective action and run the job again; if you do not find the cause, inspect the command input for syntax errors. Command-input lines begin with arrows (------>).
5. If the cause of the error is still unknown, determine whether it is a DB2 Server for VSE error or a Database Services Utility error. Look up the error message in the *DB2 Server for VSE Messages and Codes* manual and follow the recommended recovery procedure, as applicable.
6. If the cause of the error appears to be related to the Database Services Utility, review information in Chapter 9, "Error Handling and Debugging," on page 223, and apply appropriate corrective action.
7. If the error persists, see your database administrator.

To read DB2 Server for VM message file output to recover from an error, proceed as follows:

1. Print or display the message file to be read.
2. Check the messages at the end of the file to determine whether the utility job ended without error. Message lines begin with *ARI*.
3. If the job did not run without errors, read all the message lines, working backward from the end until you reach the point of the (initial) error message. Note this spot in the message file so that you can easily return to it.
4. If data is included in the message file, scan the query result to see how serious the error is.

5. If you find the cause of the error, take corrective action and run the job again; if you do not find the cause, inspect the command input for syntax errors. Command-input lines begin with arrows (------>).
6. If the cause of the error is still unknown, determine whether it is a DB2 Server for VM error or a Database Services Utility error. Look up the error message in the *DB2 Server for VM Messages and Codes*, and follow the recommended recovery procedure, as applicable.
7. If the cause of the error appears to be related to the Database Services Utility, review information in Chapter 9, “Error Handling and Debugging,” on page 223, and apply appropriate corrective action.
8. If the error persists, see your database administrator.

Part 2. Reference

This part of the manual presents additional information on the calling and running of the Database Services Utility. The material in this section is of primary interest to database application programmers and system programmers; it includes the following topics:

- Database Services Utility use from application programs
- Database Services Utility commands: reference
- Database Services Utility error handling and debugging
- Database Services Utility performance considerations.

For further reference material in the form of sample tables, see Appendix A, “Sample Tables.” Refer to Appendix B, “FILEDEF Command Syntax and Notes,” on page 249, for a syntax diagram and usage notes about the CMS FILEDEF command.

Chapter 7. Using the Database Services Utility from Application Programs

This chapter describes the procedures required to initiate Database Services Utility processing from application programs and how to use the Database Services Utility application program interface. Rules for naming objects and lists of reserved words are also provided.

In DB2 Server for VSE

The Database Services Utility is an application program; like any other program, it must be preprocessed before you can run it. Usually, the utility is preprocessed during database installation. At that time, the RUN privilege to use the utility is granted to prospective users; you must possess the RUN privilege to use the utility. Your system programmer can tell whether the Database Services Utility is properly installed and whether you are authorized to use it.

The *ddname* parameter of a Database Services Utility command identifies an EBCDIC, standard-label sequential data file needed by that command for input or output. Magnetic tape with a fixed, unblocked record format and a logical record length and block size of 2 048 bytes is used as a default for these sequential files. (The record format required varies for each command.)

As indicated in the command descriptions, you can override the defaults and assign files to a direct access storage device (DASD). A sequential file allocated to magnetic tape can reside on any device supported by the VSE DTFMT macro; files allocated to direct access storage can reside on any device supported by the VSE DTFSD macro. An exception to this is VSAM managed SAM files, which do not support spanned records. Spanned records are used by UNLOAD TABLE/DBSPACE and RELOAD TABLE/DBSPACE processing and, in some cases, by DATALOAD and DATAUNLOAD.

You can invoke the Database Services Utility in a VSE batch partition or in a VSE/ICCF interactive partition. The Database Services Utility does not use VSE dynamic device assignment. Logical units SYS004 and SYS005, respectively, must be used for input and output files allocated to magnetic tape: SYS006 and SYS007, respectively, must be used for input and output files allocated to direct access storage. Under VSE, all magnetic tape files processed by the Database Services Utility must be EBCDIC standard-label files, and only magnetic tape input files are rewound.

Special consideration should be given to the database log when using the Database Services Utility to load large amounts of data. The log must be large enough to contain all the log data generated during Database Services Utility RELOAD DBSPACE, RELOAD TABLE, or DATALOAD command processing. Log space used as a result of Database Services Utility processing is not freed until an SQL COMMIT or ROLLBACK is executed. If the log space is filled by Database Services Utility processing, a Database Services Utility processing error occurs. Your system programmer can tell whether there is enough log space to contain all the log data generated during Database Services Utility command processing.

Single User Mode Job Control

A minimum 2000K byte virtual partition is recommended to execute the Database Services Utility with single user mode for VSE archive mode on. The sample VSE job control statements in Figure 56 invoke the Database Services Utility in single user mode with DB2 Server for VSE archive mode on:

```
 1   // JOB    DBSUTIL
 2   // EXEC   PROC=ARIS75PL
     // EXEC   PROC=ARIS75DB
 3   // TLBL   file name,.......
 4   // DLBL   file name,.......
 5   // ASSGN  SYS004,.........
 6   // ASSGN  SYS005,.........
 7   // ASSGN  SYS006,.........
 8   // ASSGN  SYS007,.........
 9   // EXEC   ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,LOGMODE=A,PROGNAME=ARIDBS'
10 ....Database Services Utility control commands and optional user data
11   /*
12   /&
```

Figure 56. Single User Mode Job Control

Each job control statement is described below:

1 JOB Statement

Identifies and initiates the job control.

2 EXEC PROC=ARIS75PL and EXEC PROC=ARIS75DB

When the database manager is installed, your installations have the option of generating a starter database as described in the *DB2 Server for VSE Program Directory*. The procedure ARIS75PL contains the job control statements that identify the DB2 Server for VSE & VM library. Procedure ARIS75DB contains the job control statements that are required to access the starter database. You must execute a different procedure to access a different database. Alternatively, you can code the actual DB2 Server for VSE database and library definition job control statements in place of the EXECUTE PROCEDURE statements.

To determine the database and library definition statements required, contact the person who installed the database or refer to the *DB2 Server for VSE Program Directory*, which contains a description of the starter database and library definition job control statements and procedures.

3 TLBL Statement

This job control statement is optional. It identifies a sequential (SAM) magnetic tape file used for Database Services Utility input and/or output data. The file can reside on any type of volume supported by the VSE DTFMT macro.

Note: Tape files processed by the Database Services Utility under VSE must be EBCDIC, standard-label files.

Any number of these commands (each having a unique file name) can be included in the job control. Each file name defined is normally referenced by a *ddname* parameter in a Database Services Utility command that is contained in the SYSIPT control command input. Input magnetic tapes are rewound by Database Services Utility OPEN processing, but output magnetic tapes are not rewound. For input files other than the first file on a tape volume, a file sequence number must be specified, corresponding to the original position of that file on the tape.

4 DLBL Statement

This job control statement is optional. It identifies a sequential (SAM) DASD file used for Database Services Utility input or output data. The file can reside on any type of volume supported by the VSE DTFSD macro.

Any number of these statements (each having a unique file name) can be included in the job control. Each file name defined is normally referenced by a *ddname* parameter in a Database Services Utility command contained in the SYSIPT control command input. Job control EXTENT statements are required to complete the description of the file identified by the DLBL statement.

5 ASSGN SYS004 (Tape Input File)

This job control statement is required if a Database Services Utility sequential (SAM) input data file is allocated to a magnetic tape device. It defines the logical unit SYS004 for a tape input file.

6 ASSGN SYS005 (Tape Output File)

This job control statement is needed if a Database Services Utility sequential (SAM) output data file is allocated to a magnetic tape device. It defines the logical unit SYS005 for a tape output file. A large block size is recommended for a tape output file to improve performance.

7 ASSGN SYS006 (DASD Input File)

This job control statement is needed if a Database Services Utility sequential (SAM) input data file is allocated to a direct access device. It defines the logical unit SYS006 for a DASD input file.

8 ASSGN SYS007 (DASD Output File)

This job control statement is needed if a Database Services Utility sequential (SAM) output data file is allocated to a direct access device. It defines the logical unit SYS007 for a DASD output file.

9 EXEC Statement for Single User Mode

This statement identifies the database entry point (ARISQLDS), and contains the required SIZE=AUTO specification and the job control parameters to execute the Database Services Utility program with single user mode.

The job control parameters that you must specify are:
- SYSMODE=S
- PROGNAME=ARIDBS

SYSMODE=S indicates that you want single user execution mode. PROGNAME=ARIDBS identifies the Database Services Utility program entry point. LOGMODE=A identifies that the database manager should operate with archive and logging on.

All single user mode startup parameters and log mode considerations are described in the *DB2 Server for VSE System Administration* manual. Consult a system programmer about the startup parameters; your installation might specify additional parameters for performance reasons.

10 SYSIPT Control Statement and User Data Input

Database Services Utility control commands and user data input.

11 End SYSIPT Control Statement Input Indicator

Indicates the end of Database Services Utility SYSIPT input when SYSIPT is assigned to the reader file.

12 End of Job Indicator

Indicates the end of the job.

Single User Mode Job Control Example

The sample job control and commands shown in Figure 57 run the Database Services Utility with single user mode to perform the following functions:

- Unload the table SQLDBA.DEPARTMENT to the first file on a scratch tape.
- Unload the table SQLDBA.ACTIVITY to the second file on the tape.

```
// JOB DBS UTILITY EXAMPLE VSE SINGLE USER MODE JOB CONTROL
// EXEC PROC=ARIS75PL              <--DB2 Server for VSE Production Library Definition
// EXEC PROC=ARIS75DB              <--DB2 Server for VSE Starter Database Definition
// TLBL TAPE1,'DBSU-FILE1',0,SQLDAT,1,1          <--Tape File#1
// TLBL TAPE2,'DBSU-FILE2',0,SQLDAT,1,2          <--Tape File#2
// ASSGN SYS005,280                              <--Tape output
// MTC REW,SYS005                                <--Rewind tape
// EXEC ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,LOGMODE=Y,PROGNAME=ARIDBS'
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
UNLOAD TABLE(DEPARTMENT) OUTFILE(TAPE1)
UNLOAD TABLE(ACTIVITY) OUTFILE(TAPE2)
/*
/&
```

Figure 57. Single User Mode Job Control Example

Multiple User Mode Job Control

The VSE job control statements in Figure 58 invokes the Database Services Utility in multiple user mode:

```
//   JOB    DBSUTIL
//   EXEC PROC=ARIS75PL          <--DB2 Server for VSE Production Library Definition
//   TLBL   file name,.......
//   DLBL   file name,.......
//   ASSGN  SYS004,.........
//   ASSGN  SYS005,.........
//   ASSGN  SYS006,.........
//   ASSGN  SYS007,.........
//   EXEC   PGM=ARIDBS,SIZE=AUTO,PARM='DBNAME(SQLDB1_TOR_INV)'
 ....DBS control commands and optional user data
/*
/&
```

Figure 58. Multiple User Mode Job Control

The job control statements in Figure 58 do the same things as the corresponding statements in Figure 56 on page 106. The database manager must already be running when you invoke the Database Services Utility (or any other application program) with multiple user mode. To execute the Database Services Utility with multiple user mode, at least a 200K byte virtual partition is recommended.

DBNAME=SQLDB1_TOR_INV identifies the application server on which to process the Database Services Utility job. If the DBNAME parameter is not specified, the default application server is accessed.

Multiple User Mode Job Control Example

The example job control and commands shown in Figure 59 run the Database Services Utility with multiple user mode to perform the following functions:

- Unload the dbspace PUBLIC.SAMPLE to a DASD file.
- Reload the dbspace PUBLIC.SAMPLE from the same DASD file to reorganize the data for all tables in the dbspace.

```
// JOB DBS UTILITY EXAMPLE VSE MULTIPLE USER MODE JOB CONTROL
// EXEC PROC=ARIS75PL               <--DB2 Server for VSE Production Library Definition
// DLBL DASDI,'DBSU-FILE',0                     <--DASD input file
// EXTENT SYS006,sqlwkl,1,0,57,76               <--DASD input file
// ASSGN SYS006,150                             <--DASD input
// DLBL DASD0,'DBSU-FILE',0                     <--DASD output file
// EXTENT SYS007,sqlwkl,1,0,57,76               <--DASD output file
// ASSGN SYS007,150                             <--DASD output
// EXEC PGM=ARIDBS,SIZE=AUTO,PARM='DBNAME(SQLDB1_TOR_INV)'
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
UNLOAD DBSPACE(PUBLIC.SAMPLE) OUTFILE(DASD0 PDEV(DASD))
RELOAD DBSPACE(PUBLIC.SAMPLE) PURGE INFILE(DASDI PDEV(DASD) BLKSZ(2048))
/*
/&
```

Figure 59. Multiple User Mode Job Control Example

In DB2 Server for VM

The Database Services Utility operates in the user's virtual machine with the application server in either single user mode or multiple user mode.

The IBM-supplied SQLDBSU EXEC invokes the utility. This EXEC accepts optional parameters identifying the Database Services Utility input control file (*ddname*=SYSIN) and the Database Services Utility output message file (*ddname*=SYSPRINT). It also accepts other parameters necessary to run the database system in single or multiple user mode.

If the optional parameters identifying the Database Services Utility SYSIN and SYSPRINT files are not specified during startup of the SQLDBSU EXEC, the user can either define these files by using the CMS FILEDEF commands or run the utility with the SQLDBSU EXEC defaults, which assign the SYSIN and SYSPRINT file to the terminal.

Other DASD or tape input or output data files referenced by the Database Services Utility commands in the control file must be defined by the user with CMS FILEDEF commands before the SQLDBSU EXEC command is issued. Tape file processing and file definition restrictions apply to Database Services Utility input and output data files. Refer to the *DB2 Server for VM System Administration* manual for a description of tape file processing support. The *VM/ESA: CMS Command Reference* manual describes the CMS FILEDEF command.

Notes:

1. VM/ESA system in ESA mode is supported in the Database Services Utility only in XA-toleration mode. In this mode, the utility is always loaded and run below 16MB.
2. CMS subset mode is not supported by the Database Services Utility.

Names and Identifiers

General Rules for Naming Data Objects

The *DB2 Server for VSE & VM Application Programming* manual contain the formal definition of the SQL language and naming conventions. Briefly, the names of data objects (such as tables, columns, indexes, synonyms, or dbspaces) must be a particular kind of character string called an *identifier*. SQL identifiers must begin with a letter or number. They can contain up to 18 uppercase and lowercase letters, numbers, and underscores.

Note: If you need national language character translation for lowercase terminal input, the CMS SET INPUT *xx yy* command can be used. Refer to the *VM/ESA: CMS Command Reference* manual for more information.

The preprocessor used by the Database Services Utility for dynamic SQL statement processing converts DB2 Server for VSE & VM identifiers to uppercase if they are not in double quotation marks. For example, these two identifiers are identical to the system:

```
department      DEPARTMENT
```

If you want the system to recognize the lowercase letters in the identifier, enclose the identifier in double quotation marks. For example:

```
"department"
```

Qualifying Object Names

If a data object (such as a table, dbspace, or view) is owned by another user, you need to qualify references to the object by concatenating the owner's user identifier as in the following figure:

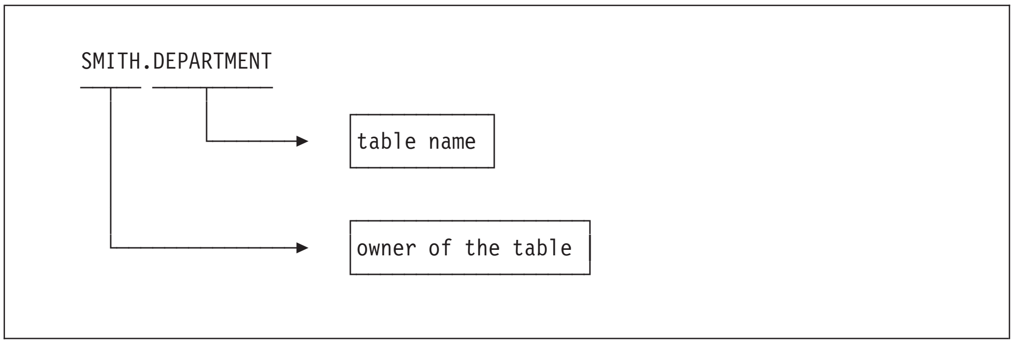


Figure 60. Object Names

The period (.) is the concatenation symbol.

You can access another user's table only if you know that person's user identifier and have the appropriate authorization to access that table.

When you concatenate a user ID to a table name, you fully qualify the table. That is, *owner.table-name* uniquely identifies a table in the database. For example, there can never be two SMITH.DEPARTMENT tables in the database at the same time.

Use fully qualified object names until you are an experienced DB2 Server for VSE & VM user. By fully qualifying database object names, you avoid confusion and errors.

Using Special Characters and Blanks within Identifiers

An identifier can contain blanks (but must not begin or end with blanks) or special symbols if you enclose them in double quotation marks. For example:

```
"RESEARCH EMPLOYEES"
```

You should not, however, use double quotation marks within an SQL identifier. The following is not a valid identifier:

```
"EMP"13"TABLE"
```

Reserved Words

SQL Reserved Words

A list of SQL reserved keywords can be found on “SQL Reserved Words” on page xvii. Do not use these words in SQL statements except:
- With their defined meaning in the SQL syntax
- As host variables (preceded by a colon).

In particular, do not use them as names for tables, indexes, columns, views, or dbspaces unless they are enclosed in double quotation marks (").

Database Services Utility Reserved Words

In addition to the SQL reserved keywords, do not use the following keywords in Database Services Utility commands as the name for a table, view, column, or dbspace unless you enclose the name in double quotation marks ("):

Table 6. Names to Avoid Using

DATALOAD	INMOD	RELOAD	SCHEMA
DATAUNLOAD	OUTFILE	REORGANIZE	UNLOAD
INFILE	REBIND		

Using Reserved Words as Identifiers

If an identifier is the same as one of the SQL keywords listed in this chapter, you must enclose the name in quotation marks. For example, you can use

```
"SELECT"
```

as a name, but if it is not delimited with quotation marks,

```
SELECT
```

is interpreted as a keyword.

Using the Database Services Utility from Programming Languages

You can invoke the Database Services Utility program from an assembler language, PL/I, C, or COBOL program by using the Database Services Utility entry point ARIDBS. (You cannot invoke the Database Services Utility from a Fortran program.)

DB2 Server for VSE

If the Database Services Utility program is link-edited with a user program, you must link-edit these modules in addition to those required for a normal DB2 Server for VSE application program:

```
ARISYSDD
ARIDSQLA
ARIDDFP
```

If you use an assembler language CDLOAD instruction in the calling program, you need not link-edit the above modules. (See the following section "Using the Database Services Utility from an Assembler Program" on page 113.)

You need not have an SQL CONNECT statement in the input control card file if the application that invokes the utility has already executed one. All authorization checking is based on the user ID supplied in the last CONNECT statement that was executed by the database manager.

ASSGN, TLBL, and DLBL commands required by the Database Services Utility must be present in the job control, and SIZE=AUTO must be specified on the EXEC command for the main program. The Database Services Utility processes all input control card file records from SYSIPT before returning control to the routine that invoked it.

DB2 Server for VM

The application program must be link-edited with ARIDBS, a member in the ARISQLLD LOADLIB, along with any other modules required for a normal DB2 Server for VM application program.

The application program can execute a CONNECT statement, supply a CONNECT statement in the Database Services Utility control file input, or take advantage of the implicit CONNECT support. All authorization checking is based on the connected user ID.

The FILEDEF commands for the Database Services Utility input control file, the output message file, and any input/output data files referenced by Database Services Utility commands must be executed before the utility is invoked. The utility processes all control file records from SYSIN before returning control to the program that invoked it.

Addressing Mode

Although the database manager can be run in 24-bit or 31-bit addressing mode, you must run the Database Services Utility in 24-bit mode.

In single user mode, if the database manager is being run in 31-bit mode, the addressing mode is switched to 24-bit mode before the Database Services Utility is started. When you call the Database Services Utility from an application program, the addressing mode is switched to 24-bit mode and the addressing mode of the application program is restored upon return. You must, however, ensure that any parameters passed by the application program to the Database Services Utility do not reside above the 16MB (MB equals 1,048,576 bytes) line.

See the *DB2 Server for VSE System Administration* or the *DB2 Server for VM System Administration* manuals for more information on addressing modes.

Register Contents for Database Services Utility Dynamic Startup

The Database Services Utility uses the following register content on entry:

- Register 0 is not used in DB2 Server for VSE; in DB2 Server for VM, it can contain the same content as Register 1.
- Register 1 can contain the address of a standard parameter address list to pass control parameters to the Database Services Utility program.
- Registers 2–12 are ignored.
- Register 13 contains the address of an area of 18 full-words to be used as a register-save area by the Database Services Utility program.
- Register 14 contains the return address for the Database Services Utility program.
- Register 15 contains the Database Services Utility entry point address.

When dynamically invoked, all registers except register 15 are restored by the Database Services Utility before to returning by way of register 14 to the invoking program. Register 15 contains the final return code from Database Services Utility processing.

Using the Database Services Utility from an Assembler Program

DB2 Server for VM

You can invoke the Database Services Utility program from an assembler language program. You must include the Database Services Utility program in the load module with the invoking program and follow the register conventions described above. The format of the CALL to invoke the utility is:

```
CALL ARIDBS
```

DB2 Server for VSE

You can invoke the Database Services Utility program from an assembler program in either of the ways described in the CALL macro description contained in the *VSE/Advanced Functions Macro Reference*. If you include the Database Services Utility program in the load module with the invoking program, the format of the CALL statement is the same as in DB2 Server for VM.

If you do not include the Database Services Utility program in the load module with the invoking program, use the following sequence of instructions to invoke the Database Services Utility program:

```
CDLOAD ARIDBS
LR     15,1
CALL   (15)
```

When using the above sequence of instructions, do not specify SIZE=AUTO on the EXEC command for the main program.

Using the Database Services Utility from a C Program

You can invoke the Database Services Utility program from a C program. ARIDBS must be declared to the compiler as an external entry point. ARIDBS must also be defined as having OS linkage using *#pragma linkage (ARIDBS, OS);*. The format of the C CALL command is:

```
ARIDBS(CLTYPEID,PARMSTR);
```

where CLTYPEID and PARMSTR can be declared as:

```
static char CLTYPEID[7]="DBSU01 ";

struct{short int        PARMLEN;
       char              PARMDATA [80];
     }PARMSTR;
```

Using the Database Services Utility from a COBOL Program

A main program written in COBOL can invoke the Database Services Utility program by using the linkage conventions described for calling assembler programs in the *DOS Full American National Standard COBOL Compiler and Library, Version 3, Programmer's Guide*. The Database Services Utility entry point name ARIDBS must be used in the COBOL CALL command used to invoke the Database Services Utility program. The format of the COBOL CALL command varies depending on whether the COBOL compiler was generated with single (') or double (") quotes as delineators:

```
CALL 'ARIDBS' USING CALLTYPEID PARMSTRING.
```

or

```
CALL "ARIDBS" USING CALLTYPEID PARMSTRING.
```

where CALLTYPEID and PARMSTRING can be declared as:

```
01 CALLTYPEID       PIC X(6) VALUE'DBSU01'.
01 PARMSTRING.
   49 PARMLEN       PIC S9(4) COMP.
   49 PARMDATA      PIC X(80).
```

Using the Database Services Utility from a PL/I Program

You can invoke the Database Services Utility program from a PL/I program by using the facilities of the IBM PL/I Optimizing Compiler Program Product. ARIDBS must be declared to the compiler as an external entry point with the ASSEMBLER and INTER options. The format of the PL/I CALL command is:

```
CALL ARIDBS(CLTYPEID,PARMSTR);
```

where CLTYPEID and PARMSTR can be declared as:

```
DCL 1 CLTYPEID       CHAR(6) INIT('DBSU01');

DCL 1 PARMSTR,
      2 PARMLEN      BINARY FIXED(15),
      2 PARMDATA     CHAR(80);
```

Using the Database Services Utility Application Program Interface

The interface described in this section allows a calling program or EXEC (in VM) to pass Database Services Utility control parameters or a single SQL statement or both. These utility control parameters provide the caller with the means to:

- Suppress all or portions of the messages written by Database Services Utility processing
- Identify the SQL SELECT statement output format
- Suppress SQL COMMIT and SQL ROLLBACK processing
- Determine the location where print data begins in the message file record
- Control the isolation level under which the Database Services Utility operates.

The calling program or EXEC can also pass a single SQL statement to the Database Services Utility for immediate processing by means of the call parameter list. Only SQL statements currently supported by Database Services Utility processing can be supplied. If an invalid parameter or Database Services Utility command is passed in the parameter list, it is processed as an SQL statement, and an error occurs.

Figure 61 shows the control parameters that can be used when invoking the Database Services Utility.

```
LINEWIDTH(www) or LW(www)PROMPTS(NO)
MESSAGES(SQLONLY)
MESSAGES(NONE)
FORMAT(CL)
FORMAT(LO)
PAGECTL(NO)
ENDLUW(NO)
ISOL(CS)
ISOL(UR)
```

Figure 61. Control Parameters

Control Parameters

Database Services Utility Control Parameters: This section lists and describes the Database Services Utility control parameters.

LINEWIDTH(*www*) or LW(*www*)

specifies the maximum number of print data positions used in a message file record containing SQL SELECT statement output. The default value for *www* is 120. In DB2 Server for VM, if the Database Services Utility message file (*ddname*=SYSPRINT) is assigned to the terminal, the number of print data positions used for the SQL SELECT statement defaults to 80. The value *www* can range from 60 to 256 but must be less than the logical record length of the report or message file. For example, if the logical record length is 100, the widest line you can set is LINEWIDTH(99).

Notes:

1. The utility always supplies an American Standards Association (ASA) control character in the first position of the print record. The second through nth positions of the print record are the print data positions. If the value *www*+1 is less than the print record length, all unused print data positions in the print record contain a blank (hex 40).
2. The Database Services Utility report record length is always 121.
3. The minimum message file record length is 81. If the control parameter PAGECTL(NO) is specified, the minimum message file record length is 80.
4. If the value *www* is equal to or greater than the print record length, an error occurs.

PROMPTS(NO)
suppresses Database Services Utility write-to-operator (WTO) messages. The WTO messages appear on the user's terminal or, in VSE, on the operator console display.

MESSAGES(SQLONLY)
suppresses the messages normally generated by Database Services Utility processing except for:

- SQL messages (ARI0500 through ARI0519)
- Message ARI0803E identifies an invalid command
- Message ARI0838E identifies an invalid CONNECT statement
- Message ARI0850I is generated after an SQL SELECT statement is successfully processed
- Message ARI0856E is generated when an error occurs during the execution of an SQL statement initiated by Database Services Utility processing
- Message ARI0884I indicates a command was processed
- Message ARI8999E: indicates an invalid control parameter or command was passed by means of the Database Services Utility parameter list
- Other Database Services Utility messages normally written to the terminal, or in VSE, to the operator console display device.

MESSAGES(NONE)
suppresses all Database Services Utility messages. Only the Database Services Utility return codes indicate the status of the processing performed.

FORMAT(CB)
is not supported.

FORMAT(CL)
formats the output of an SQL SELECT statement in either column format or list format.

FORMAT(LO)
formats the output of an SQL SELECT statement using only the list format.

PAGECTL(NO)
causes the Database Services Utility to:

- Use a default of 32767 lines per page for Database Services Utility report or message file output instead of 60 lines per page
- Write display lines to the Database Services Utility report or message file without a printer control character in position 1
- Suppress page number heading line(s) in SQL SELECT statement output.

The SET LINECOUNT command can override the default lines per page used by Database Services Utility processing.

DB2 Server for VM Only

A user-supplied FILEDEF command defining SYSPRINT to the terminal should specify RECFM F or RECFM FB. If RECFM F or RECFM FB is not specified, the first character of each Database Services Utility message file display line is truncated.

The default FILEDEF SYSPRINT command issued by the SQLDBSU EXEC defines the Database Services Utility message file record without a printer control character. This default command is:

```
FILEDEF SYSPRINT TERMINAL (RECFM F LRECL 120
```

ENDLUW(NO)

indicates that the Database Services Utility should not end the logical unit of work before returning to the calling program or, in VM, EXEC. The Database Services Utility does not issue an SQL COMMIT statement at the end of Database Services Utility processing if this parameter is specified.

Only the end of Database Services Utility COMMIT processing is suppressed by this control parameter. Other Database Services Utility COMMIT and ROLLBACK processing is not suppressed. However, when the ENDLUW(NO) control parameter is specified, and ERRORMODE CONTINUE is in effect, all Database Services Utility COMMIT and ROLLBACK processing is suppressed. In a VM system note that the SET ERRORMODE CONTINUE command is in effect when the Database Services Utility input control file is assigned to a terminal or a SET ERRORMODE CONTINUE command is processed. And, in a VSE system, the SET ERRORMODE CONTINUE command is in effect when a SET ERRORMODE CONTINUE command is processed.

Note: System-initiated COMMIT or ROLLBACK processing cannot be suppressed by any means.

ISOL(CS), ISOL(UR)

indicates that the Database Services Utility should operate under cursor stability or uncommitted read isolation level. The default mode of utility processing is repeatable read isolation level; however, if you are accessing a remote application server, the isolation level for the Database Services Utility is always set to CS and the SET ISOLATION command has no effect.

SQL Statement Parameter: If an SQL statement is supplied by means of the call parameter list, the Database Services Utility does not read the (input) control file SYSIPT. Any SQL statement currently supported by Database Services Utility processing can be supplied. This includes all SQL statements except those restricted to use in SQL application programs. One or more of the utility control parameters may precede the SQL statement. Utility commands are not supported in the call parameter list.

If an invalid parameter or a Database Services Utility command is passed in the parameter list, it is processed as an SQL statement, causing an error.

When an SQL statement is supplied in the Database Services Utility parameter list and it is not preceded by the ENDLUW(NO) control parameter, Database Services Utility processing ends the logical unit of work after the command is processed.

An SQL COMMIT statement is issued if the command is processed without errors. An SQL ROLLBACK statement is issued if a command error occurs.

If the ENDLUW(NO) control parameter precedes the SQL statement, the Database Services Utility issues neither an SQL COMMIT statement nor an SQL ROLLBACK statement after the SQL statement is processed.

Using Control Parameters with DB2 Server for VM: Use control parameters in a calling program or an EXEC when you want to set up the Database Services Utility environment and execute all the commands from one file. Figure 62 shows a REXX EXEC that directs the output messages to the terminal, executes the Database Services Utility, and passes control parameters and an SQL statement to the utility. You do not need to define the input requirements because the SQL statement is in the EXEC.

```
/* Example DBS Utility Control Parameters in REXX EXEC */
ADDRESS CMS
'FILEDEF ARISQLLD DISK ARISQLLD LOADLIB Q (NOCHANGE'
'FILEDEF SYSPRINT TERMINAL (NOCHANGE RECFM F LRECL 120'
'NUCXLOAD ARIDBS ARIDBS ARISQLLD'
'ARIDBS PROMPTS(NO) FORMAT(LO) ENDLUW(NO) SELECT * FROM DEPARTMENT'
ADDRESS
EXIT RC
```

Figure 62. Using the Database Services Utility Control Parameters in DB2 Server for VM

Using the Database Services Utility Interface Conventions

Database Services Utility control parameters or an SQL statement or both can be supplied using standard program (or in VM, EXEC interface) conventions.

Note: All system parameter string restrictions apply to Database Services Utility processing. These restrictions are not overridden by Database Services Utility parameter processing.

DB2 Server for VM Convention Formats: There are two format conventions; they are:

Format 1 - EXEC Program Interface Conventions: Your EXEC or program must follow these register conventions:

Register 0	The address of the parameter address list
Register 13	The address of the invoking program's 18 full-word save area
Register 14	The Database Services Utility return address
Register 15	The Database Services Utility entry point address on entry; the Database Services Utility processing return code on exit.

Your EXEC or program Database Services Utility parameter address list is to follow these conventions:

Position (per Byte)	Contents
1–4	Address of command identifier (not checked)
5–8	Address of PARAMETERSTRING
9–12	Address of PARAMETERSTRINGEND+1

Register 0 Parameter List Considerations:

Database Services Utility processing determines whether startup parameters are identified by register `0` by interrogating the contents of register `0` on entry to the Database Services Utility. Register `0` parameter processing is not performed if any of the following conditions occurs:
- Register `0` contains the address of the ARIDBS module.
- Content of register `0` equals `0`.
- Length of the parameter string is less than 1.
- Length of the parameter string is greater than 8192.

Note: EXEC parameter string restrictions must also be considered.

PARAMETERSTRING
identifies the start of the Database Services Utility parameter string. The parameter string can contain Database Services Utility control parameters or an SQL statement, or both. Blanks or commas can be used to separate Database Services Utility control parameters from each other or from an SQL statement. The first entry in the parameter can be the command name ARIDBS.

The parameter string format is:

```
<optional control parameters> <optional SQL statement>
```

PARAMETERSTRINGEND+1
identifies the position following the Database Services Utility parameter string.

Format 2 - Program Interface Conventions: Your program must follow these register conventions:

Register 1	The address of the Database Services Utility parameter address list
Register 13	The address of the invoking program's 18 full-word save area
Register 14	The Database Services Utility return address
Register 15	The Database Services Utility entry point address on entry; the Database Services Utility processing return code on exit.

Your program Database Services Utility parameter address list must follow these conventions:

Position (per Byte)	Contents
1–4	Address of CALLTYPEID
5–8	Address of PARAMETERSTRING

where:

CALLTYPEID
is the address of an area defined as CHAR(6) that contains the Database Services Utility call type identifier. This area must contain the character string value DBSU01.

PARAMETERSTRING
identifies the start of the Database Services Utility parameter string. The parameter string can contain Database Services Utility control parameters or an SQL statement, or both. Blanks or commas can be used to separate Database Services Utility control parameters from each other or from an SQL statement. The first entry in the parameter can be the command name ARIDBS.

The parameter string format is:

LL	<optional control parameters> <optional SQL statement>

where:

LL is defined as a FIXED(15) value representing the length of the following parameter string. The maximum length of the parameter string passed to the Database Services Utility is 8192 not including the length field.

Register 1 Parameter List Considerations:

Database Services Utility processing determines if startup parameters are identified by register 1 by interrogating the contents of register 1 on entry to the Database Services Utility. Register 1 parameter processing is not performed if any of the following conditions occur:

- Register 1 contains the address of the ARIDBS module.
- Content of register 1 equals 0.
- A valid parameter list is passed by means of register 0.
- Content of register 1 addresses the character string value ARIDBS.
- First address in the Database Services Utility parameter address list equals 0.
- First address in the Database Services Utility parameter address list does not address the character string value DBSU01.
- Length of the parameter string is less than 1 or greater than 8192.

DB2 Server for VSE Program Interface Conventions: Your program must follow these register conventions:

Register 1	The address of the Database Services Utility parameter address list
Register 13	The address of the invoking program's 18 full-word save area
Register 14	The Database Services Utility return address
Register 15	The Database Services Utility entry point address on entry; the Database Services Utility processing return code on exit.

Your program Database Services Utility parameter address list must follow these conventions:

Position (per Byte)	**Contents**
1–4	Address of CALLTYPEID
5–8	Address of PARAMETERSTRING

where:

CALLTYPEID
is the address of an area defined as CHAR(6) that contains the Database Services Utility call type identifier. This area must contain the character string value DBSU01.

PARAMETERSTRING
identifies the start of the Database Services Utility parameter string. The parameter string can contain Database Services Utility control parameters or an SQL statement, or both. Blanks or commas can be used to separate Database Services Utility control parameters from each other or from an SQL statement.

The parameter string format is:

LL	<optional control parameters> <optional SQL statement>

where:

LL is defined as a FIXED(15) value representing the length of the following parameter string. The maximum length of the parameter string passed to the Database Services Utility is 8192 not including the length field but you must also consider VSE job control restrictions.

Register 1 Parameter List Considerations: Database Services Utility processing determines if startup parameters are identified by register 1 by interrogating the contents of register 1 on entry to the Database Services Utility. Register 1 parameter processing is not performed if any of the following conditions occur:

- Register 1 contains the address of the ARIDBS module.
- Content of register 1 equals 0.
- Content of register 1 addresses the character string value ARIDBS.
- First address in the Database Services Utility parameter address list equals 0.
- First address in the Database Services Utility parameter address list does not address the character string value DBSU01.
- Length of the parameter string is less than 1 or greater than 8192.

Sample Programs

Sample DB2 Server for VM EXEC Procedure: Invoke the Database Services Utility: The sample EXEC in Figure 63 invokes the Database Services Utility against the application server in multiple user mode. The example is written in the REXX language.

```
/* EXAMPLE DBS UTILITY REXX EXEC */
ARG PARMS
ADDRESS CMS
'FILEDEF ARISQLLD DISK ARISQLLD LOADLIB Q (NOCHANGE'
'FILEDEF SYSPRINT TERMINAL (NOCHANGE RECFM F LRECL 120'
'FILEDEF SYSIN TERMINAL (NOCHANGE'
'NUCXLOAD ARIDBS ARIDBS ARISQLLD'
'ARIDBS MESSAGES(SQLONLY) FORMAT(CL)' PARMS
ADDRESS
EXIT RC
```

Figure 63. Sample REXX EXEC to Invoke the Database Services Utility

The EXEC performs the following functions:

- Unconditionally invokes the utility with the control parameters MESSAGES(SQLONLY) and FORMAT(CL). These can be overridden by control parameters specified as EXEC command parameters.
  - If MESSAGES(NONE) is specified as an EXEC command parameter, it overrides the MESSAGES(SQLONLY) parameter.
  - If the FORMAT(LO) control parameter is specified as an EXEC command parameter, it overrides the FORMAT(CL) parameter.
- Accepts optional Database Services Utility control parameters as EXEC command parameters and passes them to the utility for processing. Database

Services Utility control parameters must be specified as EXEC command parameters before any SQL statement is issued.

- Accepts an optional SQL statement as the last EXEC command parameter string and passes it to the Database Services Utility for processing. Any Database Services Utility parameters must be specified as EXEC command parameters before the SQL statement.
  - If no SQL statement is specified as an EXEC command parameter, the Database Services Utility processing invoked by the example EXEC allows you to enter one or more SQL or Database Services Utility commands from the CMS command line. You are prompted to enter the first or next command.
- Displays the results of Database Services Utility processing to the terminal.

If you create the CMS file SQL EXEC *fm* (where *fm* is the file mode) containing the sample EXEC in Figure 63 on page 121, SQL statements can then be run from the CMS command line. Some examples of running one sample EXEC are in Figure 64 on page 122.

Note: DB2 Server for VM user machine must identify the database to be accessed by running the SQLINIT EXEC before running the sample EXEC or a similar EXEC. Also, the VM terminal logical line-editing symbols (character delete, line delete, line end, and escape) must not conflict with the SQL language operators used in the SQL statements entered in the CMS command line.

```
Enter--> sql select creator,tname from system.sysaccess
           (SELECT output will be displayed in column format)

Enter--> sql format(lo) select creator,tname from system.sysaccess
            (SELECT output will be displayed in list format)

Enter--> sql

     When prompted:

       Enter--> select creator,tname from system.sysaccess
                       (SELECT output will be displayed in column format)

     When prompted:

       Enter--> exit or another SQL statement
```

Figure 64. CMS Command Line Entries to Run a Sample EXEC

DB2 Server for VM Sample User Program: Link-Edit User Programs and the Database Services Utility: The following is a general example of the way to link-edit and run a user-written program (VMUCALL) that invokes the Database Services Utility. In Figure 65 on page 123 the example program is written in COBOL. The user program is run from a user library (USERLOAD LOADLIB A). This example assumes that the user program VMUCALL needs to be compiled first and that the VMUCALL TEXT A file does not exist.

A sample link-edit REXX EXEC is shown in Figure 66 on page 123. The contents of the user link-edit control file (VMULINK TEXT A) are shown in Figure 67 on page 123. Note that for the COBOL program, you must also link edit the TEXT file ARIPADR4. Figure 68 on page 123 shows a sample REXX EXEC for running the user program with multiple user mode.

```
***************************************************************
* Example COBOL Program Calling the Database Services Utility *
***************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. VMUCALL
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CALLTYPEID              PIC X(6) VALUE 'DBSU01'.
01  PARMSTRING.
    49 PARMLEN              PIC S9(4) COMP VALUE 42.
    49 PARMDATA             PIC X(42) VALUE
       'FORMAT(LO) SELECT * FROM SYSTEM.SYSOPTIONS'.
PROCEDURE DIVISION.
    CALL 'ARIDBS' USING CALLTYPEID PARMSTRING.
FINIS.
    STOP RUN.
```

Figure 65. Sample COBOL Program Calling the Database Services Utility

```
/* Example REXX EXEC to Link-Edit a COBOL program with DBS Utility */
ADDRESS 'COMMAND'
'COBOL2 VMUCALL (APOST'
'FILEDEF SYSLIB   DISK VSC2LTXT TXTLIB  Y'
'FILEDEF ARISQLLD DISK ARISQLLD LOADLIB Q (RECFM U'
'FILEDEF VMUCALL  DISK VMUCALL  TEXT A (RECFM F LRECL 80'
'FILEDEF ARIRVSTC DISK ARIRVSTC TEXT Q (RECFM F LRECL 80'
'FILEDEF ARIPADR4 DISK ARIPADR4 TEXT Q (RECFM F LRECL 80'
'FILEDEF SYSLMOD  DISK USERLOAD LOADLIB A (RECFM U'
'LKED VMULINK (LET RENT NAME VMUMOD LIST TERM PRINT'
EXIT
```

Figure 66. Sample EXEC to Link-Edit a User Program with the Database Services Utility

```
INCLUDE VMUCALL
INCLUDE ARIRVSTC
INCLUDE ARIPADR4          <---- For user COBOL program only
INCLUDE ARISSQLD(ARIDBS)
ENTRY VMUCALL
```

Note: Position 1 of each record must be blank.

Figure 67. Sample User Link-Edit Control File

```
/* Example REXX EXEC to Run a User Program Calling DBS Utility */
ADDRESS 'COMMAND'
'FILEDEF USERLOAD DISK USERLOAD LOADLIB A'
'FILEDEF SYSIN    TERMINAL (RECFM F LRECL 120'
'FILEDEF SYSPRINT TERMINAL (RECFM F LRECL 120'
'NUCXLOAD VMUMOD  VMUMOD USERLOAD'
'VMUMOD'
'NUCXDROP VMUMOD'
EXIT
```

Figure 68. Sample EXEC to Run a User Program Link-Edited with the Database Services Utility

DB2 Server for VSE Sample COBOL Program: Call the Database Services Utility: The following is a basic example of a user-written COBOL program and the job control statements that call Database Services Utility to run SQL statements:

```
// JOB CONTROL TO RUN A USER PROGRAM THAT CALLS THE DBS UTILITY
// OPTION CATAL
   PHASE COBDBSU,S
// EXEC IGYCRCTL
   CBL TRUNC(BIN)     APOST
     *********************************************
     * Example COBOL program calling Database Services Utility *
     *********************************************
     IDENTIFICATION DIVISION.
     PROGRAM-ID. COBDBSU
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.
     01  CALLTYPEID            PIC X(6) VALUE 'DBSU01'.
     01  PARMSTRING.
         02  PARMLEN           PIC S9(4) COMP.
         02  PARMDATA          PIC X(80).
     PROCEDURE DIVISION.

         MOVE 49 TO PARMLEN.
         MOVE 'ENDLUW(NO) CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;'
              TO PARMDATA.
         CALL 'ARIDBS' USING CALLTYPEID PARMSTRING.

         MOVE 32 TO PARMLEN.
         MOVE 'SELECT * FROM SYSTEM.SYSOPTIONS;'
              TO PARMDATA.
         CALL 'ARIDBS' USING CALLTYPEID PARMSTRING.
     FINIS.
         STOP RUN.
/*
    INCLUDE ARIPRDID
    INCLUDE ARIPADR4
    INCLUDE ARISYSDD
    INCLUDE ARIDSQLA
    INCLUDE ARIDDFP
    INCLUDE ARITDSSD
//  EXEC LNKEDT
//  EXEC PGM=COBDBSU
/*
/&
```

Figure 69. Sample COBOL Program to Call the Database Services Utility

Sample Assembler Program: Load and Invoke the Database Services Utility: The following are basic examples of user-written assembler language programs and the job control statements that invoke the Database Services Utility to run SQL statements. In DB2 Server for VSE these statements run with single user mode.

```
// JOB CONTROL FOR USER PROGRAM THAT CDLOADS AND RUNS THE DBS UTILITY
// OPTION CATAL
   PHASE VSELOAD,S
// EXEC ASSEMBLY
VSELOAD CSECT
R0       EQU    0
R1       EQU    1
R12      EQU    12
R13      EQU    13
R14      EQU    14
R15      EQU    15
         STM    R14,R12,12(R13)      STORE THE CALLER'S REGISTERS
         BALR   R12,0                ESTABLISH ADDRESSABILITY
         USING  *,R12                REGISTER 12 WILL BE BASE REGISTER
         LA     R15,SAVE             GET ADDRESS OF MY SAVE AREA
         ST     R15,8(R13)           STORE FORWARD SAVE AREA POINTER
         ST     R13,4(R15)           STORE BACKWARD SAVE AREA POINTER
         LR     R13,R15              MAKE MY SAVE AREA CURRENT
         CDLOAD ARIDBS               DBS ADDRESS RETURNED IN R1
         ST     R1,DBSADDR           SAVE DBS UTILITY ADDRESS
```

Figure 70. DB2 Server for VSE Sample Load-and-Invoke Assembler Program for the Database Services Utility (Part 1 of 2)

```
         MVI    PSTRINGT,C' '        CLEAR PARAMETER AREA
         MVC    PSTRINGT+1(L'PSTRINGT-1),PSTRINGT  CLEAR PARAMETER AREA
         MVC    PSTRINGT(L'PARMDAT1),PARMDAT1      MOVE FIRST COMMAND
         LA     R0,L'PSTRINGT        LOAD PARAMETER STRING LENGTH
         STH    R0,PSTRINGL          SET PARAMETER STRING LENGTH
         LA     R1,PARMLIST          LOAD PARAMETER LIST ADDRESS
         L      R15,DBSADDR          LOAD DBS ADDRESS
         BALR   R14,R15              CALL THE DBS UTILITY
         MVI    PSTRINGT,C' '        CLEAR PARAMETER AREA
         MVC    PSTRINGT+1(L'PSTRINGT-1),PSTRINGT  CLEAR PARAMETER AREA
         MVC    PSTRINGT(L'PARMDAT2),PARMDAT2      MOVE SECOND COMMAND
         LA     R1,PARMLIST          LOAD PARAMETER LIST ADDRESS
         L      R15,DBSADDR          LOAD DBS ADDRESS
         BALR   R14,R15              CALL THE DBS UTILITY
         L      R13,4(R13)           GET ADDRESS OF CALLER'S SAVE AREA
         L      R14,12(R13)          RESTORE CALLER'S R14
         LM     R0,R12,20(R13)       RESTORE CALLER'S R0-R12
         BR     R14                  RETURN. R15=DBS RETURN CODE
SAVE     DS     18F
DBSADDR  DS     F                 DBS ADDRESS SAVE AREA
PARMLIST DS     0F                PARAMETER LIST
         DC     A(CTYPEID)        **ADDRESS OF CALL TYPE IDENTIFIER
         DC     A(PSTRING)        **ADDRESS OF PARAMETER STRING
CTYPEID  DC     CL6'DBSU01'
PSTRING  DS     0H                PARAMETER LIST
PSTRINGL DS     H                 ** PARAMETER AREA LENGTH
PSTRINGT DS     CL250             ** PARAMETER AREA
PARMDAT1 DC     C'CONNECT SQLDBS IDENTIFIED BY SQLDBAPW'
PARMDAT2 DC     C'FORMAT(LO) SELECT * FROM SYSTEM.SYSOPTIONS'
         END    VSELOAD
/*
// EXEC LNKEDT
// EXEC ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,LOGMODE=N,PROGNAME=VSELOAD'
/*
/&
```

Figure 70. DB2 Server for VSE Sample Load-and-Invoke Assembler Program for the Database Services Utility (Part 2 of 2)

```
VMULOAD CSECT
R0       EQU    0
R1       EQU    1
R12      EQU    12
R13      EQU    13
R14      EQU    14
R15      EQU    15
         STM    R14,R12,12(R13)     STORE THE CALLER'S REGISTERS
         BALR   R12,0               ESTABLISH ADDRESSABILITY
         USING  *,R12               REGISTER 12 WILL BE BASE REGISTER
         LA     R15,SAVE            GET ADDRESS OF MY SAVE AREA
         ST     R15,8(R13)          STORE FORWARD SAVE AREA POINTER
         ST     R13,4(R15)          STORE BACKWARD SAVE AREA POINTER
         LR     R13,R15             MAKE MY SAVE AREA CURRENT
         LOAD   EP=ARIDBS           DBS ADDRESS RETURNED IN R0
         ST     R0,DBSADDR          SAVE DBS ADDRESS
         MVI    PSTRINGT,C' '       CLEAR PARAMETER AREA
         MVC    PSTRINGT+1(L'PSTRINGT-1),PSTRINGT  CLEAR PARAMETER AREA
         MVC    PSTRINGT(L'PARMDATA),PARMDATA      SET PARAMETER
         LA     R0,L'PSTRINGT       LOAD PARAMETER STRING LENGTH
         STH    R0,PSTRINGL         SET PARAMETER STRING LENGTH
         SR     R0,R0               CLEAR REGISTER 0 (R1=PARM ADDRESS)
         LA     R1,PARMLIST         LOAD PARAMETER LIST ADDRESS
         L      R15,DBSADDR         LOAD DBS ADDRESS
         BALR   R14,R15             CALL THE DBS UTILITY
         L      R13,4(R13)          GET ADDRESS OF CALLER'S SAVE AREA
         L      R14,12(R13)         RESTORE CALLER'S R14
         LM     R0,R12,20(R13)      RESTORE CALLER'S R0-R12
         BR     R14                 RETURN. R15=DBS RETURN CODE
SAVE     DS     18F
DBSADDR  DS     F                DBS ADDRESS SAVE AREA
PARMLIST DS     0F               PARAMETER LIST
         DC     A(CTYPEID)       **ADDRESS OF CALL TYPE IDENTIFIER
         DC     A(PSTRING)       **ADDRESS OF PARAMETER STRING
CTYPEID  DC     CL6'DBSU01'
PSTRING  DS     0H               PARAMETER LIST
PSTRINGL DS     H                ** PARAMETER AREA LENGTH
PSTRINGT DS     CL250            ** PARAMETER AREA
PARMDATA DC     C'FORMAT(LO) SELECT * FROM SYSTEM.SYSOPTIONS'
         END    VMULOAD
```

Figure 71. DB2 Server for VM Sample Load-and-Invoke Assembler Program for the Database Services Utility

Assuming that the DB2 Server for VM sample program is contained in the CMS file VMULOAD ASSEMBLE A, you can run the program with multiple user mode by entering the commands in Figure 72:

```
/* Example Database Services Utility to Run an ASSEMBLER Program Calling */
/* DBS Utility                                                           */
ADDRESS 'COMMAND'
'GLOBAL MACLIB OSMACRO'
'GLOBAL LOADLIB ARISQLLD'     <---- Identifies DB2 Server for VM load library
'ASSEMBLE VMULOAD'
'LOAD VMULOAD'
'FILEDEF SYSPRINT TERMINAL (RECFM F LRECL 120'
'START'
EXIT
```

Figure 72. REXX EXEC to Run an Assembler Program That Loads the DBS Utility

Note: Before attempting to run the user program, identify the application server to be accessed by using the SQLINIT EXEC.

Sample Assembler Program: Call the Database Services Utility: The following are examples of user-written assembler language programs that invokes the Database Services Utility. In DB2 Server for VSE, the Utility is invoked by means of a CALL macro to process commands in a file defined as SYSIPT. The VSE job control statements to assemble, link-edit, and run the user program with single user mode are shown in the example. In DB2 Server for VM, the Utility in invoked, by means of a CALL macro to process commands in a file defined as SYSIN. All necessary CMS FILEDEF commands must be entered before this program is executed.

```
// JOB CONTROL TO RUN A USER PROGRAM THAT CALLS THE DBS UTILITY
// OPTION CATAL
   PHASE VSECALL,S
// EXEC ASSEMBLY
VSECALL  CSECT
R0       EQU    0
R1       EQU    1
R12      EQU    12
R13      EQU    13
R14      EQU    14
R15      EQU    15
         STM    R14,R12,12(R13)     STORE THE CALLER'S REGISTERS
         BALR   R12,0               ESTABLISH ADDRESSABILITY
         USING  *,R12               REGISTER 12 WILL BE BASE REGISTER
         LA     R15,SAVE            GET ADDRESS OF MY SAVE AREA
         ST     R15,8(R13)          STORE FORWARD SAVE AREA POINTER
         ST     R13,4(R15)          STORE BACKWARD SAVE AREA POINTER
         LR     R13,R15             MAKE MY SAVE AREA CURRENT
         LA     R0,0                CLEAR REGISTER 0
         LA     R1,0                INDICATE NO PARAMETER LIST PASSED
         CALL   ARIDBS
         L      R13,4(R13)          GET ADDRESS OF CALLER'S SAVE AREA
         L      R14,12(R13)         RESTORE CALLER'S R14
         LM     R0,R12,20(R13)      RESTORE CALLER'S R0-R12
         BR     R14                 RETURN. R15=DBS RETURN CODE
SAVE     DS     18F
         END    VSECALL
/*
   INCLUDE ARISYSDD
   INCLUDE ARIDSQLA
   INCLUDE ARIDDFP
   INCLUDE ARIPRDID
   INCLUDE ARITDSSD
// EXEC LNKEDT
// EXEC ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,LOGMODE=N,PROGNAME=VSECALL'
/*
/&
```

Figure 73. DB2 Server for VSE Sample Assembler Program to Call the Database Services Utility

```
VMUCALL  CSECT
R0       EQU    0
R1       EQU    1
R12      EQU    12
R13      EQU    13
R14      EQU    14
R15      EQU    15
         STM    R14,R12,12(R13)     STORE THE CALLER'S REGISTERS
         BALR   R12,0               ESTABLISH ADDRESSABILITY
         USING  *,R12               REGISTER 12 WILL BE BASE REGISTER
         LA     R15,SAVE            GET ADDRESS OF MY SAVE AREA
         ST     R15,8(R13)          STORE FORWARD SAVE AREA POINTER
         ST     R13,4(R15)          STORE BACKWARD SAVE AREA POINTER
         LR     R13,R15             MAKE MY SAVE AREA CURRENT
         LA     R0,0                CLEAR REGISTER 0
         LA     R1,0                INDICATE NO PARAMETER LIST PASSED
         CALL   ARIDBS
         L      R13,4(R13)          GET ADDRESS OF CALLER'S SAVE AREA
         L      R14,12(R13)         RESTORE CALLER'S R14
         LM     R0,R12,20(R13)      RESTORE CALLER'S R0-R12
         BR     R14                 RETURN. R15=DBS RETURN CODE
SAVE     DS     18F
         END    VMUCALL
```

Figure 74. DB2 Server for VM Sample Assembler Program to Call the Database Services Utility

Running the DB2 Server for VM Database Services Utility with Multiple User Mode

With multiple user mode, the Database Services Utility runs as an application program in the user's virtual machine. It cannot run either in the CMS/DOS environment or in CMS subset with multiple user mode.

Use the SQLINIT EXEC to specify the default database that you want to access. For additional information on the SQLINIT EXEC, refer to the *DB2 Server for VSE & VM Database Administration* manual.

Running the Database Services Utility with Single User Mode

When the Database Services Utility is run with single user mode, the database manager is started by means of the SQLSTART EXEC, which is invoked by the SQLDBSU EXEC. The database manager then loads the Database Services Utility program and transfers control to the program ARIDBS. All startup and initialization parameters cannot be directly supplied by means of the SQLDBSU EXEC. The SQLDBSU EXEC parameters that can be supplied, however, and that are applicable only to running the utility with single user mode are: DBNAME, DCSSID, LOGMODE, and PARMID. These parameters are supplied to the SQLSTART EXEC by the SQLDBSU EXEC when the database manager is started. The SQLDBSU EXEC also supplies the initialization parameter, PROGNAME(ARIDBS), to direct the database manager to execute the Database Services Utility program ARIDBS. The IBM-supplied SQLSTART EXEC and the startup and initialization parameters are described in the *DB2 Server for VM System Administration* manual.

Single or Multiple User Mode: The sample commands shown below run the Database Services Utility with single user mode and with multiple user mode to perform the following functions:

- Unload all tables in the SQLDBA database dbspace PUBLIC.SAMPLE to a DASD file

- Reload the dbspace PUBLIC.SAMPLE from the DASD file to reorganize the data for all tables in the dbspace.

The CMS file DBSU COMMANDS A contains the following Database Services Utility command input:

```
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
UNLOAD DBSPACE(PUBLIC.SAMPLE) OUTFILE(DASD1)
RELOAD DBSPACE(PUBLIC.SAMPLE) PURGE INFILE(DASD1)
```

To run the utility with single user mode, enter the following commands:

```
FILEDEF DASD1 DISK DBSUFILE DATA A4 (RECFM VBS BLOCK 2048
SQLDBSU DBNAME(SQLDBA) SYSIN(DBSU COMMANDS A) SYSPRINT(TERMINAL)
```

To run the utility with multiple user mode, enter the following commands:

```
SQLINIT DBNAME(SQLDBA)
FILEDEF DASD1 DISK DBSUFILE DATA A4 (RECFM VBS BLOCK 2048
SQLDBSU SYSIN(DBSU COMMANDS A) SYSPRINT(TERMINAL)
```

Using the SQLDBSU EXEC

SQLDBSU EXEC Format: The following syntax diagrams show the format for invoking the Database Services Utility with single or multiple user mode.

Format:

Multiple User Mode Parameters

```
►►──SQLDBSU──┬────────────────────────────────────────────────────┬──────────────►
             └─sysIN──(──┬─Reader─────────────────────────────┬─)─┘
                         ├─Terminal───────────────────────────┤
                         └─file_name─┬────────────────────────┤
                                     └─file_type─┬────────────┤
                                                 └─file_mode──┘

►──┬──────────────────────────────────────────────────────┬───────────────────►◄
   └─sysPRint──(──┬─Printer──────────────────────────────┬─)─┘
                  ├─Terminal─────────────────────────────┤
                  └─file_name─┬──────────────────────────┤
                              └─file_type─┬──────────────┤
                                          └─file_mode────┘
```

Multiple User Mode Examples:

SQLDBSU SYSIN(T) SYSPRINT(PR)
SQLDBSU

Notes:

1. When running the Database Services Utility in multiple user mode, you must issue the SQLINIT EXEC before using the SQLDBSU EXEC. The SQLINIT EXEC initializes the LASTING GLOBALV file and the database manager

bootstrap routines on your A-disk. These routines identify the application server that you want to access and the method for loading the multiple user mode support system routines.

The SQLINIT EXEC does not need to be run prior to each SQLDBSU command. You need only to run the SQLINIT EXEC when you want to establish access to a different application server, to vary the method of loading the multiple user mode support system routines, or to change other characteristics of the application requester.

An alternative method for connecting to another application server is to issue the CONNECT command. See “CONNECT” on page 20, for more information about using this command.

2. When the Database Services Utility (program ARIDBS) is run with multiple user mode, it is loaded from load library ARISQLLD LOADLIB Q by using a CMS NUCXLOAD command. Consequently, the SQLDBSU EXEC cannot be run in the CMS/DOS environment with multiple user mode.

Format:

Single User Mode Parameters

```
►►──SQLDBSU──┬─────────────────────────────────────────────────────┬───────────────────────────────►
             └─sysIN──(──┬─Reader──────────────────────────────┬─)─┘
                         ├─Terminal────────────────────────────┤
                         └─file_name──┬──────────────────────┬─┘
                                      └─file_type──┬───────────┬─┘
                                                   └─file_mode─┘

►──┬──────────────────────────────────────────────────────────┬──────────────────────────────────►◄
   └─sysPRint──(──┬─Printer─────────────────────────────┬─)─┘
                  ├─Terminal────────────────────────────┤
                  └─file_name──┬──────────────────────┬─┘
                               └─file_type──┬───────────┬─┘
                                            └─file_mode─┘

►►──Dbname──(server_name)──┬──────────────────┬──┬─────────────────────┬──┬─────────────────────┬──►◄
                           └─dcssID──(dcss_id)─┘  └─LOGmode──(──┬─A─┬─)─┘  └─PARMID──(file_name)─┘
                                                               ├─L─┤
                                                               ├─N─┤
                                                               └─Y─┘
```

Single User Mode Example: SQLDBSU IN(T) PR(DBSU LIST A) D(SQLDBA) ID(SUF1) LOG(Y) PARMID(SQL1)

Notes:

1. In single user mode, the console and program stack buffers are purged by the SQLSTART EXEC. If you use multiple-volume tape files for Database Services Utility processing with multiple user mode, the console and program stack buffers should not contain any information. Empty buffers ensure that any prompts issued by the multiple tape volume tape support can be properly processed.

The following is a description of the parameters for the Database Services Utility SQLDBSU EXEC.

sysIN (*file_name file_type file_mode***)**
identifies the file name and optionally the file type and file mode of the CMS file containing the Database Services Utility input commands. The file type defaults to DBSINPUT and the file mode defaults to A.

If you supply this form of the SYSIN parameter, the SQLDBSU EXEC issues the following CMS FILEDEF command for the Database Services Utility control file:

```
FILEDEF SYSIN DISK file_name file_type file_mode . . .
        (RECFM FB LRECL 80 BLOCK 800
```

sysIN (Reader)
specifies that the Database Services Utility input control file is a virtual reader file. If you specify SYSIN(Reader), the SQLDBSU EXEC issues the following CMS FILEDEF command for the Database Services Utility control file:

```
FILEDEF SYSIN READER (RECFM F LRECL 80
```

sysIN (Terminal)
specifies that the Database Services Utility input control file is the terminal. If you specify SYSIN(Terminal), the SQLDBSU EXEC issues the following CMS FILEDEF command for the Database Services Utility control file:

```
FILEDEF SYSIN TERMINAL (RECFM F LRECL 80
```

Notes:

1. The Database Services Utility control file is also assigned to the terminal if you did not specify the SYSIN parameter, and you did not issue a CMS FILEDEF command for the *ddname*=SYSIN before issuing the SQLDBSU command.
2. If the Database Services Utility control file is assigned to the terminal:
   a. Most Database Services Utility commands and all SQL statements must be terminated by a semicolon. Database Services Utility commands restricted to a single control file record (command line) do not require a terminating semicolon. **As a general rule, use a semicolon to terminate all commands entered through the terminal.**
   b. The end of Database Services Utility input is indicated by entering a null line. Prompts are issued to you after you use the ENTER key to enter a null line. Your response depends on whether or not a command has been partially entered and on the type of command entered. This way, you cannot end Database Services Utility processing by mistakenly using the ENTER key. If prompted to quit and you want to do so, enter QUIT or HX.
   c. Positions 1–80 of the input record are checked for command information. Therefore, command records contained in files identified by READ FILE commands entered through the terminal cannot contain sequence numbers in positions 73–80. The READ FILE command has the format:

      ```
      READ FILE file_name file_type file_mode
      ```

      If you use the READ FILE command, it must be the first command after you issued the EXEC SQLDBSU command. It must not be preceded by any other commands.
   d. If Database Services Utility input entered from the terminal must contain lowercase values, you must issue the following FILEDEF before issuing the SQLDBSU EXEC without the SYSIN parameter specification:

      ```
      FILEDEF SYSIN TERMINAL (RECFM F LRECL 80 LOWCASE
      ```

      If you issue this FILEDEF command, all the Database Services Utility command and SQL statement keywords must be entered in uppercase.

sysPRint (*file_name file_type file_mode*)
identifies the file name and optionally the file type and file mode of the CMS

file to be used for the Database Services Utility messages. The file type specification defaults to DBSLIST and the file mode specification defaults to A.

If you specify this form of the SYSPRINT parameter, the SQLDBSU EXEC issues the following CMS FILEDEF command for the Database Services Utility message file:

```
FILEDEF SYSPRINT DISK file-name file-type file-mode . . .
            (RECFM FBA LRECL 121 BLOCK 1210
```

Note: Position 1 of the Database Services Utility message file (*ddname*=SYSPRINT) print records contains American Standards Association (ASA) control characters. If the message file is a CMS file and is printed with the CMS PRINT command, the option CC should be specified in the CMS PRINT command. Refer to the *VM/ESA: CMS Command Reference* manual for a description of the CMS PRINT command.

sysPRint (Printer)
specifies that the Database Services Utility message file should be assigned to the virtual printer.

If you specify SYSPRINT (Printer), the SQLDBSU EXEC issues the following CMS FILEDEF command for the Database Services Utility message file:

```
FILEDEF SYSPRINT PRINTER (RECFM FA LRECL 121
```

sysPRint (Terminal)
specifies that the Database Services Utility message file should be assigned to the terminal.

If your specify SYSPRINT (Terminal), the SQLDBSU EXEC issues the following CMS FILEDEF command for the Database Services Utility message file:

```
FILEDEF SYSPRINT TERMINAL (RECFM F LRECL 120
```

Note: The Database Services Utility message file is also assigned to the terminal if the SYSPRINT parameter is not specified, and you did not issue a CMS FILEDEF command for the *ddname*=SYSPRINT before issuing the SQLDBSU EXEC.

Dbname(*server_name***)**
indicates that the Database Services Utility should be run with single user mode. It also identifies the name of the database to be accessed by the commands contained in the Database Services Utility control file.

If you specify this parameter, the SQLSTART EXEC parameters DBNAME(*server_name*) are specified. The initialization parameters SYSMODE=S, and PROGNAME=ARIDBS are also supplied in the SQLSTART EXEC PARM parameter.

If you omit the DBNAME parameter, the Database Services Utility is run with multiple user mode.

dcssID (*dcss_id***)**
applies to running the Database Services Utility with single user mode. It identifies the method by which all database manager modules are to be loaded for execution. If you specify this parameter, you must also specify the DBNAME parameter.

If you specify this parameter, it is supplied to the SQLSTART EXEC. The SQLSTART EXEC then uses the database manager bootstrap routines with the specified *dcss-id* to load the database manager code.

If you omit this parameter, it is not supplied as an SQLSTART EXEC parameter.

LOGmode (A|L|N|Y)
applies only to running the Database Services Utility with single user mode. This one-character field indicates how the logs are to be maintained:

A means maintain the logs and automatically archive the database.

L means maintain the logs and automatically archive the log when the specified level has been reached.

N means do not maintain the logs for recovery.

Y means maintain the logs.

If you specify this parameter, you must also specify the DBNAME parameter. It identifies the value to be used for the database initialization LOGMODE parameter when the database manager is started in single user mode.

If you omit this parameter and you specify the DBNAME parameter, the LOGMODE parameter is not supplied as an initialization parameter in the SQLSTART EXEC.

PARMID (*file_name*)
applies only to running the Database Services Utility with single user mode. If you specify this parameter, you must also specify the DBNAME parameter. It identifies the file name of the CMS file that contains the database initialization override parameters. The file type must be SQLPARM.

If you omit this parameter and you specify the DBNAME parameter, the PARMID parameter is not supplied as an initialization parameter in the SQLSTART EXEC.

For more information on the PARMID parameter, see the description of SQLSTART in the *DB2 Server for VM System Administration* manual.

Notes:

1. If virtual console spooling is started and either SYSIN or SYSPRINT is assigned to the terminal, the virtual console is spooled HOLD.
2. If the virtual printer is spooled NOHOLD, it is spooled HOLD, closed, and then spooled NOHOLD.
3. When running the Database Services Utility, set the CP SET command EMSG option to ON. The SQLDBSU EXEC checks the current EMSG option setting and, if necessary, resets it to ON. If the EMSG option is reset, the original setting is restored during the SQLDBSU EXEC termination processing. The original EMSG setting is not restored if SQLDBSU EXEC execution is abnormally terminated. Refer to the *VM/ESA: CP Command and Utility Reference* for a description of the SET command's EMSG option.
4. When running the Database Services Utility, the VM terminal logical line-editing symbols (character delete, line delete, line end, and escape) must be defined as follows:
   ```
   LINEND #   LINEDEL OFF   CHARDEL OFF   ESCAPE 1/2
   ```
   The SQLDBSU EXEC checks the current settings for these symbols and, if necessary, resets the symbols as described previously. If the symbols are reset, the original symbols are restored during the SQLDBSU EXEC termination processing. The symbols are not reset if SQLDBSU EXEC execution is

abnormally terminated. Refer to the *VM/ESA: CP Command and Utility Reference* for a description of the TERMINAL command and the logical line-editing symbols.

5. The Database Services Utility control file (*ddname*=SYSIN) logical record length (LRECL) must be 80. The record format (RECFM) should be fixed or fixed blocked.
6. The Database Services Utility message file (*ddname*=SYSPRINT) logical record length (LRECL) must be at least 81. The record format (RECFM) should be fixed or fixed blocked with ASA print control characters (F or FB).

   If the Database Services Utility control parameter PAGECTL(NO) is specified, the minimum message file record length is 80. The record format should be fixed or fixed blocked (F or FB).
7. Database Services Utility and SQL statement information must be in positions 1-72 of the control file (*ddname*=SYSIN) records except when SYSIN defines the input control file as the terminal. If the terminal is the control file, you can put Database Services Utility and SQL statement information in positions 1-80 of a command line that you enter from the terminal. If you enter a DB2 Server for VM READ FILE *file_name file_type file_mode* command from the terminal, the CMS file containing SQL statements or Database Services Utility commands cannot have sequence numbers in positions 73-80 because the Database Services Utility searches positions 1-80 for command information. For more information on the READ FILE command, see the *DB2 Server for VM Program Directory*.

   You can always reference all 80 positions of data records contained in the Database Services Utility control file as data field positions.
8. You must define all input or output data file *ddnames* referenced in the Database Services Utility commands supplied in the Database Services Utility control file through the CMS FILEDEF commands before issuing the SQLDBSU command.

   If you define DASD CMS files with variable-length spanned records for Database Services Utility command input/output, you must use the file mode number **4**.
9. If you do not specify the SYSIN information in the EXEC parameters, and you do not define the Database Services Utility control file (*ddname*=SYSIN) with the CMS FILEDEF command before issuing the SQLDBSU command, the SQLDBSU EXEC issues the following default FILEDEF command:

   ```
   FILEDEF SYSIN TERMINAL (RECFM F LRECL 80
   ```

   If you do not specify the SYSIN information in the EXEC parameters, but you define the Database Services Utility control file (*ddname*=SYSIN) with a CMS FILEDEF command before issuing the SQLDBSU command, the SQLDBSU EXEC uses the user-defined Database Services Utility control file. Specify a logical record length (LRECL) of 80 for a user-defined Database Services Utility control file.
10. If you do not specify SYSPRINT information, and you do not define the Database Services Utility message file (*ddname*=SYSPRINT) with a CMS FILEDEF command before issuing the SQLDBSU command, the SQLDBSU EXEC issues this FILEDEF command:

    ```
    FILEDEF SYSPRINT TERMINAL (RECFM F LRECL 120
    ```

    If you define the Database Services Utility message file with a CMS FILEDEF command, the minimum logical record length (LRECL) is 81. If you specify the Database Services Utility control parameter as PAGECTL(NO), the minimum message file record length is 80.

Chapter 8. Command Reference

The Database Services Utility can process commands that are unique to the Database Services Utility and SQL statements that are not restricted to use in user-written programs. This chapter provides descriptions of the Database Services Utility commands and general rules governing how you type the commands. (SQL statements are described in the *DB2 Server for VSE & VM SQL Reference*.)

Command Processing

Two kinds of commands that you can specify in the Database Services Utility are Database Services Utility commands and SQL statements.

The difference between an SQL statement and a Database Services Utility command is that a Database Services Utility command can only be issued within the Database Services Utility. If you try to issue a Database Services Utility command outside of the utility itself, it fails. On the other hand, an SQL statement can be issued in both ISQL and the Database Services Utility.

Here is a summary of the Database Services Utility commands:

DATALOAD
DATAUNLOAD
RELOAD
UNLOAD
SET
COMMENT
REORGANIZE INDEX
SCHEMA
REBIND PACKAGE

Control commands are entered by means of one or more 80-byte input records in the (input) control file. The utility usually reads only the first 72 positions of these command records; you can use positions 73 through 80 for sequence numbers. When control command input is being read directly from a DB2 Server for VM terminal, all 80 positions of the command record can contain command information. Some Database Services Utility commands allow you to place data within the (input) control file; these data records are not restricted to the first 72 positions and can have information in all 80 positions.

In DB2 Server for VSE, lowercase information supplied in an input control card file and read by the Database Services Utility is not converted to uppercase by Database Services Utility processing. ISQL, on the other hand, converts lowercase information to uppercase. Use uppercase in the input control card file to avoid a case mismatch, especially for a table or dbspace name. In DB2 Server for VM, lowercase information supplied in commands read by the utility is converted to uppercase only when the control file (SYSIN) is assigned to the terminal. If you require lowercase information and the utility reads commands from the terminal, specify LOWCASE when you issue a CMS FILEDEF command to define the control file (SYSIN). Alternatively, you can use the CMS SET INPUT xx yy command to reset the hexadecimal code xx to the hexadecimal code yy. Refer to the *VM/ESA: CMS Command Reference* manual for more information.

Except where noted, the control commands can span multiple 80-byte input records. Individual keywords or parameter values must never span input records, or a Database Services Utility processing error results. For example, these records are correct:

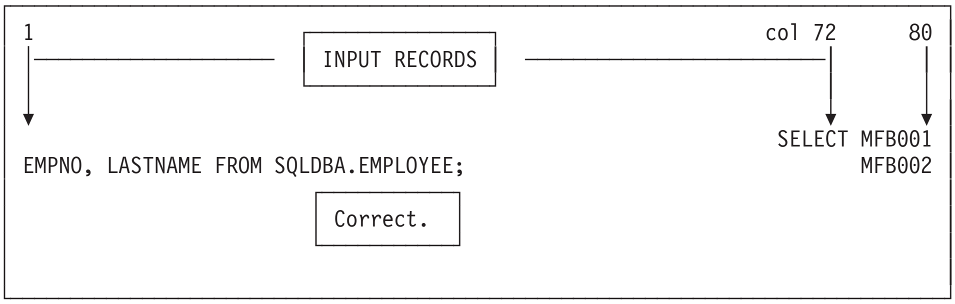


Figure 75. Example of Correct Records

MFB001 and MFB002 are sequence numbers that the Database Services Utility ignores. Note that SELECT ends in position 72 in the above example. Conceptually, the Database Services Utility inserts a single blank character between input records; the above records are interpreted as:

```
SELECT EMPNO, LASTNAME FROM SQLDBA.EMPLOYEE;
```

The following input records are incorrect:

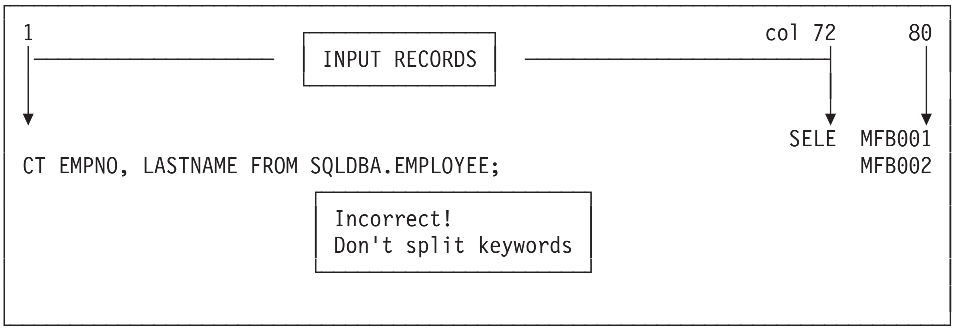


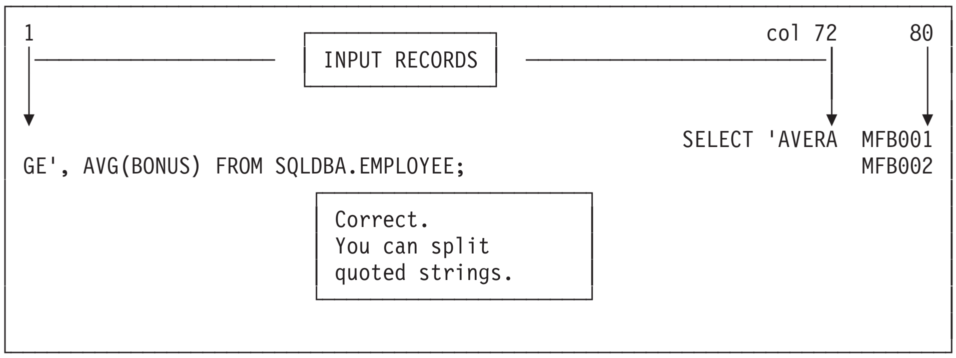
Figure 76. Example of Incorrect Records

The Database Services Utility inserts a blank after column 72, and thus interprets the input records as:

```
SELE CT EMPNO, LASTNAME FROM SQLDBA.EMPLOYEE;
```

The Database Services Utility does not recognize SELE as a Database Services Utility command, and an error results.

There is an exception to the rule that individual keywords or parameter values must not span input records. This exception occurs when a parameter is enclosed in either single (') or double (") quotation marks. For these parameters, the Database Services Utility does not insert a blank after position 72. Consider the following example in which a character string constant spans multiple input records. The character constant is delimited by a single quotation mark ('):



```
                                               SELECT 'AVERA  MFB001
GE', AVG(BONUS) FROM SQLDBA.EMPLOYEE;                         MFB002
```

Figure 77. Example of a Character String Constant Spanning Multiple Input Records

The Database Services Utility interprets the above records as:

```
SELECT 'AVERAGE', AVG(BONUS) FROM SQLDBA.EMPLOYEE;
```

Each Database Services Utility command or SQL statement must begin on a new (input) control file input record.

DB2 Server for VM

Database Services Utility commands are terminated by a semicolon, by the start of the next command, or by the end of the input control records. Terminate Database Services Utility commands with a semicolon when Database Services Utility control command input is being read directly from a terminal.

In the Database Services Utility environment, SQL statements **must** be terminated with a semicolon. Do not use the SQL continuation character (required by ISQL) in an SQL statement that spans record boundaries in either a batch or an interactive environment.

Database Services Utility processing ends when all the input records are processed.

You cannot reorganize a catalog index by using the REORGANIZE INDEX command. To reorganize the catalog index in VM, use the SQLCIREO utility. In VSE, set the STARTUP initialization parameter to one to reorganize the catalog index.

Note: You can only reorganize a primary key index or a unique index by using the ALTER TABLE ACTIVATE PRIMARY KEY and ALTER TABLE ACTIVATE UNIQUE statements.

COMMENT

With the COMMENT command, you can document input by supplying Database Services Utility COMMENT commands at appropriate points within the Database Services Utility control command input stream. The utility displays the comments in the report or message file listing. You cannot use SQL comments within Database Services Utility COMMENT commands.

COMMENT Format

Format:

```
►►—COMMENT—'string_constant' ————————————————————————————————————————————►◄
```

COMMENT
identifies a Database Services Utility COMMENT command. At least one blank must appear after the command identifier.

'*string-constant***'**
is the comment text delimited by single quotation marks. The *string-constant* can span control command input records, but must begin in the same record that contains the command identifier. All positions of a control command record containing comment text are displayed. A COMMENT command is terminated when a control command input record containing comment text ends with a single quotation mark or a single quotation mark immediately followed by a semicolon. All control command input record positions (normally positions 1-72) after the terminating single quotation mark or single quotation mark and semicolon must be blank. Positions 73-80 of comment text records can still contain sequence numbers (except in DB2 Server for VM when Database Services Utility control command input is being read directly from a terminal).

REORGANIZE INDEX

The REORGANIZE INDEX command allows you to correct index fragmentation, and correct the skewing of index key values without having to drop the index and then recreate it using the DROP INDEX and CREATE INDEX SQL statements. REORGANIZE INDEX also revalidates an invalid index.

As with other Database Services Utility commands, you can use the REORGANIZE INDEX command with both single user mode and multiple user mode.

Note: The REORGANIZE INDEX command is not supported if you are using DRDA flow.

REORGANIZE INDEX Format

Format:

```
►►—REORGANIZE INDEX—(index_name)—┬———————————————————┬——————————————————►◄
                                 └─PCTFREE =—integer─┘
```

Example:

REORGANIZE INDEX(SMITH.INDEXINV) PCTFREE = 50

Authorization: You must own the index or have DBA authority.
Note: In DB2 Server for VSE, to reorganize a catalog index, you must start the database manager in single user mode and specify STARTUP=I.

INDEX (*index-name***)**
identifies the index to be reorganized. You can further identify the index by specifying the *owner* and *server-name* of the index. For more information about identifying the index, see "Qualifying Object Names" on page 110 for details.

When reorganizing an invalid index, the database manager uses a sort similar to the one used for a CREATE INDEX statement.

PCTFREE = *integer*
: allows you to control the amount of free space that REORGANIZE reserves in the index for later insertions and updates.

integer
: is a number from `0` to 99 representing a percentage of the total index space. For practical purposes, it should not exceed 50.

If you do not specify PCTFREE, the amount of free space remains unchanged from the previous PCTFREE value.

The REORGANIZE INDEX command consists of several separate internal steps. Some steps might be completed even though the whole REORGANIZE INDEX command is not completed successfully.

The following unusual situations can occur when rolling back or recovering from a REORGANIZE INDEX command:

- Although a REORGANIZE INDEX command is successfully completed during forward processing, it might not be completed during rollback or recovery because the system has run out of physical or logical pages (or both). Checkpoints for sufficient storage can interrupt only during forward processing. As a result, the recovered index is marked invalid.
- If an index reorganization is rolled back or undone, the recovered index is nevertheless reorganized. In REORGANIZE INDEX processing, the index is effectively dropped and re-created, resulting in a reorganized index. If you changed the PCTFREE value, however, that value is set back to the original value defined before the REORGANIZE INDEX command.
- If the system ends abnormally after an index reorganization is interrupted by a checkpoint and an attempt is made to restart the database manager without the current log (because of a loss of the log or log reconfiguration), the reorganized index will be marked as invalid. It must be recovered by dropping the dbspace and re-creating it, or by restoring a previous database archive.
- The updating of index statistics is not supported during rollback or recovery. The previous index statistics are recovered, but the recovered index (which is nevertheless reorganized) may not match them.

DBSS prevents a REORGANIZE INDEX or CREATE INDEX command from proceeding if the command can overflow the invalid index limit. During rollback or recovery, if the maximum number of invalid indexes (30) is reached, the system ends abnormally. When the system is brought up again for recovery, it is very likely to end again for the same reason. If the large number of invalid indexes exists because of a lack of physical pages, add a dbextent to the system before attempting to recover again. If the large number of invalid indexes exists because of a lack of logical index pages, use filtered log recovery to skip over the index reorganizations that are causing the invalid indexes. For information about adding a dbextent, see the *DB2 Server for VM System Administration* and *DB2 Server for VSE System Administration* manuals. To find out more about using filtered log recovery, see the *DB2 Server for VSE & VM Diagnosis Guide and Reference* manual. Additional information on the REORGANIZE INDEX command is in the *DB2 Server for VSE & VM Database Administration* manual.

You cannot reorganize a catalog index by using the REORGANIZE INDEX command. To reorganize the catalog index in VM, use the SQLCIREO utility. In VSE, set the STARTUP initialization parameter to I to reorganize the catalog index.

Note: You can only reorganize a primary key index or a unique index by using the ALTER TABLE ACTIVATE PRIMARY KEY and ALTER TABLE ACTIVATE UNIQUE statements.

SCHEMA

A schema file specifies an authorization ID and a list of table, view, and privilege definitions using the syntax of the CREATE TABLE, CREATE VIEW, and GRANT statements. The SCHEMA command reads and processes the statements from a schema file.

SCHEMA Format

Table 7. SCHEMA Command Syntax

Format:

```
    (1)
►►──────SCHEMA INFILE──(──ddname──┤ option-c ├──)──┬──────────────────────┬────────────►◄
                                                   └─IN──(dbspace_name)──┘
```

Notes:

1 Option C is valid in DB2 Server for VSE only.

option-c:

```
├──┬────────────────────────────┬──┬──────────────────────────────────────┬──────────────┤
   │              ┌─2000──┐     │  │                     ┌─REWIND───┐     │
   └─BLKSZ──(──┴─size──┴──)──┘  │          ┌─(TAPE)──┴─NOREWIND─┴─┐  │
                                   └─PDEV──┴─(DASD)────────────────┴──┘
```

Example:

SCHEMA INFILE(IN1 BLKSZ(800))

Authorization:

You must be connected as the AUTHORIZATION ID specified in the CREATE SCHEMA statement.

INFILE (*ddname*)
identifies the sequential input file containing the schema.

ddname
in DB2 Server for VSE: this is the TLBL or DLBL job control statement file name for the sequential input file. The file must have a record format of fixed-length blocked and a record length of 80.

Alternatively, SCHEMA can read its input from SYSIPT by using a READ MEMBER. You use the READ MEMBER NOCONT option to properly close the SYSIPT file. An example of using READ MEMBER with NOCONT is:

```
SCHEMA INFILE(SYSIPT)
READ MEMBER schema-member (NOCONT
```

If you do not specify the NOCONT option, the database manager reads in the SYSIPT records following the READ MEMBER statement as part of the SCHEMA file; then SYSIPT can provide additional input after the READ MEMBER statement.

in DB2 Server for VM: this is the name of the sequential input file. It must have records with a fixed length of 80 characters. The file characteristics specified in the FILEDEF command or the default FILEDEF options are the source of the input record definition information for the Database Services Utility. Do not specify SYSIN or SYSPRINT as the *ddname*.

dbspace-name
specifies the name of the dbspace where the table is to be placed if no dbspace-name is given in the CREATE TABLE statement. If the CREATE TABLE statement in the SCHEMA input file specifies a dbspace-name then this overrides the name of the dbspace given in the SCHEMA command.

BLKSZ (*size*) (DB2 Server for VSE Only)
is a parameter that specifies the block size of the sequential input file. The default block size is 2 000 bytes per block.

PDEV (TAPE or DASD) (DB2 Server for VSE Only)
is an optional parameter that specifies the device type (DASD or TAPE) of the sequential (SAM) input file. If PDEV(DASD) is specified, the file resides on any device supported by the VSE DTFSD macro. VSAM-managed SAM does not support spanned records. If PDEV(TAPE) is specified, the file resides on any device supported by the VSE DTFMT macro. The default is PDEV(TAPE).

NOREWIND or REWIND
controls tape file rewind processing performed during OPEN processing. This parameter is valid only if you specify TAPE for PDEV. The default processing is REWIND.

NOREWIND
specifies that the tape file is not to be rewound by OPEN processing. If NOREWIND is specified for input tape files referenced by a series of SCHEMA commands, you must ensure that the tape files being referenced are in ascending sequence. For example, if NOREWIND is specified in a sequence of two SCHEMA commands and the first command reads tape file 2, then the second command must reference tape file 3 or a higher number. If it references tape file 1, an OPEN error occurs.

REWIND
specifies OPEN processing to rewind the tape file.

Table 8. Contents of the Schema (in a Sequential Input File)

Format:

```
                                                   ┌───────────────────────────────┐
                                           ┌;┐     ▼                               │
►►──CREATE SCHEMA AUTHORIZATION──authorization_id──┴─┴──────┬───────────────────────────┬──┴──────────────►◄
                                                    └schema_statement──┬;┬──┘
                                                                       └─┘
```

Example:

```
-- create table TAB1 and give Jones SELECT privilege
CREATE SCHEMA AUTHORIZATION SMITH
CREATE TABLE SMITH.TAB1 (COL1 CHAR(4))
GRANT SELECT ON TAB1 TO JONES
```

The first line of the above example is a comment line. To insert comments within the schema file:
- Mark the beginning of a comment with two consecutive hyphens (--)
- Begin the comment anywhere on a record or line
- End the comment with the end of the record or line.

The sequential input file must contain only one CREATE SCHEMA statement, which must be the first statement in the file (unless the preceding lines are comments); otherwise, the Database Services Utility issues an error message and stops processing the SCHEMA command.

AUTHORIZATION *authorization id*
: You must be connected as the *authorization id* in the AUTHORIZATION clause. If a schema statement does not specify an owner, the statement is processed for the authorization ID. For example, in Table 8 the SELECT privilege in the GRANT statement is granted on SMITH.TAB1.

schema-statement
: refers to every statement following the CREATE SCHEMA statement. Valid schema statements are:
  - CREATE TABLE
  - CREATE VIEW
  - GRANT (INSERT, SELECT, UPDATE, REFERENCES, ALL and DELETE privileges).

  The schema statements must be entered in uppercase. See the *DB2 Server for VSE & VM SQL Reference* for the correct syntax of valid schema statements. Successful statements are committed if the Database Services Utility AUTOCOMMIT indicator is ON.

  The Database Services Utility issues an error message if any invalid statement is in the schema, and might or might not continue processing on the next statement, depending on the setting of ERRORMODE.

You can change the setting of the ERRORMODE and AUTOCOMMIT indicators **before** the SCHEMA command by issuing the Database Services Utility SET ERRORMODE and SET AUTOCOMMIT commands. If ERRORMODE is set to CONTINUE, processing continues on the next statement following a minor error on any sequential input file statement other than the CREATE SCHEMA statement. If AUTOCOMMIT is ON, the work is committed after each successful statement. For a complete description of these commands, see “SET ERRORMODE” on page 215 and “SET AUTOCOMMIT” on page 214.

Using File Definitions with the DB2 Server for VM SCHEMA Command

The schema file must have a fixed length of 80 characters. This format is used if you do not specify any FILEDEF options when you define a schema file. For example, you can create a FILEDEF such as this:

```
FILEDEF DBSFILE DISK DBSFILE SCHEMIN A
```

where DBSFILE is the name of the schema file as you refer to it in the SCHEMA command.

For a procedure to construct a FILEDEF command, see “Using File Definitions” on page 14. For more information about FILEDEF parameters and options, see Appendix B, “FILEDEF Command Syntax and Notes,” on page 249.

SQL Statement Processing

The SQL statements that you can enter from the (input) control file are:
- SELECT statements (without an INTO clause or host variables)
- Data manipulation statements
- Data control statements
- Data definition statements
- Authorization statements.

You cannot enter some SQL statements in the Database Services Utility (input) control file. You receive an error message if you attempt to use these statements:
- SELECT statements with an INTO clause or host variables
- Cursor management statements (DECLARE, OPEN, FETCH, CLOSE)
- Commands that support dynamic SQL statement execution (PREPARE, DESCRIBE, EXECUTE, EXECUTE IMMEDIATE)
- Exception handling statements (WHENEVER statements)
- INCLUDE statements (INCLUDE SQLCA or INCLUDE SQLDA).

SQL statements entered from a Database Services Utility (input) control file must not be prefixed by EXEC SQL (as they are when embedded within application programs). In DB2 Server for VM, no information should be placed in the input record after the semicolon. This restriction does not apply to positions 73 through 80 of a control-file record when the Database Services Utility control file is assigned as a CMS file. In DB2 Server for VSE, use a semicolon to indicate the end of each SQL statement. Do not use the continuation character required by ISQL for SQL statements that span record boundaries. Except for positions 73 through 80, no information should be placed in the input record after the semicolon.

SELECT and Arithmetic Exceptions

Database Services Utility flags arithmetic exceptions in outer select-lists by filling the corresponding fields with number (or pound) signs (#). When one or more of these exceptions occur, (for example, when a number is divided by zero) a

Database Services Utility and an SQLCODE message are issued after the results are retrieved. If more than one arithmetic exception occurs, only one message—referring to the first exception—is issued. This is to alert you to any warnings occurring during Database Services Utility (input) control file processing.

Processing Summary

Figure 78 summarizes the Database Services Utility processing that you can perform on tables and views.

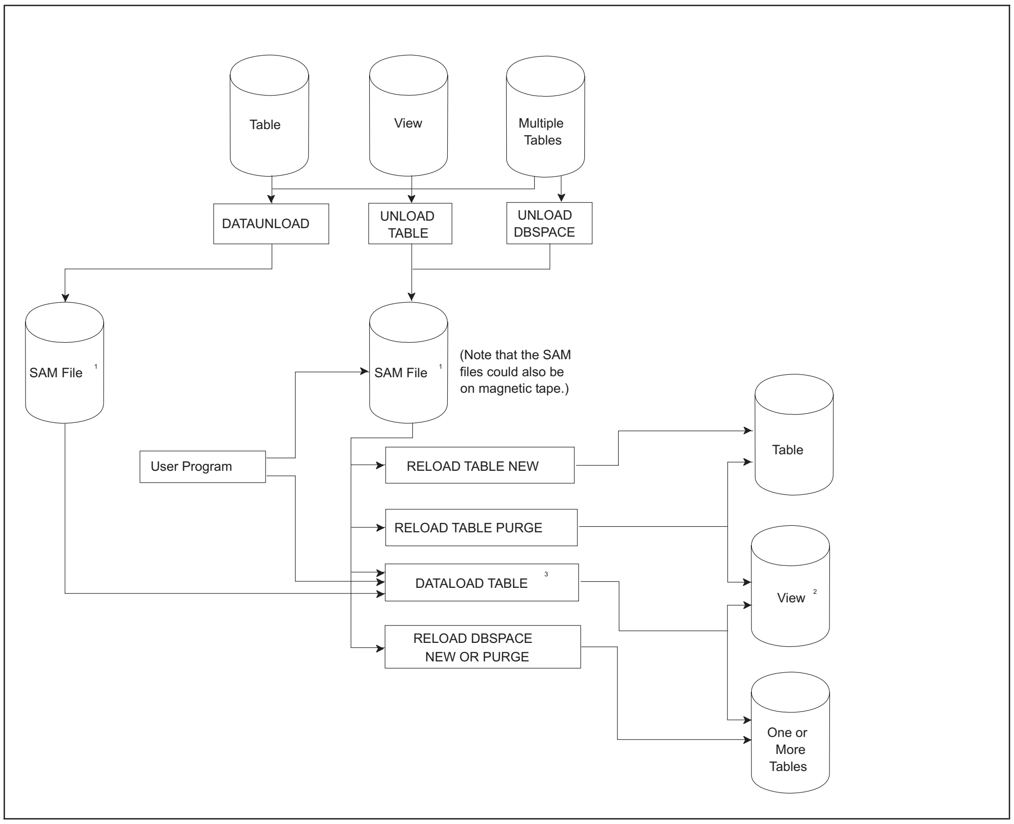


Figure 78. Database Services Utility Processing

Notes:

1. See Database Services Utility UNLOAD record format description in "UNLOAD DBSPACE" on page 201 and "UNLOAD TABLE" on page 204.
2. View definitions must not violate any of the rules related to SQL INSERT processing. In general, the view must be:
   - Defined only on one table
   - Defined to include all the NOT NULL columns of the underlying table
   - Defined without using virtual column definitions.
3. If SAM files created by Database Services Utility UNLOAD processing are used as input to Database Services Utility DATALOAD processing, the rules below must be followed. (This description assumes that you are thoroughly familiar with the Database Services Utility UNLOAD record formats and the contents of each Database Services Utility UNLOAD record type.)
   - All input data records selected for DATALOAD processing must contain data fields in the same position in each data record up to the last position of a data record type referenced by DATALOAD commands.

This rule implies that all input data record varying-length fields must have the same length. Any fields that permit nulls must all be null or must all contain data.

- The DATALOAD input-record-id-clause must be employed to select at least the UNLOAD record type 60 for DATALOAD processing.
- The position of the column data in the input data record must be computed from the following information:
  – Order of column definition in source table or view
  – Actual length of column data at time of UNLOAD
  – Fixed "overhead" in data records created by Database Services Utility UNLOAD processing.

Load-Data Commands

DATALOAD TABLE

The DATALOAD command and subcommands are contained on more than one input record. If, for example, you want to load data into 10 columns of a table, the first input record contains the DATALOAD command, and the next 10 input records contain the Table Column Identification (TCI) subcommands.

The DATALOAD command cannot be continued onto a second input record; it must be completed on a single record. The record immediately following the DATALOAD command must contain a TCI subcommand.

DATALOAD TABLE Format

Format:

```
►►──DATALOAD──TABLE──(table_name)──┬──────────────────────────┬──────────►◄
                                   └─input_record_id_clause─┘

►►──table_column_id_subcommand───────────────────────────────────────────►◄

►►──infile_subcommand──┬──────────────────────────────────┬──────────────►◄
                       │  ┌─.────────────────┐            │
                       │  ▼                  │            │
                       └──user_data_record───┴──ENDDATA───┘
```

Note: Detailed syntax is shown in the following pages.

Example:

```
DATALOAD TABLE(SMITH.ACTIVITY)
ACTNO 1-3
ACTKWD 5-10
ACTDESC 12-21
INFILE(NEWACT)      <-------in DB2 Server for VM
INFILE(NEWACT BLKSZ (2048) PDEV (TAPE) NOREWIND RECFM(FB)) <-in DB2 Server for VSE
```

Authorization: You must have the **INSERT** and **SELECT** privilege on the tables affected by the command.

TABLE (*table_name***)**
identifies the table (called *table_name*) to be loaded. (The table must already exist.) You can further identify the table by specifying the *owner* of the table. For more information about identifying tables see “Qualifying Object Names” on page 110. A synonym cannot be used as *table_name*. You can specify a view name instead of a table name if the view meets the following requirements:

- The view is defined on a single table.
- The view definition includes all NOT NULL columns in the table. That is, all columns outside of the view definition must permit the insertion of nulls.
- The view does not contain a column that is a virtual column.

  A **virtual column** is a column of a view that is not derived directly from a column of a table. For example, view columns defined with expressions such as BONUS+COMM, PRSTAFF*1.5, or AVG(BONUS) are virtual columns.

When loading data into a view that was created using the WITH CHECK OPTION clause, the database manager checks all inserts and updates to the view against the view definition and rejects them if the row to be inserted or updated does not conform to the view definition.

table_name
identifies the table to be loaded.

input_record_id_clause
is optional; it allows you to selectively load records into the table. Records are used for DATALOAD processing only if they contain the value specified in the *input_record_id_clause*. All input data records are loaded into the table if you omit this parameter.

If multiple DATALOAD commands are supplied before an INFILE command and an *input_record_id_clause* appears on one of them, all the DATALOAD commands must have the clause. Database Services Utility error messages are generated and no DATALOAD processing is performed if you break this rule.

The parameters of *input_record_id_clause* are:

startpos
identifies the starting position in the input record of the identification value. Position 1 of the input record is the first position of the logical record. If variable-length input records are used, *startpos* 1 to 4 refers to the record length control field. As a result, *startpos* 5 refers to the first data position.

endpos
identifies the last position of the identification value. If the value occupies only one position, you need only specify *startpos*. Blanks are not allowed between the starting position, hyphen, or ending position values.

constant
identifies the identification value. If an input record contains this value in the specified location, it is used for loading the specified table.

The value cannot be continued onto a second input record. It can be one of the following:

- A character-string constant that satisfies both of these requirements:
  – Must be enclosed in single quotation marks (')
  – Has a value of maximum length equal to *endpos*–*startpos*+1.
- A one-position unsigned integer constant (range is `0` to 255)

- A two-position optionally signed integer constant (default is a positive value)
- A four-position optionally signed integer constant (default is a positive value).

Notice that one-, two-, or four-position refers to the length the value occupies in the input record, not the length it occupies in the clause itself. See Figure 79 through Figure 82 for examples to clarify this definition.

Examples of Input-Record-Id-Clause

```
 IF POS (20-22) =  'RT1'     1
 IF POS (20-21) ¬= '01'

1
To be used for DATALOAD, positions 20 to 22 of the input data record
must contain the character string RT1.
```

Figure 79. Character-String Constant Value Used in the Input-Record-Id-Clause

```
 IF POS (20)   = 255      1
 IF POS (16)   ¬= 25
 IF POS (35)   >  3

1
To be used for DATALOAD, positions 20 of the input data record
must contain a hex FF value.
```

Figure 80. One-Position Integer Constant Value Used in the Input_Record_Id_Clause

```
 IF POS (20-21)   = 1       1
 IF POS (35-36)   <> 50
 IF POS (5-6)     >=  +32767
 IF POS (3-4)     <=   32767

1
To be used for DATALOAD, positions 20 through 21 of the input
record must contain a hex 0001 value.
```

Figure 81. Two-Position Integer Constant Value Used in the Input_Record_Id_Clause

```
 IF POS (20-23)   = 15      1
 IF POS (16-19)   ¬= 50
 IF POS (21-24)   >=  +2000123563
 IF POS (20-23)   <   1839107489

1
To be used for DATALOAD, positions 20 through 23 of the input
record must contain a hex 0000000F value.
```

Figure 82. Four-Position Integer Constant Value Used in the Input_Record_Id_Clause

Table_Column_Id Subcommand

Format:

```
►►─column_name─startpos─┤ options ├─┤ null_current_clause ├──────────►◄
```

options

```
                   ┌─.───────────┐
                   ▼ ┌─CHARacter─┐ │
├──┬─────────┬──────┴─┴─data_type─┴─┴──────────────────────────────────┤
   └─-endpos─┘
```

The next record following the DATALOAD command must contain a Table Column Identification (TCI) subcommand. If it does not, the Database Services Utility issues an error message. TCI subcommands identify the location in the input records of the data for a table column. Only one TCI subcommand can appear in an input record. The command parameters must not span input records, and the *column_name*, *startpos*, and *endpos* parameters must be specified first in the command and in that order.

The data must be in the same record positions in all records that relate to the table.

column_name
: specifies the name of the table column where the input data is to be stored.

startpos
: identifies the starting position of the data in each input data record. Position 1 of the input record corresponds to the first position of the logical input record. If variable-length input records are used, *startpos* 1 to 4 will refer to the record length control information. As a result, *startpos* 5 refers to the first data position.

endpos
: identifies the end position of the data in each input data record. You can omit this parameter if the data occupies only one position in the input record. If you specify this parameter, do not place blanks between the starting position and the hyphen, or between the hyphen and the ending position.

data-type
: identifies whether character, fixed-binary, floating-point, zoned, packed decimal, or graphic data values are contained in the record positions specified. The data type specification can appear either before or after null or current if both parameters are entered. The data type parameter is optional, and the default data type is character. The valid data type identifiers that you can specify are:

CHAR or CHARACTER
If the column-type is CHAR, an all-blank input record data field results in a sequence of blanks being inserted in the table.

If the table column is defined with a column-type of VARCHAR, trailing blanks are removed from the input record data field before the length of the field is established. An all-blank input record data field targeted for a varying-length character column results in a length of 0. (A sequence of blanks is not inserted in the database.)

If extended DBCS is in effect for a database, character input data can contain DBCS characters with shift-in and shift-out delimiters, but the Database Services Utility does not ensure that shift-in and shift-out delimiters are balanced.

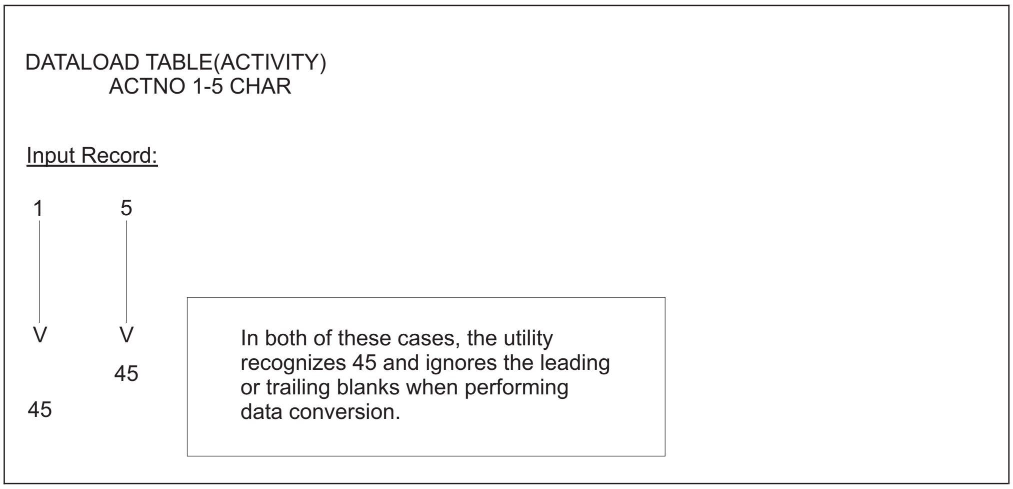
If the table column is defined as numeric (SMALLINT, INTEGER, DECIMAL, or FLOAT), character (EBCDIC) input data must be in the form of an SQL INTEGER, DECIMAL, or FLOAT constant. The character input data is then converted by Database Services Utility processing.

Note: If a data field contains an EBCDIC numeric value that is not in the form of an SQL INTEGER, DECIMAL, or FLOAT constant, (or if it contains an implied decimal point), the data field can meet the requirements of a DATALOAD ZONED input data field.

The type of representation allowed for numeric values in character (CHAR or CHARACTER) input data fields depends on the data type of the target numeric column:

- A value in an SQL INTEGER constant format is valid for a SMALLINT or INTEGER column.
- A value in an SQL INTEGER or DECIMAL constant format is valid for a DECIMAL column.
- A value in an SQL INTEGER, DECIMAL, or FLOAT constant format is valid for a FLOAT column.

The number is converted regardless of its position in the field, but leading and trailing blanks are ignored. If you specify a data field in positions 1 to 5 and type a 2-character number in positions 4 and 5, the number is recognized. In the example below, the TCI subcommand says that data for ACTNO (which has a SMALLINT data type) is in record positions 1 through 5. Additionally, it identifies that the data within those record positions is character. Thus, the Database Services Utility must convert the character data on the input record to a SMALLINT value before it can insert it into the ACTNO column shown in Figure 83.



```
DATALOAD TABLE(ACTIVITY)
         ACTNO 1-5 CHAR
```

Input Record:

In both of these cases, the utility recognizes 45 and ignores the leading or trailing blanks when performing data conversion.

Figure 83. Character Input Data Used in Data type-n

The precision and scale represented in the CHAR field targeted for a table column with a data type of DECIMAL must be less than or equal to the precision and scale defined for the column. Leading zeros after the optional sign are ignored; thus, you can code:

```
+000000011.1
```

on an input data record, and it fits into a column with a data type of DECIMAL(3,1).

Character is the default data type for input records.

DATE

If the table column is defined with a column type of DATE, input data can be in one of the following formats:

```
yyyy-mm-dd              (ISO, JIS format)
dd.mm.yyyy              (EUR format)
mm/dd/yyyy              (USA format)
installation defined    (Local format)

where:

 yyyy is the year
 mm is the month
 dd is the day
```

Local Date Format

A database administrator can change the date default format, which is defined in the SYSTEM.SYSOPTIONS table, from ISO (which is the system-supplied database default form) to any installation-defined format. See the *DB2 Server for VSE System Administration*, or *DB2 Server for VM System Administration* manuals for information about installation-defined formats and their interface.

Note: You can omit leading zeros from months and days, but do not replace them with blanks. For example, *2000-1-1* is valid while *2000-ƀ1-ƀ1* is not.

TIME

If the table column is defined with a column type of TIME, input data can be in one of the following formats:

```
hh:mm AM or hh:mm PM  (USA format)
hh.mm[.ss]            (ISO, EUR format)
hh:mm[:ss]            (JIS format)
installation defined  (Local format)

where:

 hh is the hour
 0 <= hh <= 12  for USA format
 0 <= hh <= 24  for ISO, EUR, JIS format
 mm is the minutes
 ss is the seconds
```

In the USA time format, you can specify zero in the hh field only for 00:00 a.m.

Local Time Format

A database administrator can change the time default format, which is defined in the SYSTEM.SYSOPTIONS table, from ISO (which is the system-supplied database default form) to any installation-defined format. See the *DB2 Server for VSE System Administration* or the *DB2*

Server for VM System Administration manual for information about installation-defined formats and their interface.

Note: Leading zeros can be omitted from hours. The specification of seconds is optional.

TIMESTAMP

If the table column is defined with a column type of TIMESTAMP, input data must be in the following format:

```
yyyy-mm-dd-hh.mm.ss[.[nnnnnn]]

where

 yyyy-mm-dd is the date (see ISO DATE format)
 hh.mm.ss is the time (see ISO TIME format)
 nnnnnn is the microseconds
```

Notes:

1. Leading zeros can be omitted from the month, day, and hour.
2. The microsecond format is optional.

FIXED or INT or INTEGER

If the table column is defined with a data type of SMALLINT, the input can be in a 1-byte or 2-byte binary data field. Table columns defined with a data type of INTEGER can be loaded from a 1-byte, 2-byte, or 4-byte binary input data field.

The value ranges for binary input data fields are:

- A 1-byte binary data field can contain an 8-bit binary integer with a value range of 0 to 255.
- A 2-byte binary data field can contain a 15-bit binary integer with the value range described for a table column defined with the data type SMALLINT.
- A 4-byte binary data field can contain a 31-bit binary integer with the value range described for a table column defined with the data type INTEGER.

FLOAT or REAL

If FLOAT is specified for 4-byte floating-point binary input data, set *startpos* and *endpos* so that (endpos−startpos+1) = 4. The table column identified must be defined with a data type of REAL or FLOAT(n) where n is from 1 to 21.

FLOAT or DOUBLE PRECISION

If FLOAT is specified for 8-byte floating-point binary input data, set *startpos* and *endpos* so that (endpos−startpos+1) = 8. The table column identified must be defined with a data type of FLOAT, DOUBLE PRECISION, or FLOAT(n) where n is from 22 to 53.

Note: If 8-byte floating-point binary data is loaded into a 4-byte floating-point table column, a number of digits of precision are lost.

DECIMAL or DEC (*scalevalue*)

If you have packed decimal input data, the table column must be defined with a DECIMAL data type. The precision of the input decimal data field value must be equal to or less than the precision of the target decimal column. The Database Services Utility takes the scale (number of positions to the right of the implied decimal point) of an input record decimal data value from the scale of the target column unless you specify the optional DECIMAL(*scalevalue*) form of the command parameter.

The optional *scalevalue* is an integer value (0 through 31) identifying the number of scale positions in the input record decimal data value. A *scalevalue* equal to or less than the scale of the target DECIMAL column is allowed. A Database Services Utility processing error occurs if the *scalevalue* is greater than the scale of the target DECIMAL column.

If the DECIMAL(*scalevalue*) form of the parameter is used, no blanks are allowed within the parameter specification.

Columns defined as NUMERIC are treated as DECIMAL data types.

ZONED (*scalevalue***)**

If the input record data field has a zoned value, the target table column must be defined as numeric (SMALLINT, INTEGER, DECIMAL, FLOAT). See Figure 84 on page 154 for examples.

Three variations or types of zoned data input are supported (see description below). The type of zoned data is identified by Database Services Utility processing. Database Services Utility processing converts the zoned data to the data type of the target numeric column.

Note: If a data field contains an EBCDIC numeric value with an explicit decimal point, or otherwise does not meet the requirements of a zoned input data field, the data field can meet the requirements of a DATALOAD character (CHAR) input data field. DATALOAD character input data is described in this section.

If the zoned field is for a DECIMAL column, it must contain a value with a precision less than or equal to the precision of the target DECIMAL column. The Database Services Utility uses the scale (number of positions to the right of the implied decimal point) of the target DECIMAL column for the scale of a zoned value in the input record unless the optional ZONED(*scalevalue*) form of the command parameter is specified.

The optional *scalevalue* is an integer value (0 through 31) identifying the number of scale positions in each input record zoned data field value. A *scalevalue* equal to or less than the scale of the target DECIMAL column is allowed. A Database Services Utility processing error occurs if *scalevalue* is greater than the scale of the target DECIMAL column.

If the ZONED(*scalevalue*) form of the parameter is used, no blanks are allowed within the parameter specification. Also, *scalevalue* is ignored if the target column is defined as a SMALLINT, INTEGER, or FLOAT column.

The Database Services Utility zoned data support is based on the definition of a zoned field that is described in the publication *IBM System/370 Principles of Operation* manual. It includes support for standard and extended decimal items (zoned decimal items).

The following three variations of a zoned input data field are supported:

1. A **standard** zoned data field.

   A numeric value within a standard zoned data field has the following format:

   - Each digit of a number is represented by a single byte.
   - The 4 high-order bits of each byte are zone bits except for the 4 high-order bits of the low-order byte, which represent the sign of the number.
   - The 4 low-order bits of each byte contain the value of the digit.

   The valid zone bit configuration for a standard zoned data field is:

1111 (hex F)

The valid plus-sign bit configurations for a standard zoned data field are:
1010 (hex A)
1100 (hex C)
1110 (hex E)
1111 (hex F)

The valid minus-sign bit configurations for a standard zoned data field are:
1011 (hex B)
1101 (hex D)

2. A zoned field with a **leading sign**.

   A zoned data field value with a leading sign has a format identical to the zoned data format described above except that the 4 high-order bits of the high-order byte represent the sign of the number. The 4 high-order bits of the low-order bytes contain zone bits.

   The valid plus-sign bit configuration for a zoned data field with a leading sign is:
   1100 (hex C)

   The valid minus-sign bit configuration for a zoned data field with a leading sign is:
   1101 (hex D)

   The valid zone bit configurations for a zoned data field with a leading sign is:
   1111 (hex F)

3. A zoned field with a **trailing sign in a separate position**.

   A numeric value within a zoned data field with a trailing sign in a separate position has a format similar to the zoned data format described above except that the high-order 4 bits of the high-order and low-order numeric value bytes contain zone bits. The sign of the numeric data value is contained in a separate low-order data value byte.

   The valid zone bit configuration for the numeric value bytes in a zoned data field with a trailing sign in a separate position is:
   1111 (hex F)

   A plus-sign is represented in the separate low-order data field position by an EBCDIC plus (+) sign (hex 4E) or by a blank (hex 40). A minus-sign is represented in the low-order data field position by an EBCDIC minus (–) sign (hex 60).

HEXADECIMAL CONTENTS OF FIVE(5) POSITION ZONED DATA FIELD	DESCRIPTION OF VALUE LOADED INTO NUMERIC COLUMN			
	SMALLINT	INTEGER	DECIMAL(5,2)	FLOAT
(...System/370* Zoned Data Formats...)				
F1F1F1F1A1	11111	11111	111.11	1.1111E+04
F1F1F1F1B1	−11111	−11111	−111.11	−1.1111E+04
F1F1F1F1C1	11111	11111	111.11	1.1111E+04
F1F1F1F1D1	−11111	−11111	−111.11	−1.1111E+04
F1F1F1F1E1	11111	11111	111.11	1.1111E+04
F1F1F1F1F1	11111	11111	111.11	1.1111E+04
(...COBOL Standard Zoned Data...)				
F1F1F1F1F1	11111	11111	111.11	1.1111E+04
F1F1F1F1C1	11111	11111	111.11	1.1111E+04
F1F1F1F1D1	−11111	−11111	−111.11	−1.1111E+04
(...COBOL Zoned Data with Leading Sign...)				
F1F1F1F1F1	11111	11111	111.11	1.1111E+04
C1F1F1F1F1	11111	11111	111.11	1.1111E+04
D1F1F1F1F1	−11111	−11111	−111.11	−1.1111E+04
(...COBOL Zoned Data with Trailing Sign in Separate Position...)				
F1F1F1F140	1111	1111	11.11	1.111E+03
F1F1F1F14E	1111	1111	11.11	1.111E+03
F1F1F1F160	−1111	−1111	−11.11	−1.111E+03
(...Miscellaneous Other Formats Accepted...)				
4040404040	0	0	0	0.0E0
404040F0F2	2	2	.02	2.0E+00
4040F24040	2	2	.02	2.0E+00
40C1F1F1F1	1111	1111	11.11	1.111E+03
40D1F1F1F1	−1111	−1111	−11.11	−1.111E+03
404040F24E	2	2	.02	2.0E+00
F24E404040	2	2	.02	2.0E+00
404040F260	−2	−2	−.02	−2.0E+00
F260404040	−2	−2	−.02	−2.0E+00

Figure 84. Examples of Valid Zoned Data Input

GRAPHIC or GR or G

If the input field contains double-byte character set (DBCS) data, the table columns must be defined with a data type of GRAPHIC, VARGRAPHIC, or long fields.

One DBCS character is contained in 2 data-field bytes. The input data field must be an even number (2, 4, 6...100, and so forth) of positions (bytes) in length, or a Database Services Utility processing error occurs.

The shift-out and shift-in delimiters are optional in the input data field.

If the first position of the input data field contains a shift-out delimiter (hex 0E), the last position of the data field must contain a shift-in delimiter (hex 0F); if it does not, a Database Services Utility processing error occurs.

If the first position of the input data field does not contain a shift-out delimiter, no shift-in delimiter is expected.

See Table 9 on page 165 for a summary of data type conversions.

null-current-clause

allows you to specify that a NULL, CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP is to be loaded in place of the input record data for a table column. To determine when a null or current value is to be loaded, a comparison is done between two values. The first value is taken from the input record; you specify the positions of the input record that contain this value. The second value is specified in the null or current clause.

No embedded blanks are allowed in the null or current clause within the left and right parentheses enclosing the *startpos* and *endpos* values. The format of the null or current clause is:

null-current-clause

```
►►──┬─NULL──────────────┬──┬──────┬──POS──(──startpos──┬─────────┬──)──┬─ =  ─┬──constant──►◄
    ├─CURRENT DATE──────┤  └─IF───┘                    └─-endpos─┘     ├─ <> ─┤
    ├─CURRENT TIME──────┤                                              ├─ ^= ─┤
    └─CURRENT TIMESTAMP─┘                                              ├─ <  ─┤
                                                                       ├─ >  ─┤
                                                                       ├─ <= ─┤
                                                                       └─ >= ─┘
```

where the following is true, as appropriate:
- NULL IF POS is for null columns.
- CURRENT DATE IF POS is for date columns.
- CURRENT TIME IF POS is for time columns.
- CURRENT TIMESTAMP IF POS is for timestamp columns.

These special registers identify the start of a null or current clause. Note that IF is optional; for example, you can specify the keyword phrase NULL IF POS or NULL POS.

Note: CURRENT TIMEZONE is not supported.

When CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP is to be loaded, you must provide a value for the *endpos* parameter of the TCI subcommand so that the correct length of the data field can be loaded. You need a minimum of 10 bytes for CURRENT DATE, a minimum of 5 bytes for CURRENT TIME, and a minimum of 19 bytes for CURRENT TIMESTAMP. These values follow the rules for ISO formats. To ensure that each data field is sufficiently large to accommodate the maximum value that can be entered for the field, you should define 10 bytes for CURRENT DATE, 8 bytes for CURRENT TIME, and 26 bytes for CURRENT TIMESTAMP. Refer to the input data formats required for date, time, and timestamp.

startpos

identifies the starting position in the input data record of the value that identifies a null or current table-column value. Position 1 of the input data record is the first position of the logical record. If variable-length input records are used, *startpos* 1 to 4 will refer to the record length control information. As a result, *startpos* 5 refers to the first data position. The null or current identifier value positions can be the same as, or different from, those specified for the associated data field.

endpos
identifies the last input data record position of the null or current table column identification value. If the value occupies only one input data record position, this parameter is not required.

constant
specifies the null or current table column identification value. The value cannot be continued to a second input record but can be one of the following:

- A character-string constant that:
  - Must be enclosed in single quotation marks (')
  - Has a value of maximum length equal to *endpos*–*startpos*+1.

  See Figure 85 on page 158 for examples.
- A one-position unsigned integer constant (range `0` to 255). See Figure 86 on page 158 for examples.
- A two-position optionally signed integer constant (default is a positive value). See Figure 87 on page 158 for examples.
- A four-position optionally signed integer constant (default is a positive value). See Figure 88 on page 158 for examples.

Overlapping Column Position Specifications

The *startpos* and *endpos* in the *null-current-clause* need not depend on the positions occupied by data fields in the sequential input file specified in the INFILE subcommand; however, if the positions of the data fields and the positions specified by *startpos* and *endpos* in the *null-current-clause* overlap, data can be overlaid.

During DATALOAD processing, the database manager generates an input buffer to hold one row of data for the table. When a TCI subcommand is encountered, one row of the data, either embedded or in the specified input file, is written to the input buffer. The TCI subcommand then writes the CURRENT DATE, CURRENT TIME or NULL characters to the buffer as required. The data from the specified positions in the input buffer is then written to the table. This process is performed for every row in the table. If the positions of the data fields and the positions specified by *startpos* and *endpos* in the *null-current-clause* overlap, the data in the buffer may be overlaid and cause unexpected results or errors. The following example illustrates how data may be overlaid.

```
CREATE TABLE TIMING
    (  START_DATE     DATE,
       START_TIME     TIME);

DATALOAD TABLE(TIMING)
     START_DATE 3-12 CURRENT DATE IF POS(1-10) = '          '
     START_TIME 5-12 CURRENT TIME IF POS(4-11) = '        '
INFILE(*)
                       -- FIRST 12 COLUMNS ARE BLANK
ENDDATA;
```

In this example, the START_DATE, START_TIME and the *startpos* and *endpos* of the IF POS clause overlap. The first row of the embedded data is loaded into the input buffer. The first TCI subcommand in the DATALOAD command checks column 1 to 10 in the buffer and determines that POS(1-10) = '   ' is true. The CURRENT DATE is then written into positions 3 to 12 in the input buffer. The second TCI subcommand checks positions 4 to 11 of the input buffer; however, positions 3 to 12 contain part of CURRENT DATE; therefore the IF POS(4-11) '   ' clause is not true. The data for START_TIME is then taken from column 5 to 12 in the input buffer when the START_TIME column is written to the TIMING table. Because positions 5 to 12 in the input buffer were already overwritten by the first TCI command, those positions now contain part of CURRENT DATE and the data for START_TIME is not in the correct time format. A syntax error therefore occurs.

Examples of Null-Current-Clause

```
NULL IF POS(20-23) = 'SKIP'
CURRENT DATE IF POS(20-21) ¬= '  '
CURRENT TIME IF POS(20-21) >= '01'
CURRENT TIMESTAMP IF POS(20-21) <> 'XX'
```

Figure 85. Character-String Constant Value Used in the Null-Current-Clause

```
NULL IF POS(20) = 255
CURRENT DATE IF POS(16) ¬= 25
CURRENT TIME IF POS(35) < 3
CURRENT TIMESTAMP IF POS(50) > 16
```

Figure 86. One-Position Integer Constant Value Used in the Null-Current-Clause

```
NULL IF POS (20-21) = 1
CURRENT DATE IF POS (35-36) ¬= 50
CURRENT TIME IF POS (5-6)   <= +32767
CURRENT TIMESTAMP IF POS(9-10) >= 116
NULL IF POS (3-4)   > -32768
```

Figure 87. Two-Position Integer Constant Value Used in the Null-Current-Clause

```
NULL IF POS (20-23) = 1
CURRENT DATE IF POS (16-19) ¬= 50
CURRENT TIME IF POS (21-24) <= +2000123563
CURRENT TIMESTAMP IF POS (20-23) >= -1839107489
```

Figure 88. Four-Position Integer Constant Value Used in the Null-Current-Clause

INFILE Subcommand

```
    (1)
►►───────INFILE──────────────────────────────────────────────────────────────►

►──┬─(──*──┬─────────────────────────────┬─┬──────────────────────────┬─)─┬──►
   │       │                 ┌─No──┐     │ │          ┌─Yes─┐         │   │
   │       └─CONTINUED──(────┴─Yes─┴──)──┘ └─LIST──(──┴─No──┴──)──────┘   │
   └─(──ddname──┤ option_b ├──)───────────────────────────────────────────┘

►──┬──────────────────────────┬──┬──────────────────────────────┬──────────►◄
   └─COMMITCOUNT──(ccount)────┘  └─RESTARTCOUNT──(rcount)───────┘
```

Notes:

1 Option B is valid in DB2 Server for VSE only.

option-b:

tape/disk options for DB2 Server for VSE:

```
├──┬─────────────────────────────┬──┬──────────────────────────────────────┬──►
   │               ┌─2048─┐      │  │                    ┌─REWIND───┐      │
   └─BLKSZ──(──────┴─size─┴──)───┘  │         ┌─(TAPE)───┴─NOREWIND─┴──┐   │
                                    └─PDEV────┴─(DASD)───────────────────┴───┘

►──┬────────────────────┬──┬─────────────────┬────────────────────────────────┤
   └─RECFM──(format)────┘  └─RECSZ──(size)───┘
```

The INFILE subcommand identifies the sequential input file containing the data referenced by the preceding DATALOAD and Table Column Identification subcommands.

This INFILE subcommand not only tells the utility the file the data is in, but also tells it to read that file and load the data into the table(s) identified by the previous DATALOAD TABLE command(s).

The sequential input file can contain fixed, variable, or variable-length spanned records. The records can be blocked or unblocked.

* identifies that input data is embedded within the control statements immediately following this control statement. Subsequent records are processed as user data records until an ENDDATA statement is encountered. If the (input) control file is exhausted before an ENDDATA statement is encountered, a Database Services Utility processing error occurs, and the current logical unit of work is rolled back.

Note: The CONTINUED and LIST parameters are applicable only if the * parameter has been specified.

CONTINUED (No or Yes)
indicates whether or not the input data that is embedded within the control statements can span more than one (input) control file record.

Continued record processing is supported only for data records embedded within the (input) control file because data records in sequential tape or DASD are not restricted to a maximum length of 80 positions. No blanks are allowed between or within this parameter keyword and value specification.

No
indicates that the input data does not span (input) control file records. Specify either NO or N. This is the default.

Yes
indicates that the input data can span (input) control file records. Specify either YES or Y.

If you specify CONTINUED(YES), the actual input data is constructed from one or more (input) control file data records. An input data record with a nonblank value in position 1 indicates that the input data is continued in the next (input) control file data record. An input data record with a blank (hex 40) in position 1 indicates that the input data is not continued in the next (input) control file data record. The first position (position 1) of each (input) control file data record is not included in the actual input data. Data for a column can then be contained in more than one (input) control file data record.

For example, if 10 input control card file data records are required to contain the data for each row of a table, DATALOAD processing constructs a single input data record from 10 consecutive input control card file data records. The relationship between the positions of each of the 10 input control card file data records and the positions of the actual input data record is:

Data Record	Control File Data Record Positions	Actual Input Data Record Positions
1	2-80	1-79
2	2-80	80-158
3	2-80	159-237
4	2-80	238-316
5	2-80	317-395
6	2-80	396-474
7	2-80	475-553
8	2-80	554-632
9	2-80	633-711
10	2-80	712-790

Figure 89. Relationship of Data Records

The maximum possible length of the input data is calculated from the highest *endpos* value specified in any DATALOAD command or TCI subcommand comprising the DATALOAD command set. The *endpos* value specified for an input record data field or the *endpos* value specified in an *input-record-id-clause* or null or current clause is included in this consideration. The maximum length of the actual input data (rounded to the next multiple of 80) is computed by the following formula:

```
Maximum Length        highest endpos value + 80
Actual Input    =  -------------------------------- X  80
Data Record                    80
```

Figure 90. Formula

Notes:

1. Any continuation records that would cause the actual input data record length to exceed the length computed by this formula are read and ignored by DATALOAD processing.
2. Actual input data records containing data to be loaded into a table must be at least as long as the highest *endpos* value specified in a TCI subcommand.

LIST (Yes or No)
indicates whether or not the input data that is embedded within the control statements should be displayed in the report or message file. The LIST parameter is applicable only if the data records are embedded within the (input) control file. No blanks are allowed between or within this parameter keyword and value specification.

Yes
indicates that the embedded data records should be displayed in the report or message file. Specify either YES or Y as the parameter value. The default is LIST(YES).

No
indicates that the embedded data records should not be displayed in the report or message file. Specify either NO or N as the parameter value.

If a data field error is detected in an input data record while the LIST(NO) and CONTINUED(NO) are in effect, the input data record is displayed in the report or message file before the message describing the error.

If LIST(NO) and CONTINUED(YES) are in effect, no input data is displayed in the report or message file if a data field error occurs. A meaningful display of the input data record might not be possible because the data for a record might span multiple 80-byte data records or the data field in error might span input data records. Also, it is likely that continued records contain unprintable data. The commands and input data can be rerun with LIST(YES) specified if the problem cannot be identified.

ddname
in DB2 Server for VM is the name of the sequential input file defined with a CMS FILEDEF command. The file characteristics specified in the FILEDEF command or the default FILEDEF specifications are the source of the input record definition information for the Database Services Utility. Input files with RECFM U, A, or M are not supported.

If you define DATALOAD CMS input files with variable-length spanned records (RECFM=VS or RECFM=VBS), you must use the file-mode number **4**. DATALOAD processing changes the record format from VS or VBS to VB.

Note: The RECFM, RECSZ and BLKSIZE information displayed in the message ARI0868I depends on the CMS FILEDEF command specifications for the DATALOAD input file. If you define DATALOAD input files as VS or VBS, DATALOAD processing changes the record format to VB and the RECFM, RECSZ and BLKSIZE information displayed in the message ARI0868I will indicate this change.

If variable-length input records are used, the data fields referenced by the TCI subcommands must be in the same position for each occurrence of the data record type.

Do not specify SYSIN or SYSPRINT as the *ddname*.

in DB2 Server for VSE: is the TLBL or DLBL job control statement file name for the sequential (SAM) input file or for SYSIPT if you are using the READ member statement. For more information, refer to the *DB2 Server for VSE Program Directory* manual.

You must specify the *ddname* parameter first; that is, you cannot specify BLKSZ, PDEV, RECFM, and RECSZ before the *ddname*. You can specify the other keyword parameters in any order.

BLKSZ (*size*) (DB2 Server for VSE Only)
is a parameter that specifies the block size of the sequential input file. The default block size is 2048 bytes per block.

PDEV (TAPE or DASD) (DB2 Server for VSE Only)
is an optional parameter that specifies the device type (DASD or TAPE) of the sequential (SAM) input file. If PDEV(DASD) is specified, the file resides on any device supported by the VSE DTFSD macro. Managed SAM does not support spanned records. If PDEV(TAPE) is specified, the file resides on any device supported by the VSE DTFMT macro. The default is PDEV(TAPE).

NOREWIND or REWIND (DB2 Server for VSE Only)
controls tape file rewind processing performed during OPEN processing. This parameter is valid only if you specify TAPE for PDEV. The default processing is REWIND.

NOREWIND (DB2 Server for VSE Only)
specifies that the tape file will not be rewound by OPEN processing. If NOREWIND is specified for input tape files referenced by a series of DATALOAD commands, you must ensure that the tape files being referenced are in ascending sequence. For example, if NOREWIND is specified in a sequence of two DATALOAD commands and the first command reads tape file 2, then the second command must reference tape file 3 or a higher number. If it references tape file 1, an OPEN error occurs.

REWIND (DB2 Server for VSE Only)
specifies OPEN processing to rewind the tape file.

RECFM (*format*) (DB2 Server for VSE Only)
is an optional parameter that specifies the format of the records in the input data file. For format, substitute one of the following values:

Value	Meaning
F	fixed, unblocked
FB	fixed, blocked
V	variable, unblocked
VB	variable, blocked
S	variable spanned, unblocked
SB	variable spanned, blocked

The default is RECFM(F).

If variable-length input records are used, the data fields referenced by the TCI subcommands must be in the same position for each occurrence of the data record type.

RECSZ(size) (DB2 Server for VSE Only)
is a parameter that specifies the length of a logical record for the input data file.

Default record size values are specified as follows:

- If RECFM = F or FB, the default record size is the block size.
- If RECFM = V or VB, the default record size is the block size minus four.
- If RECFM = S or SB, the default record size is the block size minus four or the highest input record position referenced, whichever is greater.

COMMITCOUNT (*ccount*)
identifies the frequency of COMMIT action during DATALOAD processing.

ccount
is a number from 1 to 2,147,483,647 indicating that a COMMIT statement should be executed after the number of input data records equal to *ccount* are processed by DATALOAD.

Database Services Utility AUTOCOMMIT ON processing must be in effect when you use DATALOAD COMMITCOUNT processing. If AUTOCOMMIT is OFF and the COMMITCOUNT parameter is used, an error message is written. DATALOAD command processing is not performed.

If a SET ERRORMODE CONTINUE command is in effect during DATALOAD COMMITCOUNT processing, input data records with incorrect data fields might not be used. The incorrect input records are skipped if:

- Multiple DATALOAD commands were used preceding an INFILE subcommand and the records were not used for successful inserts by any other DATALOAD commands.
- An SQL insert error occurs identified by SQLCODE -405, -424, -530, -802, or -803, followed by message ARI0862E, and insert blocking is not in effect.

Insert blocking is not in effect under the following conditions:

- Database Services Utility is running with single user mode.
- Database Services Utility is running with multiple user mode, but was preprocessed with the NOBLOCK option.
- Insert blocking is suppressed by the database manager.

Note: For more information, refer to “Skipping Bad Records” on page 45.

If an invalid *ccount* value is specified, an error message is written, and DATALOAD command processing is not performed.

For DATALOAD CONTINUED record processing, the *ccount* value refers to the number of physical input data records, not the number of logical records constructed from input records. A COMMIT statement is performed when the number of physical input data records processed equals or exceeds the *ccount* value.

RESTARTCOUNT (*rcount***)**
identifies the restart point for DATALOAD processing.

rcount
is a number from 1 to 2147483647 that indicates the number of input data records to be skipped before DATALOAD record processing begins.

If this parameter is omitted, no records are skipped and DATALOAD processing begins with the first input data record.

If an invalid *rcount* value is specified, an error message is written and DATALOAD processing is not performed.

If an end-of-file condition occurs before the number of records specified by the *rcount* value are read, an error condition exists. Error message ARI0844E is written to the message file before DATALOAD processing ends.

For DATALOAD CONTINUED processing, the *rcount* value refers to the number of physical input data records, not the number of logical records constructed from input records. If the *rcount*+1 input record is not the first physical record of a set of physical records comprising a logical record, error message ARI0887E is issued.

ENDDATA Subcommand

Format:

```
▶▶──ENDDATA──────────────────────────────────────────────────────────▶◀
```

The ENDDATA subcommand identifies the end of user data embedded within the (input) control file. This command is valid only if the previous Database Services Utility command processed was an INFILE(*) subcommand.

No other information is allowed in this subcommand. If ENDDATA is not alone on the input record, the utility reads it as data. If ENDDATA is terminated by a semicolon (;), no blanks are permitted between the keyword and the semicolon. SQL comments are not allowed on the ENDDATA subcommand.

During CONTINUED(YES) processing, an ENDDATA command is recognized only if the previous (input) control file data record contains a blank (hex 40) in position

1. If the previous (input) control file data record contains a nonblank in position 1, the ENDDATA command is processed as a continuation data record.

DATALOAD Data Conversion Summary

Table 9 summarizes the data conversion performed by DATALOAD processing.

YES means that the utility performs the conversion; NO means that the utility cannot convert the input data into the data type of the target column. The numbers in the chart refer to the notes below.

Table 9. DATALOAD Data Conversion Table

Input Field Data Type	Target Column Data Type							
	CHAR, VAR-CHAR, or LONG VAR-CHAR	DECIMAL	SMALL-INT	INTEGER	REAL[11]	DOUBLE PRECI-SION[12]	DATE, TIME or TIME-STAMP[13]	DBCS GRAPHIC, VAR-GRAPHIC, or LONG VAR-GRAPHIC
CHAR	Yes[1,4]	Yes[2]	Yes[2]	Yes[2]	Yes[2]	Yes[2]	Yes	No
1-Byte FIXED	No	No	Yes	Yes	No	No	No	No
2-Byte FIXED	No	No	Yes	Yes	No	No	No	No
4-Byte FIXED	No	No	No	Yes	No	No	No	No
4-Byte FLOAT	No	No	No	No	Yes	Yes[10]	No	No
8-Byte FLOAT	No	No	No	No	Yes[9]	Yes	No	No
DECIMAL	No	Yes[3]	No	No	No	No	No	No
ZONED	No	Yes[5,6]	Yes[5]	Yes[5]	Yes[5]	Yes[5]	No	No
GRAPHIC (G)	No	No	No	No	No	No	No	Yes[7]
DATE, TIME, or TIME-STAMP	No	No	No	No	No	No	Yes[8]	No

Notes for Table 9:

1. Character (CHAR) input data fields for VARCHAR or long field columns have trailing (low-order) blanks truncated before the length of the varying column is established. All-blank CHAR input data fields result in a length of `0`.
2. The first trailing blank after the number within a character (CHAR) input data field terminates the character string used for character-to-numeric data conversion. An all-blank CHAR field or a CHAR field with only a sign (+ or –) results in a numeric value of `0`. The data can be in the form of an integer, decimal, or float constant.
3. Decimal (DECIMAL) input data fields should contain data with a precision less than or equal to the precision of the target DECIMAL column. The Database Services Utility uses the scale defined for the target column for the input data unless a scale value equal to or less than that defined for the target column is specified. A Database Services Utility processing error occurs if the scale specified for the input data field is greater than that of the target column.
4. Character (CHAR) input data fields for CHAR columns are padded with trailing blanks if they are less than the length of the target column. CHAR input data fields with a length greater than the length of the target CHAR, VARCHAR, or long field columns are not allowed.
5. Leading and trailing blank positions within a zoned input data field are ignored. An all-blank zoned input data field results in a numeric (SMALLINT, INTEGER, DECIMAL, or FLOAT) column value of `0`. A zoned data input field containing only an EBCDIC plus (+) sign or minus (–) sign is not valid.
6. A zoned (ZONED) data input field should contain a value with a precision less than or equal to the precision of the target DECIMAL column. The Database Services Utility uses the scale defined for the target column as the scale for the zoned data value unless a scale value equal to or less than the scale of the target column is specified. A Database Services Utility processing error occurs if the scale specified for the input data field is greater than that of the target column.
7. A DBCS input data field must be an even number (2, 4, 6,...100, and so forth) of positions (bytes) in length. The shift-out (hex 0E) and shift-in (hex 0F) delimiters can be in the first position (*startpos*) and last position (*endpos*) of the data field. A two-position DBCS data field containing only the shift-out and shift-in delimiter values is treated as a blank input data field.

   If the shift-out and shift-in delimiters are present in the data field, the Database Services Utility treats data field positions *startpos*+1 to *endpos*–1 as DBCS data. The hex value 4040 is treated as a blank DBCS character.

   A DBCS input data field for VARGRAPHIC or a long field has trailing (low-order) DBCS blank characters truncated before the length of the varying column is established. An all-blank DBCS input data field results in a column value with a length = `0` for columns defined with the data type VARGRAPHIC or long field.
8. The datetime data type input and target type must match; for example, the input data type of TIME is valid only for the target column data type of TIME.
9. When 8-byte floating-point data is loaded into a 4-byte floating-point table column, a number of digits of precision are lost. The fraction (mantissa) is reduced from 14 to 6 digits of precision. If an error occurs during the conversion process, the message ARI0864E is generated, and processing of the DATALOAD command stops.
10. 4-byte floating-point data is padded with hex `0000 0000`.

11. The REAL input data field represents single-precision floating-point data and is synonymous with FLOAT(N), where 1 is less than or equal to N, and N is less than or equal to 21.
12. DOUBLE PRECISION represents double-precision floating-point data and is synonymous with FLOAT or FLOAT(N), where 22 is less than or equal to N, and N is less than or equal to 53.
13. When a current date, current time, or current timestamp value is to be loaded, you must provide a value for the *endpos* parameter of the TCI subcommand so that the correct length of the data field can be loaded. You need a minimum of 10 bytes for current date, a minimum of 5 bytes for current time, and a minimum of 19 bytes for current timestamp. These values follow the rules for ISO formats. To ensure that each data field is sufficiently large to accommodate the maximum value that can be entered for the field, you should define 10 bytes for current date, 8 bytes for current time, and 26 bytes for current timestamp. Refer to the input data formats required for date, time, and timestamp.

DATAUNLOAD

DATAUNLOAD Format

Format:

```
>>──DATAUNLOAD────────────────────────────────────────────────────────────><

>>──select_statement──;───────────────────────────────────────────────────><

>>──┬──────────────────────────────────┬──────────────────────────────────><
    │  ┌─,────────────────────────┐    │
    │  V                          │    │
    └────┤ data_field_id Subcommand ├──┘

>>──OUTFILE──(──ddname──┤ option_a ├──)───────────────────────────────────><
```

data_field_id subcommand (DFI)

```
                                                 ┌─CHARacter─┐
>>──┬─column_reference─┬──startpos──┬─────────┬──┼───────────┼─────────────>
    └─integer──────────┘             └─-endpos─┘  └─data_type─┘

>──┬──────────────────────┬───────────────────────────────────────────────><
   └─┤ set_null_clause ├──┘
```

set_null_clause

```
     ┌─IF─┐        ┌─SET─┐
>>───┴────┴──NULL──┴─────┴──POS──(──startpos──┬─────────┬──) =──value────><
                                              └─-endpos─┘
```

option_a valid in DB2 Server for VSE only:

```
├──┬──────────────────┬──┬──────────────────────────────────────┬─────────>
   └─BLKSZ──(size)────┘  │                   ┌─NOREWIND─┐       │
                         │          ┌─(TAPE)─┴─REWIND───┴─┐     │
                         └─PDEV─────┴─(DASD)──────────────┴─────┘

>──┬──────────────────┬──┬────────────────┬───────────────────────────────┤
   └─RECFM──(format)──┘  └─RECSZ──(size)──┘
```

```
Example 1:
DATAUNLOAD
SELECT AVG(BONUS) FROM EMPLOYEE;
OUTFILE(EXTRA)
Example 2:
DATAUNLOAD
SELECT SALARY FROM EMPLOYEE;
SALARY 1-10 CHAR
OUTFILE(REGULAR)
```

Authorization:

All normal **SELECT** privilege ground rules apply.

DATAUNLOAD
identifies the start of the DATAUNLOAD command sequence. A Database Services Utility processing error occurs if other information is present in the (input) control file record after the command identifier DATAUNLOAD.

select-statement
is any valid SQL SELECT statement without host variables. The SQL SELECT statement must begin in the next (input) control file record following the one containing the DATAUNLOAD command. **A semicolon must be used to terminate the SQL SELECT statement.**

The results of the SQL SELECT statement supplied after a DATAUNLOAD command are not written to the Database Services Utility report or message file. An output file data record is written for each row (except those containing data values that exceed the capacity of numeric output record data fields) returned as a result of executing the SQL SELECT statement.

If the user-supplied SQL SELECT statement is not valid, or is not terminated by a semicolon, a Database Services Utility processing error occurs.

If an arithmetic exception occurs, the DATAUNLOAD command handles it in a way similar to arithmetic exceptions under the SELECT statement. (See “SELECT and Arithmetic Exceptions” on page 143 for a description of the way arithmetic exceptions are handled under the SELECT statement.) If an arithmetic exception occurs when the data is to be placed into an output numeric data type field (FIXED, FLOAT, DECIMAL, or ZONED), an error message is issued and processing is terminated because the DB2 Server for VSE & VM system incorrectly reads the number (or pound) symbols (#) used under SELECT as real data. Stopping the processing prevents the arithmetic exception from generating incorrect output.

Data_Field_Id Subcommand

Format:

```
                                               ┌CHARacter─┐
►►─┬column_reference─┬─startpos─┬─────────┬──┼──────────┼──┬───────────────────┬──────────────►◄
   └integer──────────┘          └-endpos──┘  └data_type─┘  └┤ set_null_clause ├┘
```

The next record following the end of the SQL SELECT statement can contain one or more Data Field Identification (DFI) subcommands or an OUTFILE subcommand. If the OUTFILE subcommand is missing, a Database Services Utility processing error occurs. If DFI subcommands are omitted, the default output record format described in the section "DATAUNLOAD Output Data Field Defaults" on page 179 is used.

A DFI subcommand identifies the location in the output record where the data for a column in the select-list should be placed. The subcommand also identifies the output record data-field data type. If the output record data-field data type is different from the select-list column data type, the Database Services Utility converts the column data. The data conversions performed by DATAUNLOAD processing are described in Table 13 on page 187.

Only one DFI subcommand can appear in an input record. The command parameters must not span input records, and the *column_reference*, *startpos*, and *endpos* parameters must be specified first in the command and in that order.

If DFI subcommands are specified, only the data for the select-list columns referenced by these subcommands is unloaded to the output data record. The data for a column in the select-list (explicitly specified, or implicitly specified by the * specification in the SQL SELECT) not referenced by a DFI subcommand is not unloaded. A Database Services Utility warning message identifies each select-list column that is ignored by DATAUNLOAD processing.

The data for the same column in the select-list can be unloaded to more than one output record data field by specifying two or more DFI subcommands that reference the column.

column_reference

identifies the select-list column to be used as the source of the output data field value. *Column_reference* can be any valid form of a table column name or integer that refers to a select-list column. For example, the integer value 1 (hex F1) refers to the first item in the select-list; the integer value 10 (hex F1F0) refers to the 10th item in the select-list.

A Database Services Utility processing error occurs if:

- The column name specified in the DFI subcommand is not in the select-list.
- The integer value identifying the column name exceeds the number of columns in the select-list.

Use the integer notation for *column_reference* to identify the column if:

- The select-list contains columns from different tables with the same column name. For example:

```
SELECT...,EMPLOYEE.EMPNO,EMP_ACT.EMPNO,
                ...FROM EMPLOYEE,EMPLOYEE.ACTIVITY...;
```

- The select-list contains a column that is a constant or is derived from an expression or function:

```
SELECT...,'SALARY(+6%)=',SALARY*1.06,MAX(SALARY)...
                          ...FROM EMPLOYEE...;
```

startpos
identifies the starting position (byte) of the data in each output data record. Position 1 of the output record corresponds to the first position of the logical output record. If variable-length input records are used, *startpos* 1 to 4 refers to the record length control field. As a result, *startpos* 5 refers to the first data position.

endpos
identifies the end position (byte) of the data in each output data record. You can omit this parameter if the data occupies only one position in the output record. If you specify this parameter, do not place blanks between the starting position and the hyphen, or between the hyphen and the ending position.

To unload a column defined with a double-byte character set (DBCS) data type, the length of the output data field must be an even number (4, 6, ..., 100, ...) of positions (bytes) other than 2. A Database Services Utility processing error occurs if DBCS data is identified for an output record data field with an odd number of positions, or with only two positions.

data-type
identifies whether character, graphic, fixed-binary, floating-point, packed decimal, or zoned data values should be placed in the output data record positions specified. The data type specification must appear after the *startpos-endpos* values in the subcommand. The default data type is character. The valid data type identifiers that you can specify are:

CHAR or CHARACTER
If the table column data type is anything but GRAPHIC, you can create a CHAR output data field.

CHAR output data derived from a CHAR, VARCHAR, DATE, TIME, TIMESTAMP, or long field select-list column is left-justified and padded on the right with blanks (hex 40). Trailing (low-order) data is truncated if the output data field length is less than the length of the column data except for TIME, DATE, and TIMESTAMP. For TIME and DATE, an error occurs if the output data field length is less than the length of the column data. For TIMESTAMP, if the output data field length is less than 19 bytes, an error occurs; if the output data field length is less than 26 but greater than or equal to 19 bytes, trailing digits of the MICROSECONDS part of the timestamp are truncated.

The CHAR output data can also be derived from select-list columns with data type SMALLINT, INTEGER, DECIMAL, and FLOAT. See the section, "DATAUNLOAD Data Conversion Summary" on page 187, for a description of the content of CHAR output data fields derived from numeric column data types.

If extended DBCS processing is in effect, character data can contain DBCS/EBCDIC mixed data. See page 231 for a discussion of extended DBCS support.

When an arithmetic exception occurs and the data is to be placed into an output data type field, no error message is issued and processing continues. Number (or pound) symbols (#) are used, as under SELECT, to fill the data type field and to indicate that an exception occurred during processing.

CHAR is the default data type specification.

GRAPHIC or GR or G

If the table column is defined with the data type GRAPHIC, VARGRAPHIC, or long field, you can create a DBCS output data field.

The *startpos* and *endpos* for a DBCS output record data field reflect the number of bytes the data field occupies in the data record; they do not reflect the number of DBCS characters that the data field contains.

The *startpos* of the output data field contains a shift-out (hex 0E) delimiter. The *endpos* of the output data field contains a shift-in (hex 0F) delimiter. Two intervening positions are required for each DBCS character.

The DBCS output record data field must occupy an even number of bytes in the data record, or a Database Services Utility processing error occurs. DBCS column data is truncated if the length of the output record data field is less than the column data length plus 2. For a DBCS column, the column data length equals the number of DBCS characters times 2.

A blank DBCS output data field contains the hex 40 value in all positions except for the first and the last. A null source column value also results in a blank output record data field.

DATE

If the table column is defined with a column type of DATE, output data is in one of the following formats:

```
yyyy-mm-dd           (ISO, JIS format)
dd.mm.yyyy           (EUR format)
mm/dd/yyyy           (USA format)
installation defined  (Local format)

where:

yyyy is the year
mm is the month
dd is the day
```

The format is dependent on the SYSOPTIONS default format value or is specified by the CHAR function in the SELECT statement.

Local Date Format

A database administrator can change the date default format, which is defined in the SYSTEM.SYSOPTIONS table, from ISO (which is the system-supplied database default form) to any installation-defined format. See the *DB2 Server for VM System Administration* and *DB2 Server for VSE System Administration* manuals for information about installation-defined formats and their interface.

Note: See page 169 for information on arithmetic error handling.

TIME

If the table column is defined with a column type of TIME, output data is in one of the following formats:

```
hh:mm AM or hh:mm PM  (USA format)
hh.mm[.ss]            (ISO, EUR format)
hh:mm[:ss]            (JIS format)
installation defined  (Local format)
```

```
where:

hh is the hour
0 <= hh <= 12  for USA format
0 <= hh <= 24  for ISO, EUR, JIS format
mm is the minutes
ss is the seconds
```

The format is dependent on the SYSOPTIONS default format value or is specified by the CHAR function in the SELECT statement.

Local Time Format
A database administrator can change the time default format, which is defined in the SYSTEM.SYSOPTIONS table, from ISO (which is the system-supplied database default form) to any installation-defined format. See the *DB2 Server for VSE System Administration*, or *DB2 Server for VM System Administration* manual for information about installation-defined formats and their interface.

Note: See page 169 for information on arithmetic error handling.

TIMESTAMP
If the table column is defined with a column type of TIMESTAMP, output data is in the following format:

```
yyyy-mm-dd-hh.mm.ss[.[nnnnnn]]

where

yyyy-mm-dd is the date (see ISO DATE format)
hh.mm.ss is the time (see ISO TIME format)
nnnnnn is microseconds
```

If the output data field length is less than 19 bytes long, an error occurs. If the output data field is less than 26 bytes, but greater than or equal to 19 bytes, trailing digits of the microseconds part of the timestamp are truncated.

FIXED or INT or INTEGER
If the table column is defined with a data type of SMALLINT or INTEGER, you can define a fixed-point binary-output data field. If a row selected from the database by the SQL SELECT statement supplied for DATAUNLOAD processing contains a column value that exceeds the capacity of a 1-byte or 2-byte FIXED output data field, an error message is issued, and no output data file record is written for the row.

The value ranges for binary-output data fields are:

- A 1-byte binary data field can contain an 8-bit binary integer with a value range of 0 to 255.
- A 2-byte binary data field can contain a 15-bit binary integer with the value range described for a table column defined with the data type SMALLINT.
- A 4-byte binary data field can contain a 31-bit binary integer with the value range described for a table column defined with the data type INTEGER.

Note: See page 169 for information on arithmetic error handling.

FLOAT or REAL
If the table column is defined with a data type of REAL or FLOAT(*n*),

where *n* is from 1 to 21, you can define 4-byte floating-point binary-output data, where (endpos−startpos+1) = 4. If a row selected from the database by the SQL SELECT statement supplied for DATAUNLOAD processing contains a column value that exceeds the capacity of the FLOAT output data field, an error message is issued, and no output data file record is written for the row.

Note: See page 169 for information on arithmetic error handling.

FLOAT or DOUBLE PRECISION

If the table column is defined with a data type of FLOAT, DOUBLE PRECISION, or FLOAT(*n*), where *n* is from 22 to 53, you can define 8-byte floating-point binary-output data, where (endpos−startpos+1) = 8.

Note: If 8-byte floating-point binary data is unloaded into a 4-byte floating-point output data field, a number of digits of precision is lost. If a row selected from the database by the SQL SELECT statement supplied for DATAUNLOAD processing contains a column value that exceeds the capacity of the FLOAT output data field, an error message is issued, and no output data file record is written for the row.

See page 169 for information on arithmetic error handling.

DECIMAL or DEC

If the table column is defined with a DECIMAL data type, you can specify DECIMAL or DEC for packed decimal output data.

The length of the output data field must be large enough to accommodate all significant digits of the column data value. The minimum length of an output field derived from DECIMAL column data is (column scale/2)+1. The implied scale of the output data field value is the same as that defined for column.

If a row selected from the database by the SQL SELECT statement supplied for DATAUNLOAD processing contains a column value that exceeds the capacity of a decimal output data field, an error message is issued, and no output data file record is written for the row.

Columns defined as NUMERIC are treated as DECIMAL data types.

Note: See page 169 for information on arithmetic error handling.

ZONED

If the table column is defined with a data type of SMALLINT, INTEGER, or DECIMAL, you can specify ZONED for zoned output data. The length of a zoned output record data field derived from a DECIMAL column must be equal to or greater than the column scale.

Each digit of the table column value is represented by a single byte in the zoned output data field. The 4 high-order bits of each byte are the zone bits. The 4 high-order bits of the low-order byte are the sign of the value. The 4 low-order bits of each byte contain the value of the digit.

The zone bits are 1111 (hex F). A plus-sign is represented by the bits 1100 (hex C), and a minus-sign is represented by the bits 1101 (hex D).

The output data field value is right-justified. Leading (high-order) zeros are either added to, or truncated from, the column value depending on the length of the output data field.

If a row selected from the database by the SQL SELECT statement supplied for DATAUNLOAD processing contains a column value that exceeds the capacity of a zoned output data field, an error message is issued, and no output data file record is written for the row.

Examples of zoned output data fields:

The hexadecimal content of a 5-position zoned data field containing the value +00011 is:

Hexadecimal Value	F0	F0	F0	F1	C1
Field Position	1	2	3	4	5

Note: See page 169 for information on arithmetic error handling.

The hexadecimal content of a 5-position zoned data field containing the value -00011 is:

Hexadecimal Value	F0	F0	F0	F1	D1
Field Position	1	2	3	4	5

See Table 13 on page 187 for a table summarizing the data conversion performed by Database Services Utility DATAUNLOAD processing.

set_null_clause
specifies the output data record position and value that identifies a null table column value. The null identifier value can be a character or an integer value (see below); it does not assume the data type specified for the output record data field.

The *set_null_clause* must appear after the *startpos-endpos* values in the subcommand. No embedded blanks are allowed in the *set_null_clause* within the left and right parentheses enclosing the *startpos* and *endpos* values.

set_null_clause:

```
     ┌─IF─┐        ┌─SET─┐
►►───┴────┴──NULL──┴─────┴──POS──(──startpos──┬─────────┬──)──= value────────►◄
                                              └─-endpos─┘
```

The parameters are:

IF NULL SET POS
identifies the start of the *set_null_clause*. You can use either the keyword phrase IF NULL SET POS or NULL POS.

startpos
identifies the starting position (byte) in the output data record of the value that identifies a null table column value. The null identifier value positions can overlap the positions assigned to an output record data field.

Position 1 of the output data record is the first position of the logical record. If variable-length output records are used, *startpos* 1 to 4 refer to the record length control information and the data begins at *startpos* 5.

endpos
: identifies the last output data record position (byte) of the null table column identification value. If the value occupies only one output data record position, this parameter is not required.

value
: specifies the null table column identification value. If an occurrence of the column value is null, the value specified is placed in the output data record positions specified after the data field value for the default output record is set.

  If an occurrence of the column value is not null, no value is placed in the output data record positions specified in the *set_null_clause*. These positions contain blanks (hex 40) if they do not contain a default output record data field value for null column data.

  The *set_null_clause* value cannot be continued to a second output record. It can be one of the following:

  - A character-string constant that:
    - Must be enclosed in single quotation marks (')
    - Has a maximum length equal to *endpos* −*startpos*+1.
  - A one-position unsigned integer constant (0 to 255)
  - A two-position optionally signed integer constant (default is a positive value)
  - A four-position optionally signed integer constant (default is a positive value).

Examples of Set-Null-Clause:

- Character-string constant:
```
IF NULL SET POS(20-23) = 'NULL'
IF NULL SET POS(20-22) = ' ? '
```
- One-position integer constant:
```
IF NULL SET POS(20) = 255
```
- Two-position integer constant:
```
IF NULL SET POS(20-21) = 32767
```
- Four-position integer constant:
```
IF NULL SET POS(20-23) = -1839107489
```

OUTFILE Subcommand

VSE Format:

```
►►──OUTFILE──(──ddname──────────────────────────────────────────────────────►

►──┬────────────────┬──┬────────────────────────────────────────┬───────────►
   └─BLKSZ──(size)──┘  │                   ┌─NOREWIND─┐          │
                       │        ┌─(TAPE)───┴─REWIND───┴─┐        │
                       └─PDEV───┴─(DASD)────────────────┴────────┘

►──┬─────────────────┬──┬───────────────┬──)──────────────────────────────►◄
   └─RECFM──(format)─┘  └─RECSZ──(size)─┘
```

VM Format:

```
►►──OUTFILE──(ddname)──────────────────────────────────────────────────────►◄
```

The OUTFILE subcommand identifies the sequential output file that contains the data referenced by the preceding DATAUNLOAD commands and subcommands. It tells the utility the file in which to put the data and to begin to unload the data.

The sequential output file can contain fixed, variable-length, or variable-length spanned records. The records can be blocked or unblocked. If you want variable length records to be generated, variable-length spanned records must be used if the total length of the column values to be unloaded exceeds 32752 bytes.

A blank (hex 40) is placed in all positions (bytes) of the output data record before the data record field values are inserted.

ddname

in DB2 Server for VM: this is the name of the sequential output file defined with a CMS FILEDEF command. If you define DATAUNLOAD CMS output files with variable-length spanned records (RECFM=VS or RECFM=VBS), you must use file-mode number **4**. DATAUNLOAD processing changes the record format from VS or VBS to U. See Appendix B, "FILEDEF Command Syntax and Notes," on page 249 for more information about undefined (U) record format usage. If a tape output file is used, performance can be improved by using a large block size value (greater than 8244) on the FILEDEF.

The file characteristics specified in the FILEDEF command or the default FILEDEF specifications are the source of the output record definition information for the Database Services Utility. Output files with RECFM U, A, or M are not supported.

Note: The RECFM, RECSZ and BLKSIZE information displayed in the message ARI0868I depends on the CMS FILEDEF command specifications for the DATAUNLOAD output file. If you define DATAUNLOAD output files on CMS as VS or VBS, DATAUNLOAD processing changes the record format to U and the RECFM, RECSZ and BLKSIZE information displayed in message ARI0868I will reflect this change. However, the CMS FILEDEF command that defines the DATALOAD input data file must still specify RECFM=VBS or VS accordingly. Except for the ddname, CMS FILEDEF command information for DATALOAD command processing must be identical to the information in the FILEDEF command used when DATAUNLOAD command processing created the file. See Appendix B, "FILEDEF Command Syntax and Notes," on page 249 for more information about undefined (U) record format usage.

The length of the output record supplied by the FILEDEF must be long enough to contain column data selected for unload (including intervening data field blanks if the default format is used). If it is not, a Database Services Utility processing error occurs. If the length of the output record is greater than the length required to unload the data, the remaining positions of the output record are set to blanks (hex 40).

If variable length output records are used, the data fields referenced by DFI subcommands appear in the same positions on each output data record.

Do not specify SYSIN or SYSPRINT as the *ddname*.

in DB2 Server for VSE: this is the TLBL or DLBL job control statement file name for the sequential (SAM) output file. This parameter must be the first parameter specified; you cannot specify BLKSZ before the *ddname*. The other keyword parameters can be specified in any order.

BLKSZ (*size*)
is an optional parameter that specifies the block size of the sequential output file.

If a tape output file is used, performance can be improved by using a large block size value (greater than 8244).

The default block size values depend upon the output file record format:

- If RECFM = F or FB, the default block size is equal to the highest data field end-position value
- If RECFM = V or VB, the default block size is equal to the highest data field end-position value plus four
- If RECFM = S or SB, the default block size is 2048 bytes.

PDEV (TAPE or DASD)
is an optional parameter that specifies the device type (DASD or TAPE) of the sequential (SAM) output file. If PDEV(DASD) is specified, the file resides on any device supported by the VSE DTFSD macro. An exception to this is VSAM-managed SAM files. VSAM-managed SAM does not support spanned records. If PDEV(TAPE) is specified, the file resides on any device supported by the VSE DTFMT macro. The default is PDEV(TAPE).

NOREWIND or REWIND
controls tape file rewind processing performed during CLOSE processing. This parameter is valid only if you specify TAPE for PDEV. The default is NOREWIND.

NOREWIND
specifies that the tape file will not be rewound by CLOSE processing.

REWIND
specifies that the tape file is rewound by CLOSE processing.

RECFM (*format*)
is an optional parameter that specifies the format of the records in the output data file. For format, substitute one of the following values:

Value	Meaning
F	fixed, unblocked
FB	fixed, blocked
V	variable length, unblocked
VB	variable length, blocked
S	variable spanned, unblocked
SB	variable spanned, blocked

If the length of the logical record (RECSZ) is equal to or less than 32760, the default is RECFM(F). Otherwise, the default is RECFM(SB). The output record format is identified in a Database Services Utility informational message.

Note: If variable-length output records are used, the data fields referenced by the DFI subcommands will be in the same position for each occurrence of a data record. Positions 1-4 of each variable-length record contain record length control information.

RECSZ (*size***)**
is an optional parameter that specifies the length of a logical record for the output data file. The length of the output record must be long enough to contain column data selected for DATAUNLOAD processing (including intervening data field blanks if the default format is used). If it is not, a Database Services Utility processing error occurs.

The default record size values depend upon the output file record format:

- If RECFM = F or FB, the default record size is the block size.
- If RECFM = V or VB, the default record size is the block size minus four.
- If RECFM = S or SB, the default record size is equal to the highest data field end-position value.

DATAUNLOAD Output Data Field Defaults

If no DFI commands are supplied, the output data fields appear in the output record in the same order as the associated columns in the select-list. The output data field associated with the first select-list column starts in position 1 of the fixed length output records or position 5 of variable length (or variable length spanned) output records. Positions 1-4 of variable length or variable length spanned records are reserved for the record length control field. In DB2 Server for VSE, if the record format (RECFM) is not supplied by the OUTFILE subcommand, DATAUNLOAD processing writes either fixed length or variable length spanned output records. Fixed length records are written if the required logical record length is less than 32760 positions; otherwise, variable length spanned records (RECFM=S) are written.

In DB2 Server for VM, the DFI subcommand will refer to the first data position as *startpos* 5. The FILEDEF command that defines the output file always supplies the record format information.

One blank (hex 40) position separates each output record data field. The output data field associated with the next select-list column starts two positions after the trailing (low-order) position of the data field derived from the preceding select-list column.

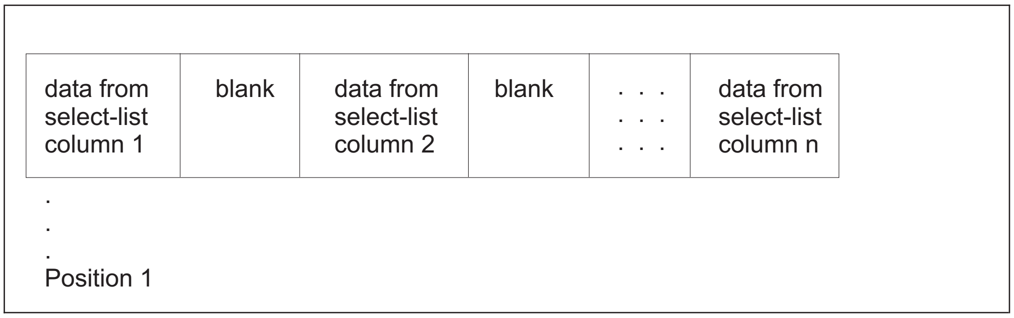

data from select-list column 1
blank
data from select-list column 2
blank
. . .
data from select-list column n
Position 1

Figure 91. Default Fixed-Length Output Logical Record Content

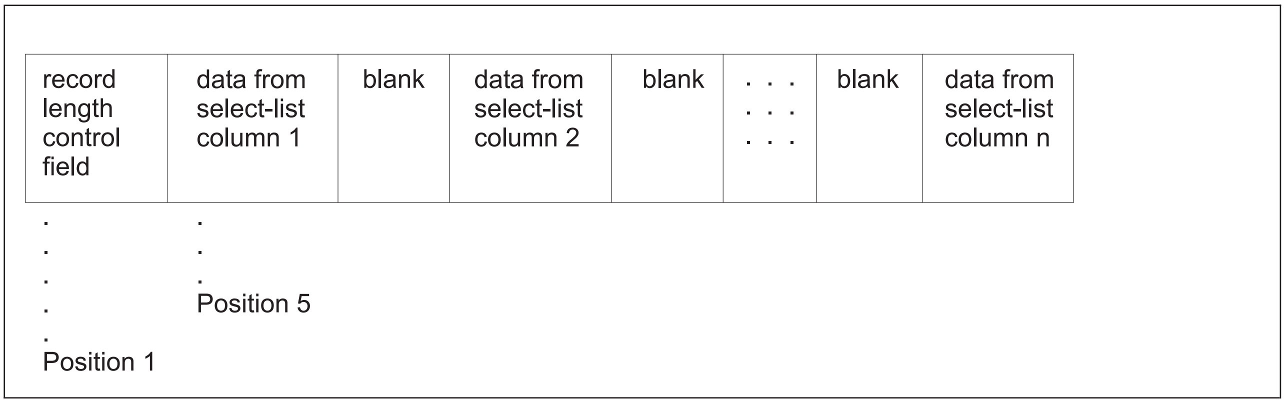

record length control field
data from select-list column 1
blank
data from select-list column 2
blank
. . .
blank
data from select-list column n
Position 5
Position 1

Figure 92. Default Variable-Length Spanned Logical Output Record Content

Default Output Data Field Formats: Table 10 on page 181 summarizes the default output field formats generated by the DATAUNLOAD processing if no DFI subcommands are supplied. The default data type of the output data field is CHAR (or GRAPHIC if the source column contains DBCS data). The format of the data in the output data field depends on the data type, length, or maximum length of the select-list column from which the data is derived.

Table 10. Default Output Formats

Source Column Data Type	Default Database Services Utility DATAUNLOAD Output Data Fields Default Data Type = CHAR
CHAR	Length: Defined length of source column.
VARCHAR length <= 254	Length: Defined maximum length of column. Note: If the actual length of an occurrence of the column data is less than the defined maximum length of the column, the data is left-justified in the output data field and padded with trailing (low-order) blanks.
VARCHAR length > 254 or LONG VARCHAR	Length: 512 positions (bytes). Notes: If the actual length of an occurrence of the column data is greater than 512, the column data is truncated. If the actual length of an occurrence of the column data is less than 512, the column data is left-justified and padded with trailing (low-order) blanks.
SMALLINT	Length: 6 Format: *snnnnn*
INTEGER	Length: 11 Format: *snnnnnnnnnn*
DECIMAL	Length: Precision of source column + 2. Format: Examples: Column Precision=7, Scale=2: *snnnnn.nn* Column Precision=5, Scale=5: *s.nnnnn* Note: NUMERIC is a synonym for DECIMAL.
REAL or FLOAT (N) 1 <= N <= 21	Length: 12 (single precision float). Format: *sn.n*E*snbbbbb* (minimum value) *sn.nnnnn*E*snn* (maximum value) Note: The value is left-justified and, if necessary, padded with trailing (low-order) blanks in the output data field.
FLOAT or DOUBLE PRECISION or FLOAT (N) 22 <= N <= 53	Length: 20 (double precision float). Format: *sn.n*E*snbbbbbbbbbbbbb* (minimum value) *sn.nnnnnnnnnnnnn*E*snn* (maximum value) Note: The value is left-justified and, if necessary, padded with trailing (low-order) blanks in the output data field.

Legend For FLOAT:

s = EBCDIC SIGN: Plus (+) sign (hex 4E)
Minus (-) sign (hex 60)
Blank (hex 40) if null value.

n = EBCDIC numeric character (hex F0–F9)

. = EBCDIC decimal point (hex 4B)

b = Blank (hex 40)

Source Column Data Type	**Default Database Services Utility DATAUNLOAD Output Data Fields Default Data Type = CHAR**
DATE	Default DATE length format ISO 10 *yyyy-mm-dd* JIS 10 *yyyy-mm-dd* EUR 10 *dd.mm.yyyy* USA 10 *mm/dd/yyyy* LOCAL installation defined *yyyy* is the year *mm* is the month *dd* is the day Note: The length and format of the output data field depends on the default DATE for the database. You can query the SYSTEM.SYSOPTIONS catalog to determine the output format for DATE.
TIME	Default TIME length format ISO 8 *hh.mm.ss* JIS 8 *hh:mm:ss* EUR 8 *hh.mm.ss* USA 8 *hh.mm* AM (or *hh.mm* PM) LOCAL Installation defined *hh* is the hour 0 <= *hh* <= 24 for ISO, JIS, EUR formats 0 <= *hh* <= 12 for USA format *mm* is the minute *ss* is the second Note: The length and format of the output data field depends on the default TIME for the database. You can query the SYSTEM.SYSOPTIONS catalog to determine the output format for TIME.
TIMESTAMP	Length: 26 Format: *yyyy-mm-dd-hh.mm.ss.nnnnnn* *yyyy-mm-dd* is the date (ISO format) *hh.mm.ss* is the time (ISO format) *nnnnnn* is the microsecond
GRAPHIC	Length: (Defined length of column * 2) + 2. Note: The first position of the output record DBCS data field contains an SO delimiter and the last position contains an SI delimiter.

Source Column Data Type	Default Database Services Utility DATAUNLOAD Output Data Fields Default Data Type = CHAR
VARGRAPHIC with defined length <= 127	Length: (Defined maximum length of column * 2) + 2. Notes: The first position of the output record DBCS data field contains an SO delimiter and the last position contains an SI delimiter. If the actual length of an occurrence of the DBCS column data is less than the defined maximum length of the column, the data is left-justified and padded with trailing blanks in the second through n-1 positions of the output record field.
VARGRAPHIC with defined length >127 or LONG VARGRAPHIC	Length: 512 positions. Notes: The first position of the output record DBCS field contains an SO delimiter and the last position contains an SI delimiter. If the actual length of an occurrence of the DBCS column data is greater than 510 (255 DBCS characters), the column data is truncated. If the actual length of an occurrence of the DBCS column data is less than 510 (255 DBCS characters), the data is left-justified and padded with trailing blanks in the second through n-1 positions of the output record field.

DATAUNLOAD Default Output Record Format Example: This example unloads data for the columns EMPNO, PROJNO, and EMPTIME in the tables EMP_ACT and EMPLOYEE, based on the selection criteria specified in the WHERE clause. The output records are generated in EMPNO value ascending sequence. Because no DFI subcommands are present, the default DATAUNLOAD output record data field format is used.

The DATAUNLOAD command sequence is:

```
DATAUNLOAD
SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME
FROM EMP_ACT,EMPLOYEE
WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
ORDER BY EMP_ACT.EMPNO;
OUTFILE(OUTPUT1)
```

Figure 93. DATAUNLOAD Command without DFI Subcommands

The Database Services Utility message file output that results is shown in the following examples.

```
ARI0801I DBS Utility started: 11/13/89 16:48:16.
         AUTOCOMMIT = OFF ERRORMODE = OFF
         ISOLATION LEVEL = REPEATABLE READ
------>
------> DATAUNLOAD
------> SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME
------> FROM EMP_ACT,EMPLOYEE
------> WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
------> ORDER BY EMP_ACT.EMPNO;
------> OUTFILE(OUTPUT1)
ARI0852I DATAUNLOAD processing started.
ARI0868I DNAME=OUTPUT1 RECFM=F RECSZ=80 BLKSIZE=80       <------ See Note
ARI0836I Default output record data field positions:
ARI0837I EMPNO 1-6
ARI0837I PROJNO 8-13
ARI0837I EMPTIME 15-21
ARI0835I 74 record(s) written to the output data file.
ARI0855I DATAUNLOAD processing successful.
ARI0802I End of command file input.
ARI8997I ...Begin COMMIT processing.
ARI0811I ...COMMIT of any database changes sucessful.
ARI0809I ...No error(s) occurred during command processing.
ARI0808I DBS processing completed: 11/13/89 16:48:20.
```

Figure 94. DB2 Server for VM Database Services Utility Message File Output

Note: The RECFM, RECSZ, and BLKSIZE information displayed in the message ARI0868I depends on the CMS FILEDEF command specifications for the output file with *ddname*=OUTPUT1.

```
ARI0801I DBS Utility started: 11/13/89 16:48:16.
         AUTOCOMMIT = OFF ERRORMODE = OFF
         ISOLATION LEVEL = REPEATABLE READ
------> CONNECT "SQLDBA " IDENTIFIED BY ********;
ARI8004I User SQLDBA connected to database SQLDBA.
ARI0500I SQL processing was successful.
ARI0505I SQLCODE = 0 SQLSTATE = 00000 ROWCOUNT = 0
------>
------> DATAUNLOAD
------> SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME
------> FROM EMP_ACT,EMPLOYEE
------> WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
------> ORDER BY EMP_ACT.EMPNO;
------> OUTFILE(OUTPUT1)
ARI0852I DATAUNLOAD processing started.
ARI0868I DNAME=OUTPUT1 RECFM=F RECSZ=80 BLKSIZE=80
ARI0836I Default output record data field positions:
ARI0837I EMPNO 1-6
ARI0837I PROJNO 8-13
ARI0837I EMPTIME 15-21
ARI0835I 74 record(s) written to the output data file.
ARI0855I DATAUNLOAD processing successful.
ARI0802I End of command file input.
ARI8997I ...Begin COMMIT processing.
ARI0811I ...COMMIT of any database changes sucessful.
ARI0809I ...No error(s) occurred during command processing.
ARI0808I DBS processing completed: 11/13/89 16:48:20.
```

Figure 95. DB2 Server for VSE Database Services Utility Report Output

The format of the records in the output file identified by the *ddname* OUTPUT1 is shown in Table 11:

Table 11. Default Output Record Format

Record Position	Data Value Source (Column or Other)	Output Record Field Data Type
1-6	EMPNO	CHAR
7	blank	CHAR
8-13	PROJNO	CHAR
14	blank	CHAR
15-21	EMPTIME	CHAR

DATAUNLOAD User-Specified Output Record Format Example: Figure 96 selects data for the columns EMPNO, PROJNO, and EMPTIME in the table EMP_ACT, and data for the column JOB in the EMPLOYEE table based on the selection criteria specified in the WHERE clause. Only data for the columns EMPNO, PROJNO, and EMPTIME is unloaded because JOB does not have a DFI subcommand. The output records are generated in EMPNO sequence.

The Database Services Utility DATAUNLOAD command sequence is:

```
DATAUNLOAD
SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME,JOB
FROM EMP_ACT,EMPLOYEE
WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
ORDER BY EMP_ACT.EMPNO;
EMPNO        1-6
PROJNO       8-13
EMPTIME      15-21    DECIMAL    IF NULL SET POS(22) = '?'
OUTFILE(OUTPUT1)
```

Figure 96. DATAUNLOAD Command with DFI Subcommands

The Database Services Utility report or message file output generated as a result of these commands is shown in the following:

```
ARI0801I DBS Utility started: 10/05/89 14:54:41.
         AUTOCOMMIT = OFF ERRORMODE = OFF
         ISOLATION LEVEL = REPEATABLE READ
------> CONNECT "SQLDBA " IDENTIFIED BY ********;
ARI8004I User SQLDBA connected to database SQLDBA.
ARI0500I SQL processing was successful.
ARI0505I SQLCODE = 0 SQLSTATE = 00000 ROWCOUNT = 0
------>
ARI8003I ...Extended DBCS (DBCS=YES) processing was in effect.
------> DATAUNLOAD
------> SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME
------> FROM EMP_ACT,EMPLOYEE
------> WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
------> ORDER BY EMP_ACT.EMPNO;
------>  EMPNO 1-6
------>  PROJNO 8-13
------>  EMPTIME 15-21    DECIMAL      IF NULL SET POS(22) = '?'
------> OUTFILE(OUTPUT1)
ARI0831I Column JOB data will not be unloaded.
ARI0868I DNAME=OUTPUT1 RECFM=F RECSZ=80 BLKSIZE=80
ARI0835I 74 record(s) written to the output data file.
ARI0855I DATAUNLOAD processing successful.
ARI0802I End of command file input.
ARI8997I ...Begin COMMIT processing.
ARI0811I ...COMMIT of any database changes sucessful.
ARI0809I ...No error(s) occurred during command processing.
ARI0808I DBS processing completed: 10/05/89 14:54:44.
```

Figure 97. DB2 Server for VSE Database Services Utility Report Output

```
1ARI0801I DBS Utility started: 10/05/89 14:54:41.
          AUTOCOMMIT = OFF ERRORMODE = OFF
          ISOLATION LEVEL = REPEATABLE READ
 ARI8003I ...Extended DBCS (DBCS=YES) processing was in effect.
0------> DATAUNLOAD
 ------> SELECT EMP_ACT.EMPNO,PROJNO,EMPTIME,JOB
 ------> FROM EMP_ACT,EMPLOYEE
 ------> WHERE EMP_ACT.EMPNO=EMPLOYEE.EMPNO
 ------> ORDER BY EMP_ACT.EMPNO;
 ------>  EMPNO 1-6
 ------>  PROJNO 8-13
 ------>  EMPTIME 15-21    DECIMAL   IF NULL SET POS(22) = '?'
 ------> OUTFILE(OUTPUT1)
 ARI0852I DATAUNLOAD processing started.
 ARI0831I Column JOB data will not be unloaded.
 ARI0868I DNAME=OUTPUT1 RECFM=F RECSZ=80 BLKSIZE=80      <------  See Note
 ARI0835I 74 record(s) written to the output data file.
 ARI0855I DATAUNLOAD processing successful.
 ARI0802I End of command file input.
 ARI8997I ...Begin COMMIT processing.
 ARI0811I ...COMMIT of any database changes sucessful.
 ARI0809I ...No error(s) occurred during command processing.
 ARI0808I DBS processing completed: 10/05/89 14:54:44.
```

Figure 98. DB2 Server for VM Database Services Utility Message File Output

Note: The RECFM, RECSZ, and BLKSIZE information displayed in the message ARI0868I depends on the CMS FILEDEF command specifications for the output file with *ddname*=OUTPUT1.

The format of the records in the output file identified by the ddname OUTPUT1 is shown in Table 12.

Table 12. User-Defined Output Record Format

Record Position	**Data Value Source (Column or Other)**	**Output Record Field Data Type**
1-6	EMPNO	CHAR
7	blank	CHAR
8-13	PROJNO	CHAR
14	blank	CHAR
15-21	EMPTIME	DECIMAL
22	EMPTIME null indicator	CHAR

DATAUNLOAD Data Conversion Summary

Table 13 summarizes the data conversion performed by Database Services Utility DATAUNLOAD processing. Yes means that the utility performs the conversion. No means that the utility cannot convert the source column data type into the data type specified for the output record data field and that any attempt to do so results in a Database Services Utility processing error. The numbers in the chart refer to the notes below the figure.

Table 13. DATAUNLOAD Data Conversion Table

	Source Column Data Type							
Output Field Data Type	**CHAR, VARCHAR, or LONG VARCHAR**	**DECIMAL[8]**	**SMALLINT**	**INTEGER**	**REAL[12]**	**Double Precision[13]**	**DATE, TIME, or TIMESTAMP**	**DBCS, GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC**
CHAR	Yes[1]	Yes[2,3]	Yes[2,4]	Yes[2,4]	Yes[2,5]	Yes[2,5]	Yes[1]	Yes
GRAPHIC (G)	No	No	No	No	No	No	No	Yes[9]
1-Byte FIXED	No	No	Yes[6]	Yes[6]	No	No	No	No
2-Byte FIXED	No	No	Yes[6]	Yes[6]	No	No	No	No
4-Byte FIXED	No	No	Yes[6]	Yes[6]	No	No	No	No
4-Byte FLOAT	No	No	No	No	Yes	Yes[7]	No	No
8-Byte FLOAT	No	No	No	No	Yes[11]	Yes	No	No
DECIMAL	No	Yes[14]	No	No	No	No	No	No
ZONED	No	Yes	Yes[6]	Yes[6]	No	No	No	No
DATE, TIME, or TIMESTAMP	No	No	No	No	No	No	Yes[10]	No

Notes for Table 13:

1. The CHAR, VARCHAR, and long field column data may be truncated if the length of the source data is greater than the length of an output CHAR data field. For TIME and DATE, an error occurs if the length of the source data is greater than the length of an output CHAR data field. For TIMESTAMP, if the output data field length is less than 19 bytes, an error occurs; if the output data field length is less than 26 bytes but greater than or equal to 19 bytes,

trailing digits of the microseconds part of the timestamp is truncated. If the length of the output record data field is greater than the length of the source column data, all trailing (low-order) positions of the data field are padded with a blank (hex 40) value. Occurrences of null character column data result in an all blank output record data field.

2. If a CHAR output record data field is potentially too small to contain all significant digits, the sign, and the decimal point for a value derived from a column defined with a numeric (SMALLINT, INTEGER, DECIMAL, or FLOAT) data type, then:
   - An error message is written to the Database Services Utility message file identifying the column name.
   - The output record CHAR data field associated with a numeric column contains asterisks (*) if a data overflow condition actually occurs.
3. A CHAR output data field derived from a column with a DECIMAL data type contains an EBCDIC plus sign (hex 4E) or minus sign (hex 60) in the leading (high-order) position. The data value is right-justified in the low-order positions of the output data field and represented using the values hex F0 through hex F9 in each position except for the decimal point position. A decimal point (hex 4B) precedes the low-order scale value positions in the output data field. The leading (high-order) positions of the output data field (except for the first position) contain zeros (hex F0) if the number of significant positions of the data value is less than the length of the output data field minus 2.

   Occurrences of null column data result in an unsigned output data field value of `0`. The leading position of the field contains a blank (hex 40) and the remainder of the data field contains the value hex F0 (except for the decimal point position).

   For example, the hexadecimal values in each position of an eight-position CHAR output record data field derived from a DECIMAL (5,2) column containing the value +11.11 are:

Hexadecimal Value	4E	F0	F0	F1	F1	4B	F1	F1
EBCDIC Character	+	0	0	1	1	.	1	1
Field Position	1	2	3	4	5	6	7	8

   Note: The minimum length of a CHAR output data field derived from a DECIMAL select-list column is the column scale length plus 2.

4. A CHAR output data field derived from a column with a SMALLINT or INTEGER data type contains a minus sign (hex 60) in the leading (high-order) position for negative column values. If the column value is positive, the leading (high-order) position of the data field contains a blank (hex 40). The data value is right-justified in the low-order positions of the output data field and represented using the values hex F0 through hex F9 in each position. The leading (high-order) positions of the output data field (except for the first position) contain zeros (hex F0) if the number of significant positions of the data value is less than the length of the output data field minus 1.

   Occurrences of null column data result in an unsigned output data field value of `0`. The leading position of the field contains a blank (hex 40) and the remainder of the data field positions contains the value hex F0.

   For example, the hexadecimal values contained in each position of an eight-position CHAR output record data field derived from a SMALLINT column containing the value +32767 are:

Hexadecimal Value	40	F0	F0	F3	F2	F7	F6	F7
EBCDIC Character		0	0	3	2	7	6	7
Field Position	1	2	3	4	5	6	7	8

Note: The minimum length of a CHAR output data field derived from a SMALLINT or INTEGER column is 2.

5. A CHAR output data field derived from a column with a FLOAT data type is left-justified in the leading (high-order) positions of the output data field. The format of the output data field value ranges from:

   `sn.nEsn to sn.nnnnnnnnnnnnnEsnn (for 8-byte float)`

   or

   `sn.nEsn to sn.nnnnnEsnn (for 4-byte float)`

   where:

```
s = EBCDIC sign:  Plus (+) sign (hex 4E)
                  Minus (-) sign (hex 60)
n = EBCDIC numeric character (hex F0–F9)
. = EBCDIC decimal point (hex 4B)

E = EBCDIC character E (hex C5)
```

   The trailing (low-order) positions of the output data field contain blanks (hex 40) if the length of the field is greater than the EBCDIC representation of the column value.

   Occurrences of null column data result in an unsigned output data field value (ƀ0.0E+0). The leading position of the data field contains a blank (hex 40).

   For example, the hexadecimal values contained in each position of a 10-position CHAR output record data field derived from an 8-byte FLOAT column containing the value +1.11E+02 are:

Hexadecimal Value	4E	F1	4B	F1	F1	C5	4E	F0	F2	40
EBCDIC Character	+	1	.	1	1	E	+	0	2	
Field Position	1	2	3	4	5	6	7	8	9	10

Note: The minimum length of a CHAR output data field derived from a column with FLOAT data type is 7.

6. Leading (high-order) zero-value positions of column data are truncated if the length of a numeric (FIXED, DECIMAL, or ZONED) output data field is less than the length of the numeric (SMALLINT, INTEGER, or DECIMAL) column value. A null column value results in a numeric output record data field value of 0.

   If a numeric output record data field is too small to contain all significant digits of the data value from a numeric column:

   - The message ARI0833E is written to the Database Services Utility message file identifying the column name and the SQL SELECT row count.
   - No data for the row is written to the output data file.

7. A null column value results in a numeric output record data field value of 0.

   An 8-byte float column unloaded into a 4-byte output record FLOAT data field results in the loss of a number of digits of precision. The fraction (mantissa) is reduced from 14 to 6 digits of precision. During the conversion process, if the exponent value exceeds a value of +63, the message ARI0833E is generated and no data for the row is unloaded.

8. The minimum length of an output data field derived from a DECIMAL column is the (scale value/2+1). The scale of output data derived from a DECIMAL column is the same as that for the source column.

   If the length of the output data field is greater than the value of (column precision/2)+1, the column value is extended with high-order zeros in the output data field.

   If the length of the output data field is less than the value of (column precision/2)+1, nonsignificant high-order zeros to the left of the implied decimal point are not reflected in the data field value.

9. Only fixed-length DBCS (data type of GRAPHIC) output data fields are produced by DATAUNLOAD processing. The *startpos* of the output data field contains a shift-out delimiter (hex 0E). The *endpos* of the output data field contains a shift-in delimiter (hex 0F).

   DBCS column data is truncated if the length of the column data plus 2 is greater than the length of the output GRAPHIC data field. The length of the column data is the number of DBCS characters times 2.

   If the length of the output data field minus 2 is greater than the column data length, the output data field is padded with trailing (low-order) DBCS blank (hex 4040) characters in the unused low-order data field positions. The last position of the data field always contains a shift-in delimiter (hex 0F).

   If the column data value is all blanks or null, *startpos*+1 to *endpos*−1 of the output DBCS data field contains DBCS blank characters (hex 4040).

10. The datetime data type input and target type must match; for example, the input data type of TIME is valid only for the target column data type of TIME.

11. A 4-byte float column unloaded into an 8-byte float data field is padded with hex `0000 0000`.

12. REAL represents single precision floating point data and is synonymous with FLOAT(*n*) where 1 is less than or equal to *n* is less than or equal to 21.

13. DOUBLE PRECISION represents double precision floating point data and is synonymous with FLOAT or FLOAT(*n*) where 22 is less than or equal to *n* is less than or equal to 53.

14. If the length of the output record data field is greater than the length of the source column data, all leading (high-order) positions of the data field are padded with hex zeros.

RELOAD DBSPACE

RELOAD DBSPACE Format

VM Format:

```
►►──RELOAD DBSPACE──(dbspace_name)──┬─NEW───┬──INFILE──(ddname)──────────────►
                                    └─PURGE─┘

►──┬─────────────────────────────┬──┬──────────────────────────────────┬──────►
   └─COMMITCOUNT──(──ccount──)───┘  └─RESTARTTABLE──(──table_name──)───┘

►──┬──────────────────────────────┬───────────────────────────────────────────►◄
   └─RESTARTCOUNT──(──rcount──)───┘
```

VSE Format:

```
►►──RELOAD DBSPACE──(dbspace_name)──┬─NEW───┬─────────────────────────────────►
                                    └─PURGE─┘

►─INFILE──(──ddname──┬────────────────────────────────┬───────────────────────►
                     │                ┌─2048─┐        │
                     └─BLKSZ──(───────┴─size─┴───)────┘

►──┬─────────────────────────────────────────┬─)──┬─────────────────────────────┬──►
   │              ┌─REWIND────┐              │    └─COMMITCOUNT──(──ccount──)───┘
   │       ┌─(TAPE)──┴─NOREWIND──┴─┐         │
   └─PDEV──┴─(DASD)────────────────┴─────────┘

►──┬──────────────────────────────────┬──┬──────────────────────────────┬──────►◄
   └─RESTARTTABLE──(──table_name──)───┘  └─RESTARTCOUNT──(──rcount──)───┘
```

Examples:

```
RELOAD DBSPACE(JOHNS.SPACE1) PURGE INFILE(TEMP)
RELOAD DBSPACE(PUBLIC.SPACE2) NEW INFILE(TEMP)
RELOAD DBSPACE(DBS1) PURGE INFILE(IFILE) COMMITCOUNT(300)
   RESTARTTABLE(EMPLOYEE) RESTARTCOUNT(600)
```

Authorization:

You must have the **INSERT** privilege on the tables affected by the command. Additional authority is required depending on the keywords specified:
  RESOURCE–if NEW is specified.
  SELECT, **DELETE**, and **INSERT**– if PURGE is specified.
  DBA–if PURGE is specified, and if any indexes defined on an affected table are owned by someone else. DBA authority is also required if NEW is specified, and any tables are to be created for another user.

Before you reload tables into a dbspace, it must already exist.

Note: The RELOAD DBSPACE command is not supported if you are using DRDA flow.

DBSPACE (*dbspace-name*)
identifies a RELOAD DBSPACE request and identifies the dbspace to be loaded. The Database Services Utility loads the tables into the dbspace in the order that they occur in the input data. If you do not own a private dbspace with the *dbspace-name* identified, the data is loaded into a public dbspace (if one having that name exists). The owner of a public dbspace is PUBLIC.

NEW
instructs the Database Services Utility to create each table contained in the input file before loading the data. Tables represented in the input data file that already exist in the database are not processed. The tables are created for the current Database Services Utility user. You must have RESOURCE authorization to use this keyword.

If either the RESTARTCOUNT or RESTARTTABLE parameters appear on the RELOAD DBSPACE command, the NEW parameter will not cause the restart table to be created. The RESTARTCOUNT and RESTARTTABLE parameters indicate that the RELOAD DBSPACE operation is being restarted, therefore, NEW processing must have already occurred, so it is not required to create the restart table again. Note that NEW processing is performed on all tables to be reloaded before any rows are reloaded to any table. This means that NEW processing will have already occurred for all tables to be reloaded.

PURGE
instructs the Database Services Utility that existing tables within the dbspace are to be loaded. The rows for all dbspace tables to be processed are deleted before the first table is loaded. The tables that are processed are those that are in the input file; that is, if table JONES.PROJECT exists in the dbspace, but the input file contains only JONES.EMPLOYEE and JONES.DEPARTMENT, then JONES.PROJECT is unaffected by RELOAD processing. Even if the input file contains SMITH.PROJECT, JONES.PROJECT is unaffected. The Database Services Utility uses fully qualified table names when determining the tables to reload. You must have the DELETE privilege to use this keyword if you do not own the affected tables. You must also have DBA authority if any indexes defined on an affected table are owned by someone else.

If either the RESTARTCOUNT or RESTARTTABLE parameters appear on the RELOAD DBSPACE command, the PURGE parameter will not cause all rows of the restart table to be deleted. The RESTARTCOUNT and RESTARTTABLE parameters indicate that the RELOAD DBSPACE operation is being restarted, therefore, PURGE processing must have already occurred, so it is not required to delete all rows from the restart table again. Note that PURGE processing is performed on all tables to be reloaded before any rows are reloaded to any table. This means that PURGE processing will have already occurred for all tables to be reloaded.

Note: You must specify either NEW or PURGE in the RELOAD DBSPACE statement. Because existing tables might be greatly affected by the choice of these parameters, there is no default specification.

INFILE (*ddname*)
in DB2 Server for VSE, this identifies and describes the sequential (SAM) file containing the input dbspace data. The default record format in a DB2 Server for VSE system is variable-length blocked, spanned (SB), with LRECL=(BLKSIZE−4) for variable and spanned records or LRECL=BLKSIZE for fixed and undefined records.

The default record format in a DB2 Server for VM system is variable-length blocked, spanned (VBS). Block size and record format information is specified using a CMS FILEDEF command; the LRECL parameter is not applicable.

Note: The RECFM, RECSZ, and BLKSIZE information displayed in the message ARI0868I depends on the CMS FILEDEF command specifications for the RELOAD input file. However, RELOAD processing in a DB2 Server for VM environment always opens the input data file as a VB file. The RECFM, RECSZ, and BLKSIZE information displayed in the message ARI0868I indicates this change.

ddname

In DB2 Server for VSE: this is the TLBL or DLBL job control statement file name for the sequential input file.

In DB2 Server for VM: this is the name of the sequential input file defined with a CMS FILEDEF command. Except for the *ddname*, use the same CMS FILEDEF command information for RELOAD command processing that you used when UNLOAD command processing created the file. Define the CMS file used for RELOAD command input with the file-mode number **4**. Do not specify SYSIN or SYSPRINT as the *ddname*.

COMMITCOUNT (*ccount***)**

identifies the frequency of COMMIT action during RELOAD processing. *ccount* is a number from 1 to 2,147,483,647 indicating that a COMMIT statement should be executed after the number of input table rows equal to *ccount* are processed by RELOAD for each table. A COMMIT statement will also be executed after the last row of each table has been reloaded.

Note: Database Services Utility AUTOCOMMIT ON processing must be in effect when you use RELOAD COMMITCOUNT processing. If AUTOCOMMIT is OFF and the COMMITCOUNT parameter is used, an error message is written and RELOAD command processing is not performed.

RESTARTTABLE (*table_name***)**

identifies at which table the RELOAD DBSPACE processing will be restarted. If a RELOAD DBSPACE operation ended normally, and the RELOAD DBSPACE statement included the COMMITCOUNT parameter, the RELOAD DBSPACE operation can be restarted by using the RESTARTTABLE and RESTARTCOUNT parameters. *table_name* identifies the table where RELOAD processing should begin. If RESTARTTABLE is omitted, RELOAD DBSPACE processing will begin reloading the first table, and the RESTARTCOUNT parameter, if specified, will apply to the first table.

Note: If the table does not exist in the database when RELOAD DBSPACE with RESTARTCOUNT or RESTARTTABLE is issued, an error message is displayed.

RESTARTCOUNT (*rcount***)**

identifies the restart point for RELOAD processing. *rcount* is a number from 1 to 2,147,483,647 that identifies the number of input table rows in the restart table to be skipped before RELOAD command processing begins. If RESTARTCOUNT is omitted, no table rows are skipped and RELOAD processing begins with the first table row of the restart table.

Note: If an end-of-table condition occurs before *rcount* rows of the restart table are read, an error message is written before RELOAD processing ends.

BLKSZ (*size*) (DB2 Server for VSE Only)
is an optional parameter that specifies the block size of the sequential output file. The default block size is 2048 bytes per block.

PDEV (TAPE or DASD)
is an optional parameter that specifies the device type (DASD or TAPE) of the sequential (SAM) input file. Specify PDEV(DASD) if the input file resides on any device supported by the VSE DTFSD macro. An exception to this is VSAM-managed SAM files. VSAM-managed SAM does not support spanned records. Specify PDEV(TAPE) if the input file resides on a device supported by the VSE DTFMT macro. The default is PDEV(TAPE).

BLKSZ and PDEV can be specified in any order but must occur after the *ddname* parameter.

REWIND or NOREWIND
controls tape file rewind processing performed during OPEN processing. This parameter is valid only if you specify TAPE for PDEV. The default processing is REWIND.

REWIND
specifies that the tape file is rewound by OPEN processing.

NOREWIND
specifies that the tape file is not rewound by OPEN processing. If NOREWIND is specified for input tape files referenced by a series of RELOAD commands, you must ensure that the tape files being referenced are in ascending sequence. For example, if NOREWIND is specified in a sequence of two RELOAD commands and the first command reads tape file 2, then the second command must reference tape file 3 or higher number. If it references tape file 1, an OPEN error occurs.

Selective dbspace file reloads may be performed by specifying the file sequence number in the TLBL statement. The following JCL is an example of how this can be accomplished:

```
// JOB RELOAD DBSPACE
// ASSGN SYS005,181
// MTC REW,SYS005
* RELOAD 1 DBSPACE FROM TAPE FILE SEQ# 1
// TLBL DBSP,'PUBLIC.CRP01',,,1
// EXEC ARIDBS,SIZE=AUTO
RELOAD DBSPACE (PUBLIC.CRP01)
INFILE (DBSP BLKSZ(24720) PDEV(TAPE) );
/*
* RELOAD 2 DBSPACES FROM TAPE FILE SEQ# 2
// TLBL DBSP,'PUBLIC.CRPXX',,,2
// EXEC ARIDBS,SIZE=AUTO
RELOAD DBSPACE (PUBLIC.CRP02)
INFILE (DBSP BLKSZ(24720) PDEV(TAPE) );
RELOAD DBSPACE (PUBLIC.CRP03)
INFILE (DBSP BLKSZ(24720) PDEV(TAPE) );
/*
/&
```

Release Coexistence Considerations for DB2 Server for VM

Changes were required in Version 7 Release 1 to handle file I/O correctly when using CMS 15 and later. These changes affect the format of data that is unloaded and reloaded by the UNLOAD and RELOAD commands of the DBS Utility. If you use the DBS Utility's UNLOAD and RELOAD commands with databases at different release levels, you must ensure that the code changes have been applied at all release levels. For releases prior to Version 7 Release 1, you must apply the following APARs:

Release	APAR
3.5	PQ28584
5.1	PQ28583
6.1	PQ27957

RELOAD TABLE

RELOAD TABLE Format

VSE Format:

```
►►──RELOAD TABLE──(table_name)──┬─PURGE─────────────────┬──────────────►
                                └─NEW──(dbspace_name)───┘

►──┬──────────────────────────┬────────────────────────────────────────►
   └─INTABLE──(table_name)────┘

►─INFILE──(──ddname──┬────────────────────────────┬────────────────────►
                     │            ┌─2048────┐     │
                     └─BLKSZ──(───┴──┬──────┴──)──┘
                                     └─size─┘

►──┬──────────────────────────────────┬─)──┬─────────────────────────────┬──►
   │                 ┌─REWIND────┐    │    └─COMMITCOUNT──(──ccount──)───┘
   │        ┌─(TAPE)─┴─NOREWIND──┤    │
   └─PDEV───┴─(DASD)─────────────┴────┘

►──┬─────────────────────────────────┬─────────────────────────────────►◄
   └─RESTARTCOUNT──(──rcount──)──────┘
```

VM Format:

```
►►──RELOAD TABLE──(table_name)──┬─PURGE─────────────────┬──────────────►
                                └─NEW──(dbspace_name)───┘

►──┬──────────────────────────┬──INFILE──(ddname)──────────────────────►
   └─INTABLE──(table_name)────┘

►──┬─────────────────────────────┬──┬──────────────────────────────┬──►◄
   └─COMMITCOUNT──(──ccount──)───┘  └─RESTARTCOUNT──(──rcount──)───┘
```

VSE Examples:

```
RELOAD TABLE(SALARY)
       NEW(DBSPACE1)
       INTABLE(SMITH.SALARY)
       INFILE(CIPHER3 PDEV(TAPE))
RELOAD TABLE(SALARY)
       NEW(DBSPACE1)
       INTABLE(SMITH.SALARY)
       INFILE(CIPHER3)
       COMMITCOUNT(300)
       RESTARTCOUNT(600)
```

VM Example:

```
RELOAD TABLE(SALARY)
       NEW(DBSPACE1)
       INTABLE(SMITH.SALARY)
       INFILE(CIPHER3)
       COMMITCOUNT(300)
       RESTARTCOUNT(600)
```

Authorization:

```
You must have the INSERT privilege on the "target" table.
Additional authority is required depending on the keywords specified.
    RESOURCE—if NEW is specified.
    DELETE and INSERT—if PURGE is specified and the table is owned by another
    user.
    DBA—if PURGE is specified, and if any indexes defined on an affected table
    are owned by someone else.
```

Note: The RELOAD TABLE command is not supported if you are using DRDA flow.

TABLE (*table_name***)**
identifies a RELOAD TABLE request and the table to be loaded. You can further identify the table by specifying the *owner* of the table (see "Qualifying Object Names" on page 110 for details). You cannot use a synonym for a *table_name*. If you specify the NEW option, a table called *table_name* is created for that user. If you specify the PURGE option, you can specify a view name instead of a table name if the view meets the following requirements:

- The view is defined on a single table.
- The view definition includes all the NOT NULL columns in the table. That is, all columns outside of the view definition must permit the insertion of nulls.
- The view has no column definitions based on functions (virtual data columns).

When reloading data into a view that was created using the WITH CHECK OPTION clause, the database manager checks all inserts and updates to the view against the view definition and rejects them if the row to be inserted or updated does not conform to the view definition.

NEW (*dbspace_name***)**
instructs the Database Services Utility that the table to be loaded does not exist and must first be created. You can identify the dbspace by the *owner*. If you do not specify the owner of the dbspace (see "Qualifying Object Names" on page 110 for information about *owner*), a private dbspace that you own with *dbspace_name* specified is loaded. If no such private dbspace exists, a public dbspace with *dbspace_name* is loaded. The owner of a public dbspace is PUBLIC, for example, NEW (PUBLIC.PRODUCTION). If *owner* is specified for the table name and *owner* is not specified for the dbspace name, the Database Services Utility does not use the *owner* specified for the table name to identify the private dbspace.

If the RESTARTCOUNT parameter appears on the RELOAD TABLE command, the NEW parameter will not cause the table to be created. The RESTARTCOUNT parameter indicates that the RELOAD TABLE operation is being restarted, therefore, NEW processing must have already occurred, so it is not required to create the table again.

PURGE
identifies that the output table (table to be loaded) exists and that all existing table rows should be deleted by RELOAD TABLE processing before loading. You must have the DELETE privilege on the output table. If you are not the owner of the output table, you require DELETE and INSERT authority for the table. If any indexes for the table are owned by another user, you require DBA authority.

If the RESTARTCOUNT parameter appears on the RELOAD TABLE command, the PURGE parameter will not cause all row to be deleted. The RESTARTCOUNT parameter indicates that the RELOAD TABLE operation is being restarted, therefore, PURGE processing must have already occurred, so it is not required to delete all rows from the table again.

Note: You must specify either NEW or PURGE in the RELOAD TABLE statement. Because existing tables might be greatly affected by the choice of these parameters, there is no default specification.

INTABLE (*table_name*)
is optional. If omitted, the Database Services Utility loads data from the first table it finds in the input file. INTABLE identifies data in the input file to be used for RELOAD TABLE processing. Because the input file must be created by UNLOAD processing, the data is organized by the tables from which it was unloaded. Thus, the *table_name* that you specify here is the name of a table that was unloaded at an earlier time. This parameter is useful if your input file was created by an UNLOAD DBSPACE command. The UNLOAD DBSPACE command can unload many tables into a sequential file. The INTABLE parameter merely identifies which of those tables you now want to reload.

You can use *owner* to specify the user ID of the person who created the table in the input file. If you omit the owner (see "Qualifying Object Names" on page 110 for more information about *owner*), the utility uses the data of the first table encountered in the input file with the *table_name* specified. In this instance, *owner* does not default to the user ID of the current Database Services Utility user.

INFILE (*ddname*)
identifies and describes the sequential (SAM) input file containing the data to be loaded into the table. The file must be created with UNLOAD processing.

The default record format in a DB2 Server for VSE system is variable-length blocked, spanned (SB), with LRECL=(BLKSIZE−4) for variable and spanned records or LRECL=BLKSIZE for fixed and undefined records.

The default record format in a DB2 Server for VM system is variable-length blocked, spanned (VBS), with block size and record format information specified by a CMS FILEDEF command; the LRECL parameter is not applicable.

Note: The RECFM, RECSZ and BLKSIZE information displayed in the message ARI0868I depends on the CMS FILEDEF command specifications for the RELOAD input file. However, RELOAD processing in a DB2 Server for VM environment always opens the input data file as a VB file. The RECFM, RECSZ and BLKSIZE information displayed in the message ARI0868I indicates this change.

COMMITCOUNT (*ccount*)
identifies the frequency of COMMIT action during RELOAD processing. *ccount*

is a number from 1 to 2,147,483,647 indicating that a COMMIT statement should be executed after the number of input table rows equal to *ccount* are processed by RELOAD TABLE.

Note: Database Services Utility AUTOCOMMIT ON processing must be in effect when you use RELOAD COMMITCOUNT processing. If AUTOCOMMIT is OFF and the COMMITCOUNT parameter is used, an error message is written and RELOAD command processing is not performed.

RESTARTCOUNT (*rcount***)**
identifies the restart point for RELOAD processing. *rcount* is a number from 1 to 2,147,483,647 that identifies the number of input table rows to be skipped before RELOAD command processing begins. Row *rcount + 1* will be the first row to be reloaded. If RESTARTCOUNT is omitted, no rows are skipped and RELOAD processing begins with the first input row.

Note: If an end-of-table condition occurs before *rcount* rows are read from the input UNLOAD file, an error message is written before RELOAD processing ends.

ddname
in DB2 Server for VSE: this is the TLBL or DLBL job control statement file name for the sequential input file.

In DB2 Server for VM: this is the name of the sequential input file defined with a CMS FILEDEF command. Except for the ddname, CMS FILEDEF command information for RELOAD command processing should be identical to the information in the FILEDEF command used when the file was created by UNLOAD command processing. You must define a CMS file used for RELOAD command input with the file-mode number **4**. Do not specify SYSIN or SYSPRINT as the *ddname*.

BLKSZ (*size***)**
is an optional parameter that specifies the block size of the sequential output file. The default block size is 2048 bytes per block.

PDEV (TAPE or DASD)
is an optional parameter that specifies the device type (DASD or TAPE) of the sequential (SAM) input file. Specify PDEV(DASD) if the input file resides on any device supported by the VSE DTFSD macro. An exception to this is VSAM-managed SAM files. VSAM-managed SAM does not support spanned records. Specify PDEV(TAPE) if the input file resides on a device supported by the VSE DTFMT macro. The default is PDEV(TAPE).

BLKSZ and PDEV can be specified in any order but must occur after the ddname parameter.

REWIND or NOREWIND
controls tape file rewind processing performed during OPEN processing. This parameter is valid only if you specify TAPE for PDEV. The default processing is REWIND.

REWIND
specifies that the tape file is rewound by OPEN processing.

NOREWIND
specifies that the tape file is not rewound by OPEN processing. If NOREWIND is specified for input tape files referenced by a series of RELOAD commands, you must ensure that the tape files being

referenced are in ascending sequence. For example, if NOREWIND is specified in a sequence of two RELOAD commands and the first command reads tape file 2, then the second command must reference tape file 3 or higher. If it references tape file 1, an OPEN error occurs.

Release Coexistence Considerations for DB2 Server for VM

Changes were required in Version 7 Release 1 to handle file I/O correctly when using CMS 15 and later. These changes affect the format of data that is unloaded and reloaded by the UNLOAD and RELOAD commands of the DBS Utility. If you use the DBS Utility's UNLOAD and RELOAD commands with databases at different release levels, you must ensure that the code changes have been applied at all release levels. For releases prior to Version 7 Release 1, you must apply the following APARs:

Release	**APAR**
3.5	PQ28584
5.1	PQ28583
6.1	PQ27957

UNLOAD DBSPACE

UNLOAD DBSPACE Format

VM Format:

```
►►──UNLOAD DBSPACE──(dbspace_name)──OUTFILE──(ddname)────────────────────►◄
```

VSE Format:

```
►►──UNLOAD DBSPACE──(dbspace_name)────────────────────────────────────────►

►─OUTFILE──(──ddname──┬──────────────────────────────┬────────────────────►
                      │              ┌─2048─┐         │
                      └─BLKSZ──(─────┴─size─┴───)─────┘

►──┬──────────────────────────────────────────┬─)─────────────────────────►◄
   │                       ┌─NOREWIND─┐       │
   │          ┌─(TAPE)─────┴─REWIND───┴──┐    │
   └─PDEV─────┴─(DASD)───────────────────┴────┘
```

VM Example:

```
UNLOAD DBSPACE(THOMPSON.SPACE1) OUTFILE(HISTORY)
```

VSE Example:

```
UNLOAD DBSPACE(MIKE.SP2) OUTFILE(SAVE BLKSZ(2048))
```

Authorization:

You must have the **SELECT** privilege on the table(s) being unloaded.

UNLOAD DBSPACE unloads all tables of the specified dbspace to a sequential output file.

Note: The UNLOAD DBSPACE command is not supported if you are using DRDA flow.

Following are the descriptions for each portion of the command:

DBSPACE (*dbspace_name*)
identifies an UNLOAD DBSPACE request and the dbspace to be unloaded. The utility unloads the tables of the dbspace in an unpredictable order. The *dbspace_name* is the name of the dbspace to be unloaded. If you do not specify *owner*, the utility unloads one of your dbspaces. See "Names and Identifiers" on page 110 for more information about naming conventions for data objects. If you do not own a dbspace called *dbspace_name*, the utility unloads a public dbspace having that name. If there is no public dbspace having that name, UNLOAD processing is unsuccessful, and an error message is written to the Database Services Utility message file.

For example, suppose your user ID is GENE and you specify:

```
UNLOAD DBSPACE(SPACE1) ...
```

The Database Services Utility unloads the private dbspace named GENE.SPACE1. If there is no such dbspace, the utility unloads the public dbspace named PUBLIC.SPACE1. If there is no PUBLIC.SPACE1, no dbspace is unloaded, and you receive an error message in the message file.

If you own a private dbspace with the same name as a public dbspace, and you want to unload the public dbspace, you must specify PUBLIC.*dbspace_name*. If *owner* is omitted, the private dbspace is unloaded.

OUTFILE | OUTFILE (*ddname***)**
identifies and describes the sequential (SAM) output file that is to contain the data unloaded from the dbspace. The default record format is variable-length blocked, spanned (VBS). A minimum logical record length (LRECL) of 8240 bytes is the default in DB2 Server for VSE. A block size greater than 8244 is recommended for tape output files to improve performance.

ddname

DB2 Server for VSE

This is the TLBL or DLBL job control statement file name for the sequential output file.

Note: If the message ARI0868I generated during Database Services Utility UNLOAD command processing identifies RECFM=VS for an output file defined with RECFM VBS, the file can be read by Database Services Utility RELOAD command processing using RECFM VBS.

DB2 Server for VM

This is the name of the sequential output file defined with a CMS FILEDEF command. The FILEDEF command should contain the record format specification RECFM VBS or a block size (BLOCK or BLKSIZE) value or both. You must define a CMS file used for UNLOAD command output with the file mode number **4**.

If the row length (sum of defined column lengths) for any table in the dbspace being unloaded exceeds 8 240 bytes, the largest row length value is used as the minimum logical record length.

Notes:

1. Always specify a record format (RECFM) of VBS for UNLOAD processing. UNLOAD processing changes the record format to U if the system-required logical record length is greater than the specified block size (BLOCK) value minus 4. Otherwise, UNLOAD processing changes the record format to VB. This change will be indicated in the ARI0868I message, generated by the UNLOAD processing. See Appendix B, "FILEDEF Command Syntax and Notes," on page 249 for more information about undefined (U) record format usage.
2. If the message ARI0868I, generated using DBS UNLOAD processing, identifies RECFM=U or RECFM=VB for an output file defined with RECFM=VBS, the CMS FILEDEF command that defines the RELOAD input data files must still specify RECFM=VBS. Except for the ddname, CMS FILEDEF information for RELOAD command processing must be identical to the information in the FILEDEF command used when UNLOAD command processing created the file. See Appendix B, "FILEDEF Command Syntax and Notes," on page 249 for more information about undefined (U) record format usage.
3. If the message ARI0868I indicates RECFM=U for a tape output file, significant performance improvements can be obtained by increasing

the block size (BLOCK) value for the file. A block size greater than 8244 bytes is recommended for tape output files created by UNLOAD processing.

Do not specify SYSIN or SYSPRINT as the *ddname*.

BLKSZ (*size*) (DB2 Server for VSE Only)
is an optional parameter that specifies the block size of the sequential output file. The default block size is 2048 bytes per block.

PDEV (TAPE or DASD) (DB2 Server for VSE Only)
is an optional parameter that specifies the device type (DASD or TAPE) for the sequential output file. Specify PDEV(DASD) for files that reside on any device supported by the VSE DTFSD macro. An exception to this is VSAM-managed SAM files. VSAM-managed SAM does not support spanned records. Specify PDEV(TAPE) for files that reside on any device supported by the VSE DTFMT macro. The default is PDEV(TAPE).

BLKSZ and PDEV can be specified in any order but must occur after the ddname parameter.

NOREWIND or REWIND (DB2 Server for VSE Only)
controls tape file rewind processing performed during CLOSE processing. This parameter is valid only if you specified TAPE for PDEV. The default is NOREWIND.

NOREWIND
specifies that the tape file will not be rewound by CLOSE processing.

REWIND
specifies that the tape file is rewound by CLOSE processing.

Selective dbspace file unloads may be performed by specifying the file sequence number in the TLBL statement. The following JCL is an example of how this can be accomplished:

```
// JOB UNLOAD DBSPACE
* UNLOAD 1 DBSPACE INTO TAPE FILE SEQ# 1
// TLBL DBSP,'PUBLIC.CRP01',,,,1
// ASSGN SYS005,181
// MTC REW,SYS005
// EXEC ARIDBS,SIZE=AUTO
UNLOAD DBSPACE (PUBLIC.CRP01)
OUTFILE (DBSP BLKSZ(24720) PDEV(TAPE) );
/*
* UNLOAD 2 DBSPACES INTO TAPE FILE SEQ# 2
// TLBL DBSP,'PUBLIC.CRPXX',,,,2
// EXEC ARIDBS,SIZE=AUTO
UNLOAD DBSPACE (PUBLIC.CRP02)
OUTFILE (DBSP BLKSZ(24720) PDEV(TAPE) );
UNLOAD DBSPACE (PUBLIC.CRP03)
OUTFILE (DBSP BLKSZ(24720) PDEV(TAPE) );
/*
/&
```

Release Coexistence Considerations for DB2 Server for VM

Changes were required in Version 7 Release 1 to handle file I/O correctly when using CMS 15 and later. These changes affect the format of data that is unloaded and reloaded by the UNLOAD and RELOAD commands of the DBS Utility. If you use the DBS Utility's UNLOAD and RELOAD commands with databases at different release levels, you must ensure that the code changes have been applied at all release levels. For releases prior to Version 7 Release 1, you must apply the following APARs:

Release	APAR
3.5	PQ28584
5.1	PQ28583
6.1	PQ27957

UNLOAD TABLE

UNLOAD TABLE Format

VSE Format:

```
►►──UNLOAD TABLE──(table_name)─────────────────────────────────────────────►

►─OUTFILE──(──ddname──┬──────────────────────────────┬──────────────────────►
                      │                ┌─2048─┐       │
                      └─BLKSZ──(──────┴─size─┴────)──┘

►─┬─────────────────────────────────────┬─)────────────────────────────────►◄
  │                  ┌─NOREWIND─┐       │
  │        ┌─(TAPE)──┴─REWIND───┴──┐    │
  └─PDEV───┴─(DASD)────────────────┴────┘
```

VM Format:

```
►►──UNLOAD TABLE──(table_name)──OUTFILE──(ddname)──────────────────────────►◄
```

VSE Example:

```
UNLOAD TABLE (EMPLOYEE) OUTFILE(SAVE13 PDEV(DASD))
```

VM Example:

```
UNLOAD TABLE (EMPLOYEE) OUTFILE(SAVE13)
```

Authorization:

```
You must have the SELECT privilege on the table being unloaded.
```

The UNLOAD TABLE command unloads a specific table or view to an output file.

Note: The UNLOAD TABLE command is not supported if you are using DRDA flow.

Following are descriptions of each portion of the command:

TABLE (*table_name***)**
: identifies an UNLOAD TABLE request and the table to be processed. You can UNLOAD a view merely by specifying a view name instead of a table name. You can further identify the table or view by specifying the *owner* of the table or view (see "Qualifying Object Names" on page 110 for details). A synonym cannot be used for *table_name*.

OUTFILE|OUTFILE (*ddname*)
identifies and describes the sequential (SAM) output file that is to contain the data unloaded from the table. The default record format is variable-length blocked, spanned (VBS). A minimum logical record length (LRECL) of 8 240 bytes is the default in a VSE system. To improve performance, a block size greater than 8 244 bytes is recommended for tape output files.

ddname
in DB2 Server for VSE: this is the TLBL or DLBL job control statement file name for the sequential output file.

Note: If the message ARI0868I generated during Database Services Utility UNLOAD command processing identifies RECFM=VB for an output file defined with RECFM VBS, the file can be read by Database Services Utility RELOAD command processing using RECFM VBS.

In DB2 Server for VM: this is the name of the sequential output file defined with a CMS FILEDEF command. The FILEDEF command should contain the record format specification RECFM VBS or a block size (BLOCK or BLKSIZE) value or both. You must define a CMS file used for UNLOAD command output with the file mode number **4**. UNLOAD processing writes variable-length spanned records with a minimum logical record length (LRECL) of 8 240 bytes.

Notes:

1. Always specify a record format (RECFM) of VBS for UNLOAD processing. UNLOAD processing changes the record format to U if the system-required logical record length is greater than the specified block size (BLOCK) value minus 4. Otherwise, UNLOAD processing changes the record format to VB. This change will be indicated in the ARI0868I message generated by the UNLOAD processing. See Appendix B, "FILEDEF Command Syntax and Notes," on page 249 for more information about undefined (U) record format usage.
2. If the message ARI0868I, generated using DBS UNLOAD processing, identifies RECFM=U or RECFM=VB for an output file defined with RECFM=VBS, the CMS FILEDEF command that defines the RELOAD input data files must still specify RECFM=VBS. Except for the ddname, CMS FILEDEF information for RELOAD command processing must be identical to the information in the FILEDEF command used when UNLOAD command processing created the file. See Appendix B, "FILEDEF Command Syntax and Notes," on page 249 for more information about undefined (U) record format usage.
3. If the message ARI0868I indicates RECFM=U for a tape output file, significant performance improvements can be obtained by increasing the block size (BLOCK) value for the file. A block size greater than 8244 is recommended for tape output files created by UNLOAD processing.

   Do not specify SYSIN or SYSPRINT as the *ddname*.

BLKSZ (*size*)
is an optional parameter that specifies the block size of the sequential output file. The default block size is 2048 bytes per block.

PDEV (TAPE or DASD)
is an optional parameter that specifies the device type (DASD or TAPE) for the sequential output file. Specify PDEV(DASD) for files that reside on any device supported by the VSE DTFSD macro. An exception to this is VSAM-managed

SAM files. VSAM-managed SAM does not support spanned records. Specify PDEV(TAPE) for files that reside on any device supported by the VSE DTFMT macro. The default is PDEV(TAPE).

BLKSZ and PDEV can be specified in any order but must occur after the ddname parameter.

NOREWIND or REWIND
controls tape file rewind processing performed during CLOSE processing. This parameter is valid only if you specify TAPE for PDEV. The default is NOREWIND.

NOREWIND
specifies that the tape file is not rewound by CLOSE processing.

REWIND
specifies that the tape file is rewound by CLOSE processing.

Release Coexistence Considerations for DB2 Server for VM

Changes were required in Version 7 Release 1 to handle file I/O correctly when using CMS 15 and later. These changes affect the format of data that is unloaded and reloaded by the UNLOAD and RELOAD commands of the DBS Utility. If you use the DBS Utility's UNLOAD and RELOAD commands with databases at different release levels, you must ensure that the code changes have been applied at all release levels. For releases prior to Version 7 Release 1, you must apply the following APARs:

Release	APAR
3.5	PQ28584
5.1	PQ28583
6.1	PQ27957

Load-Package Commands

Processing for the Load-Package Commands

Figure 99 indicates the data flow for the UNLOAD PACKAGE and RELOAD PACKAGE commands.

Note: PROGRAM is a synonym for PACKAGE. Therefore, UNLOAD and RELOAD PROGRAM are equivalent to UNLOAD and RELOAD PACKAGE.

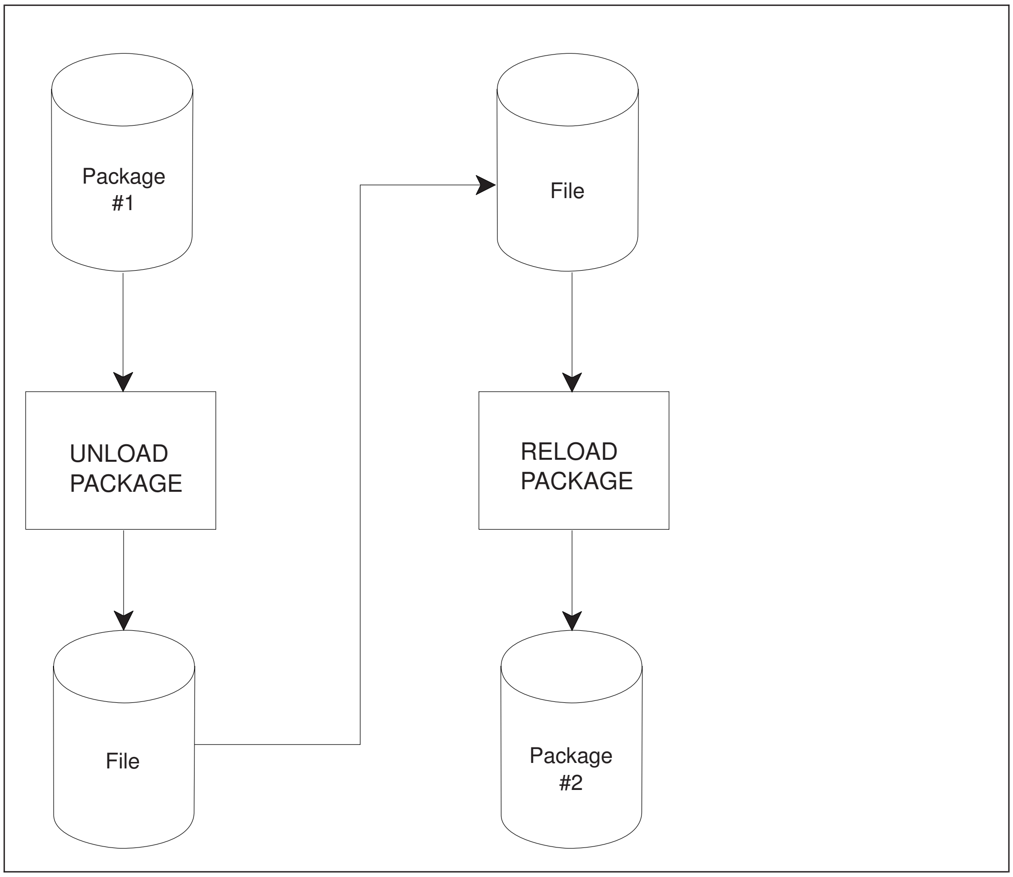
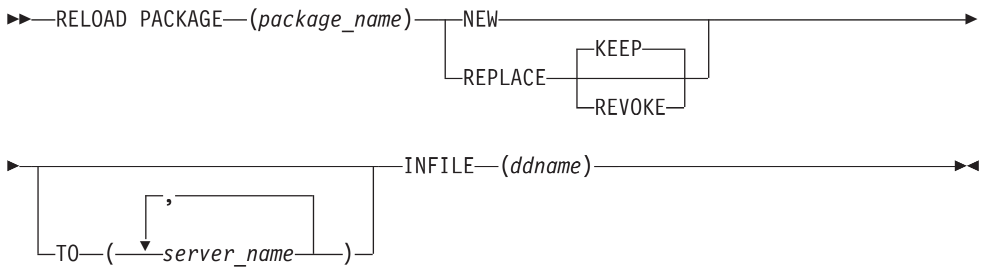


Figure 99. Database Services Utility Processing Diagram for UNLOAD and RELOAD PACKAGE

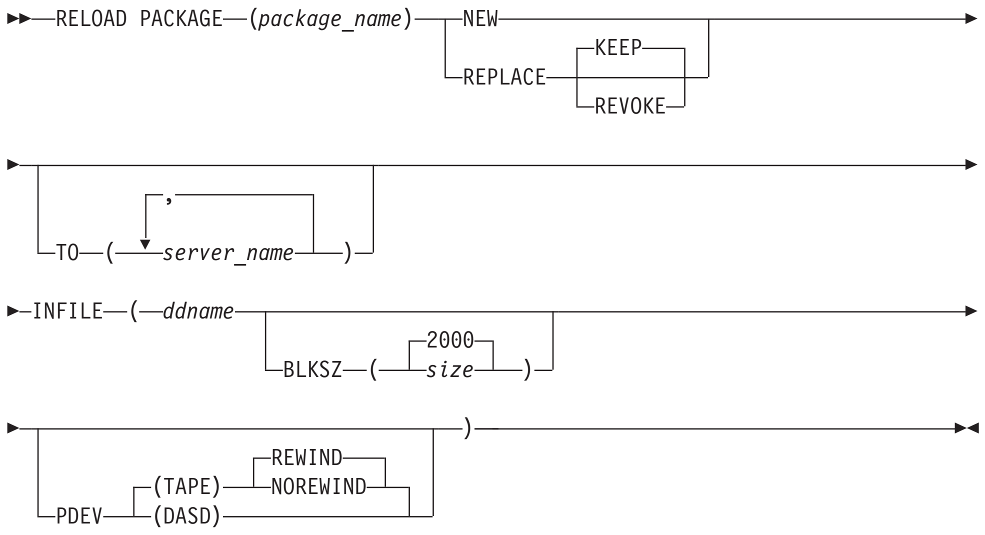
RELOAD PACKAGE

RELOAD PACKAGE Format

VM Format:



VSE Format:



VM Example:

```
RELOAD PACKAGE(JONES.PROG4) REPLACE KEEP TO(RDB2,RDB3)
INFILE(IN1)
```

VSE Example:

```
RELOAD PACKAGE(JONES.PROG4) REPLACE KEEP INFILE(IN1)
```

Authorization: You must be the owner of the package that you want to reload. To reload another user's package, you must have DBA authority. In VM, you must also have CONNECT authority to all named databases.

PACKAGE (*package_name***)**
identifies a RELOAD PACKAGE request and the package to be loaded.

You can further qualify the *package_name* with the *owner*, separating the two names with a period. The name of the package is *package_name*. If you do not specify the owner, *owner* defaults to the connected authorization ID. (See

"Qualifying Object Names" on page 110 for details on accessing data objects that are owned by other users.) The authorization ID is either:

- The authorization ID specified in a previous explicit connect (when TO is not used)
- The VM user ID (when TO is used) because it is the user ID used to connect to *server_name.*

REPLACE
is specified if an existing package is to be replaced by the reload. If the package does not exist, a new package is created without an error or warning message.

NEW
instructs the Database Services Utility that the package to be loaded does not exist and is to be created. If a package with the same name and owner already exists in the database, the reload fails.

KEEP
causes the grants of RUN privilege to remain in effect when the package is reloaded. The KEEP and REVOKE parameters apply if the package has previously been created and the owner of the package has granted the RUN privilege on the resulting package to other users. The KEEP and REVOKE parameters are allowed only with REPLACE; KEEP is the default.

REVOKE
if the REVOKE parameter is specified, or if the owner of the package is not entitled to grant all privileges embodied in the package, all existing grants of the RUN privilege are revoked. The KEEP and REVOKE parameters are allowed only with REPLACE; KEEP is the default.

TO (*server-name***)**
in DB2 Server for VSE, this identifies the application server or servers onto which the package is to be reloaded. The Database Services Utility connects to each application server in turn, and if the connection is successful, the package is reloaded. If an LUW is active when the RELOAD command begins, processing is unsuccessful and an error occurs.

In DB2 Server for VM, this can be specified when you must load the package onto more than one database. To reload the package, the Database Services Utility connects to each specified *server-name* in turn (using database switching, which requires APPC/VM in multiple user mode). If an LUW is active when the RELOAD command begins, processing is unsuccessful and an error occurs.

When the TO clause is specified, the CONNECT statement is processed with no user ID or password. The attempted connection fails if implicit connections are not allowed on *server-name*. After the RELOAD is performed on each database, a COMMIT RELEASE (or ROLLBACK RELEASE) is done, releasing the connection to that *server-name*. The default user ID (the VM user ID) and the default database (as specified to SQLINIT) are reestablished for a new LUW. Any explicit connections done before the RELOAD are therefore lost and must be reissued if required.

If *owner* was not specified, the VM user ID is assumed, because it is the ID used to connect to each *server-name*.

When you use TO, the Database Services Utility ignores preceding CONNECT statements and uses the VM user ID as a default. If you do not want to use your VM user ID, issue the explicit CONNECT statements as required, and use the RELOAD command without a TO clause. If you do not specify the TO clause, the Database Services Utility reloads the package onto only the currently connected database.

There is no specific limit on the number of database names typed; however, there is an implied limit in that the maximum length of a Database Services Utility command is 8192 characters.

INFILE (*ddname*)
identifies and describes the sequential (SAM) tape or disk input file containing the package to be loaded into the database. **The file must be created by UNLOAD PACKAGE processing, and its contents must not be changed in any way.** RELOAD package processing uses a record format of fixed-length blocked (FB) and a record length of 80. The block size should be identical to that used for UNLOAD processing; that is, it must be a multiple of 80.

ddname
in DB2 Server for VSE: this is the TLBL or DLBL job control statement file name for the sequential input file.

Alternatively, RELOAD PACKAGE can read its input from SYSIPT by using a READ MEMBER. You use the READ MEMBER NOCONT option to properly close the SYSIPT file. An example of using READ MEMBER with NOCONT is:

```
RELOAD PACKAGE (package_name) REPLACE INFILE(SYSIPT);
READ MEMBER package_member (NOCONT
```

In DB2 Server for VM: this is the name of the sequential input file defined with a CMS FILEDEF command. Do not specify SYSIN or SYSPRINT as the *ddname*.

BLKSZ (*size*) (DB2 Server for VSE Only)
is an optional parameter that specifies the block size of the sequential output file. It should be identical to that used for UNLOAD processing; that is, it must be a multiple of 80. The default block size is 2000 bytes per block.

PDEV (TAPE or DASD)
is an optional parameter that specifies the device type (DASD or TAPE) of the sequential (SAM) input file. Specify PDEV(DASD) if the input file resides on any device supported by the VSE DTFSD macro. Specify PDEV(TAPE) if the input file resides on a device supported by the VSE DTFMT macro. The default is PDEV(TAPE).

REWIND or NOREWIND
controls tape file rewind processing performed during OPEN processing. This parameter is valid only if you specify TAPE for PDEV. The default processing is REWIND.

REWIND
specifies that the tape file is rewound by OPEN processing.

NOREWIND
specifies that the tape file is not rewound by OPEN processing. If NOREWIND is specified for input tape files referenced by a series of RELOAD commands, you must ensure that the tape files being referenced are in ascending sequence. For example, if NOREWIND is specified in a sequence of two RELOAD commands and the first command reads tape file 2, then the second command must reference tape file 3 or higher. If it references tape file 1, an OPEN error occurs.

UNLOAD PACKAGE

UNLOAD PACKAGE Format

Format:

```
►►──UNLOAD PACKAGE──(package_name)──┬──────────────────────┬──────────────►
                                    └─FROM──(server_name)──┘

►─OUTFILE──(──ddname──┬───────────────────────────┬───────────────────────►
                      │               ┌─2000─┐    │
                      └─BLKSZ──(──────┴─size─┴──)─┘

►─┬─────────────────────────────────────┬─)───────────────────────────────►◄
  │                   ┌─NOREWIND─┐      │
  │          ┌─(TAPE)─┴─REWIND───┴─┐    │
  └─PDEV─────┴─(DASD)──────────────┴────┘
```

VM Example:

UNLOAD PACKAGE(PROG1) FROM(DB1) OUTFILE(OUT1)

VSE Example:

UNLOAD PACKAGE(PROG1) OUTFILE(OUT1)

Authorization:

You must be the owner of the package you want to unload. To unload another user's package, you must have DBA authority.

In VM, you must also have CONNECT authority to a specified database.

The UNLOAD PACKAGE command unloads a specific package to a file. You can unload packages flagged as invalid in the database because RELOAD command processing automatically preprocesses the package again, thus revalidating it. Any unresolved dependencies in the package are then flagged when the RELOAD command preprocesses it again.

Note: The UNLOAD PACKAGE command is not supported if you are using DRDA flow.

Following are descriptions of each part of the command:

PACKAGE (*package_name***)**
identifies an UNLOAD PACKAGE request and the package to be processed.

In DB2 Server for VSE, you can further qualify the *package_name* with the *owner*, separating the two names with a period.

The name of the package is *package_name*. If you do not specify the owner (see "Qualifying Object Names" on page 110 for details), *owner* defaults to the currently connected authorization ID.

The DB2 Server for VM authorization ID is either:

- The authorization ID specified in a previous explicit CONNECT (when FROM is not used)
- The VM user ID (when FROM is used) because it is the user ID used to connect to *server_name*.

FROM (*server-name***)**
in DB2 Server for VSE, this identifies the application server against which the UNLOAD PACKAGE command should be issued. If an LUW is active, the UNLOAD command is unsuccessful and an error occurs.

In DB2 Server for VM, this identifies the application server containing the package. To unload the package, the Database Services Utility connects to the specified *server_name* (using application server switching, which requires APPC/VM in multiple user mode). If an LUW is active, the UNLOAD command fails with an error. When FROM is specified, the CONNECT is attempted with no user ID or password. The attempted connection fails if implicit connects are not allowed on *server_name*.

After the package is unloaded, a COMMIT (or ROLLBACK) RELEASE is done, releasing the connection to *server_name*. The default user ID (the VM user ID), and the default database (as specified to SQLINIT) are reestablished for a new LUW. Any explicit CONNECT statements that you issued before the UNLOAD command are therefore lost and must be reissued, if required. When you use FROM, the Database Services Utility ignores any preceding CONNECT statements and uses the VM user ID as a default. If you do not want to use your VM user ID, issue the explicit CONNECT statements as needed, and use the UNLOAD command without a FROM clause.

OUTFILE (*ddname***)**
identifies and describes the sequential (SAM) tape or disk output file that is to contain the unloaded package. UNLOAD program processing uses a record format of fixed-length blocked (FB) and a record length of 80. The block size, in a VM environment, must be a multiple of 80 and defaults to 80 if not specified. This file contains only one unloaded package. If the file already exists, its contents are replaced; otherwise, the file is created.

ddname
in DB2 Server for VM, this is the name of the sequential output file defined with a CMS FILEDEF command. Do not specify SYSIN or SYSPRINT as the *ddname*.

In DB2 Server for VSE, this is the TLBL or DLBL job control statement file name for the sequential output file.

BLKSZ (*size***) (DB2 Server for VSE only)**
is an optional parameter that specifies the block size of the sequential output file. It must be a multiple of 80. The default block size is 2000 bytes per block.

PDEV (TAPE or DASD)
is an optional parameter that specifies the device type (DASD or TAPE) for the sequential output file. Specify PDEV(DASD) for files that reside on any device supported by the VSE DTFSD macro. Specify PDEV(TAPE) for files that reside on any device supported by the VSE DTFMT macro. The default is PDEV(TAPE).

BLKSZ and PDEV can be specified in any order but must occur after the ddname parameter.

NOREWIND or REWIND
controls tape file rewind processing performed during CLOSE processing. This parameter is valid only if you specify TAPE for PDEV. The default is NOREWIND.

NOREWIND
specifies that the tape file is not rewound by CLOSE processing.

REWIND
specifies that the tape file is rewound by CLOSE processing.

REBIND PACKAGE

REBIND PACKAGE Format

Format:

```
|                                           ┌─EXPLAIN (NO)──┐
 ►►──REBIND PACKAGE──(package_name)───┴─EXPLAIN (YES)─┴──────────────────────►◄
```

Example:

```
REBIND PACKAGE(SMITH.PROG)
```

Authorization:

You must be the owner of the package you want to rebind. To rebind another user's package, you must have DBA authority.

The REBIND PACKAGE command allows you to repreprocess an existing package immediately without unloading and reloading the package. You can use the REBIND PACKAGE command with single or multiple user mode.

Note: The REBIND PACKAGE command is not supported if you are using the DRDA protocol.

Following is the description of the command:

PACKAGE (*package_name***)**
identifies the package that you want to rebind.

You can qualify the name of the package by specifying the *owner* of the package; you must, however, have DBA authority to repreprocess a package belonging to another user. If you do not specify the owner, *owner* defaults to the connected authorization ID. See "Qualifying Object Names" on page 110 for details on accessing data objects that are owned by other users.

| **EXPLAIN(***YES or NO***)**
| determines if explain processing should be performed.

| If EXPLAIN(YES) is specified, then EXPLAIN ALL processing will be done and
| the explain tables will be updated. See EXPLAIN in the *DB2 Server for VSE &*
| *VM SQL Reference* for details on EXPLAIN processing.

| If EXPLAIN is not specified, the default is NO.

Set-Item Commands

SET AUTOCOMMIT

SET AUTOCOMMIT Format

Format:

```
                    ┌(┐  ┌OFF┐  ┌)┐
►►──SET AUTOCOMMIT──┴─┴──┴ON─┴──┴─┴──────────────────────────────►◄
```

The SET AUTOCOMMIT command allows you to activate or suppress the execution of SQL COMMIT statements by the Database Services Utility. This command cannot span input records.

OFF

identifies that you do not want the utility to commit database changes after control commands are successfully processed. You must supply SQL COMMIT statements in the Database Services Utility input stream at the points at which you commit the changes.

In this mode of operation, the only execution of an SQL COMMIT by the utility is at the end-of-program after all control commands have been successfully processed.

ON or OFF must be specified in this command. If you do not supply a SET AUTOCOMMIT command in the input records, the utility operates as if you had issued SET AUTOCOMMIT OFF.

ON

identifies that you want the utility to run an SQL COMMIT command after the successful processing of any control command (that is, Database Services Utility commands or SQL statements) except those noted below. The utility ensures that any database changes made before the receipt of the AUTOCOMMIT ON command are committed before processing continues.

In AUTOCOMMIT ON mode, the Database Services Utility does not run an SQL COMMIT command after the successful processing of these commands:

```
SQL Statements    Database Services Utility Commands

COMMIT            COMMENT
CONNECT           SET AUTOCOMMIT OFF
LOCK              SET ERRORMODE
ROLLBACK          SET FORMAT
                  SET ISOLATION
                  SET LINECOUNT
                  SET LINEWIDTH
                  SET UPDATE STATISTICS
```

SET ERRORMODE

SET ERRORMODE Format

Format:

```
                  ┌(┐   ┌OFF─────┐   ┌)┐
►►──SET ERRORmode─┴─┴───┼ON──────┼───┴─┴──────────────────────────────────►◄
                        └CONTINUE┘
```

The SET ERRORMODE command allows you to:

- Suspend the normal Database Services Utility actions taken after a command processing error is detected and cause the utility to continue processing commands after an error has occurred.
- Force the Database Services Utility to enter error mode processing.
- Resume normal Database Services Utility command processing.

This command cannot span input records. If you do not supply a SET ERRORMODE command in the input records, the utility operates as if you had issued SET ERRORMODE OFF.

OFF

causes the utility to resume execution of Database Services Utility commands and SQL statements after a command processing error has occurred or to terminate error mode CONTINUE processing.

Any subsequent command processing errors cause the utility to execute an SQL ROLLBACK statement and enter Database Services Utility error mode processing.

Notes:

1. The SET ERRORMODE OFF command also terminates the processing mode established by the Database Services Utility SET AUTOCOMMIT ON command.
2. A SET ERRORMODE OFF command is not used:
   a. If the error mode is already off.
   b. If a serious database error has previously occurred. A serious error, by definition, causes all subsequent SQL statements to terminate. Thus, a SET ERRORMODE OFF command in this case has no effect.

ON

causes the utility to suspend execution of following Database Services Utility commands and SQL statements. A SET ERRORMODE ON command terminates:

- The normal command processing mode that was established by default when the utility was run or by a previous SET ERRORMODE OFF command
- The processing mode established by a SET ERRORMODE CONTINUE command.

When a SET ERRORMODE ON command is processed, the utility initiates error mode processing. While in error mode, the Database Services Utility displays commands in the report or message file listing and performs syntax checking on Database Services Utility commands. No SQL statements or Database Services Utility commands (except SET ERRORMODE OFF or SET ERRORMODE CONTINUE) are executed during error mode processing. Therefore, any errors that result from command execution (for example, SQL syntax errors and data file errors) are not detected.

Database Services Utility error mode processing is also entered when the utility detects a command processing error, and the utility is operating in normal command processing mode. If the processing mode established by a SET ERRORMODE CONTINUE command is in effect, and a command processing error is detected, Database Services Utility error mode processing is entered only when the error is a serious database error.

Notes:

1. A SET ERRORMODE ON command also terminates the processing mode established by the SET AUTOCOMMIT ON command.
2. A SET ERRORMODE ON command is not used if Database Services Utility error mode processing is already in effect.

CONTINUE

suppresses the normal Database Services Utility error processing after an error is detected; that is, Database Services Utility error mode processing is not performed. An SQL ROLLBACK statement is not executed, and the utility continues to process Database Services Utility commands and SQL statements after a command processing error occurs. If any errors occur, and they occur only while SET ERRORMODE CONTINUE is in effect, the Database Services Utility issues error message ARI8007I when the Database Services Utility command processing ends.

Notes:

1. If a serious database error occurs while the processing mode established by a SET ERRORMODE CONTINUE command is in effect, an SQL ROLLBACK is executed, and the utility enters error mode processing. A serious database error, by definition, causes all subsequent SQL statements to terminate.
2. After a SET ERRORMODE CONTINUE command is processed, because the Database Services Utility does not execute an SQL ROLLBACK command does not mean that the logical unit of work is always in progress. Certain SQL statement errors cause the logical unit of work to be terminated by the database manager. If one of these errors occurs, all database changes made during the logical unit of work or since the last SQL COMMIT statement are lost.

   To control the logical unit of work after a SET ERRORMODE CONTINUE command is processed, you can use SQL COMMIT statements or the Database Services Utility SET AUTOCOMMIT ON command.
3. The SET ERRORMODE CONTINUE command is not used:
   a. If the processing mode established by a previous SET ERRORMODE CONTINUE command is still in effect
   b. If a serious database error has previously occurred.
4. If the Database Services Utility is not in error mode when a SET ERRORMODE CONTINUE command is encountered, the Database Services Utility AUTOCOMMIT processing status is not changed.

   If the Database Services Utility is in error mode when a SET ERRORMODE CONTINUE command is encountered, the Database Services Utility AUTOCOMMIT processing status is set to off.

5. Database Services Utility end-of-program COMMIT processing is based on the current status of the command processing. That is, if Database Services Utility error mode processing and the SET AUTOCOMMIT ON command processing mode are not in effect, Database Services Utility end-of-program COMMIT processing is performed.
6. All SQL statements are treated as one LUW when running the database with LOGMODE=N and Database Services Utility with AUTOCOMMIT=OFF and ERRORMODE=CONTINUE. If an error occurs, ALL statements are rolled back.

Note: The ERRORMODE setting has an effect on the final Database Services Utility return code. For more information, see Chapter 9, "Error Handling and Debugging," on page 223.

SET FORMAT

SET FORMAT Format

Format:

```
                     ┌(┐   ┌CB┐  ┌)┐
►►──SET FORMAT───────┴─┴───┼CL┼──┴─┴──────────────────────────────────────────►◄
                           └LO┘
```

This command allows you to identify whether the Database Services Utility should use column or block format, column or list format, or only list format for SQL SELECT statement results. If the format is not specified, Database Services Utility processing uses column or block format for SQL SELECT statement output. This command cannot span input records.

The SET FORMAT command overrides the formats specified by either a parameter list format control parameter or the default column or block format. This command specification remains in effect until another SET FORMAT command is encountered and successfully processed, or when Database Services Utility processing ends.

CB
: identifies that either column or block format should be used. The column format is used when a report or message file record can contain all column names or column data for a selected row. The block format is used when a report or message file record cannot contain all column names or column data for a selected row. Column or block format is the default if you do not override it by supplying either a format control parameter—FORMAT(CL) or FORMAT(LO)—or a SET FORMAT command.

CL
: identifies that either column or list format should be used. The column format is used when a report or message file record can contain all column names or column data for a selected row. The list format is used when a report or message file record cannot contain all column names or column data for a selected row.

LO
: identifies that only list format is to be used. The list format is used even when the report or message file record can contain all column names or column data for a selected row.

SET ISOLATION

SET ISOLATION Format

Format:

```
                       ┌(┐   ┌RR┐   ┌)┐
►►──SET ISOLation──────┴─┴───┼CS┼───┴─┴──────────────────────────────────►◄
                             └UR┘
```

This command allows you to control the isolation level used for Database Services Utility processing. Every time the utility is run, the isolation level is initialized to repeatable read (RR). SQL processing through the utility is performed at this isolation level until a SET ISOLATION command is encountered. The utility sets the isolation level to the value specified in the command and processes at this level until another SET ISOLATION command is executed or Database Services Utility processing ends. The other isolation level settings are cursor stability (CS) and uncommitted read (UR). This command cannot span input records.

If you are accessing a non-DB2 Server for VM application server, or if you are using DRDA flow, the isolation level for the Database Services Utility is always set to CS and the SET ISOLATION command has no effect.

RR
: is used to protect a logical unit of work from uncommitted updates of another logical unit of work. Also, no other logical unit of work can modify any row that has been read by this logical unit of work.

CS
: is used to protect a logical unit of work from uncommitted updates of another logical unit of work. After data is read, the data is freed for others to update before the end of the logical unit of work.

Use this setting only when the data is read or when you are the only user authorized to update the data.

UR
: is used when protection from other logical units of work is not required. Data can be read without waiting for other logical units of work that are updating the data. Reading data will not prevent other application processes from updating it.

Note that data integrity may be compromised because read-only access to uncommitted data is allowed.

This setting applies only to read-only operations (SELECTs, DATAUNLOAD and UNLOAD) against data in public dbspaces with ROW or PAGE level locking. For other operations (UPDATE, DELETE, INSERT, DATALOAD, and LOAD), the rules of CS apply.

For dbspaces with DBSPACE level of locking, the rules of RR apply.

Recommended settings for Database Services Utility processing:

- Repeatable Read (RR)
  - To ensure that the database is in a consistent state when using UNLOAD and RELOAD TABLE/DBSPACE commands for database backup or migration.

  The isolation level used to perform DATALOAD commands or perform RELOAD DBSPACE/TABLE commands with the NEW option is not important.

  Regardless of your isolation level setting, all UNLOAD/RELOAD PACKAGE functions are performed with isolation level repeatable read. This does not affect your setting of isolation level when you are performing other functions.
- Cursor Stability (CS)
  - To reduce the contention on the database when running the Database Services Utility with multiple user mode
  - To perform RELOAD DBSPACE/TABLE commands with the PURGE option
  - To use the Database Services Utility in the terminal input mode with AUTOCOMMIT OFF in DB2 Server for VM
  - To perform UNLOAD DBSPACE/TABLE or DATAUNLOAD processing for read only data
  - To update data for which you are the only person with update authorization.
- Uncommitted Read (UR)
  - To reduce the contention on the database when running the Database Services Utility with multiple user mode

  Note that data integrity may be compromised when using UR. UR should only be used when it is not necessary that the data be committed.

SET LINECOUNT, SET LINEWIDTH

SET LINECOUNT (LINEWIDTH) Format

Format:

```
►►──SET──┬─LineCount──(ccc)──┬──────────────────────────────┬─┬──────────►◄
         │                   │               ┌─80──┐        │ │
         │                   │               ├─120─┤        │ │
         │                   └─LineWidth──(──┴─www─┴──)─────┘ │
         └─LineWidth──(www)──┬──────────────────────────────┬─┘
                             │               ┌─60──┐        │
                             └─LineCount──(──┴─ccc─┴──)─────┘
```

Note: 80 is valid in DB2 Server for VM only.

The SET LINECOUNT/LINEWIDTH command allows you to:

- Define the number of lines per page for Database Services Utility report output or message file output.
- Define the number of print data positions used in each Database Services Utility report or message file record containing SQL SELECT statement output.

This command cannot span input records.

LINECOUNT(*ccc*) or LC(*ccc*)
If LINECOUNT(*ccc*) or LC(*ccc*) is specified, the value *ccc* is the number of lines per page of printed output written to the Database Services Utility report or message file. The value *ccc* can range from 10 to 32767; the default value is 60.

LINEWIDTH(*www*) or LW(*www*)
If LINEWIDTH(*www*) or LW(*www*) is specified, the value *www* is the maximum number of print data positions used in a Database Services Utility report or message file record containing SQL SELECT statement output. The default value for *www* is 120. In DB2 Server for VM, if the Database Services Utility message file (*ddname*=SYSPRINT) is assigned to the terminal, the number of print data positions used for the SQL SELECT statement defaults to 80.

The value *www* can range from 60 to 256, but cannot be equal to or greater than the logical record length of the Database Services Utility message file.

Notes:

1. The Database Services Utility always supplies an American Standards Association (ASA) control character in the first position of the print record. The second through nth positions of the print record are the print data positions. If the value *www*+1 is less than the print record length, all unused print data positions in the print record contain a blank (hex 40).
2. The DB2 Server for VSE Database Services Utility report record length is always 121.
3. The minimum DB2 Server for VM Database Services Utility message file record length is 81. If the Database Services Utility control parameter PAGECTL(NO) is specified, the minimum message file record length is 80.
4. If the value *www* is equal to or greater than the print record length, an error occurs.

SET UPDATE STATISTICS

SET UPDATE STATISTICS Format

Format:

```
            ┌─UPDATE─┐              ┌─(─┐ ┌─ON──┐ ┌─)─┐
►►──SET─────┴────────┴──STATISTICS──┴───┴─┴─OFF─┴─┴───┴───────────────────►◄
```

The SET UPDATE STATISTICS command allows you to control the automatic statistics collection performed during Database Services Utility RELOAD TABLE, RELOAD DBSPACE, and DATALOAD TABLE command processing. This command cannot span input records. If you do not supply a SET UPDATE STATISTICS command in the input records, the utility operates as if you had issued SET UPDATE STATISTICS ON.

The SET UPDATE STATISTICS command is not supported on a non-DB2 Server for VM application server or if you are using DRDA flow.

ON
causes the utility to automatically collect statistics for each table loaded during

Database Services Utility RELOAD TABLE, RELOAD DBSPACE, and DATALOAD TABLE command processing. This is the default mode of processing.

The Database Services Utility writes message ARI8980I for each table or dbspace loaded by a RELOAD TABLE, RELOAD DBSPACE, or DATALOAD TABLE command. The message informs you that the statistics were collected while the data was loading.

The Database Services Utility writes message ARI8996I and issues an SQL UPDATE STATISTICS statement for each table loaded if you are using the DATALOAD command and when any one of the following is true:
- You are loading data into more than one table.
- Indexes exist on the table.
- The table that you are loading data into already contains rows.

The SQL UPDATE STATISTICS FOR TABLE statement issued by the Database Services Utility must read the whole table to update the internal DB2 Server for VSE & VM statistics for the table. The statistics are updated based on the current contents of the table and dbspace to which the table is assigned. Refer to the *DB2 Server for VSE & VM Application Programming* manual and the *DB2 Server for VSE & VM Database Administration* manual for a description of the SQL UPDATE STATISTICS statement processing.

OFF
suppresses the Database Services Utility statistics collection performed during RELOAD TABLE, RELOAD DBSPACE, and DATALOAD TABLE command processing.

A SET UPDATE STATISTICS OFF remains in effect until a SET UPDATE STATISTICS ON command is processed or until the Database Services Utility is restarted.

Note: If tables are loaded by the Database Services Utility RELOAD TABLE, RELOAD DBSPACE, and DATALOAD TABLE commands while SET UPDATE STATISTICS OFF is in effect, you must issue an SQL UPDATE STATISTICS statement for the table or dbspace to update the internal statistics.

To avoid the processing overhead associated with an SQL UPDATE STATISTICS statement processing, you can suppress the normal Database Services Utility UPDATE STATISTICS when you are using the DATALOAD TABLE command to load a few records into a table that currently contains a larger number of records. See “Update Statistics Considerations” on page 229 for details.

Chapter 9. Error Handling and Debugging

This chapter describes the types of errors you can encounter, return codes, and storage dumps. The action you take depends on the message that you receive. The Database Services Utility generates its own messages as well as displaying DB2 Server for VSE & VM messages. All messages are explained in the *DB2 Server for VSE Messages and Codes* and *DB2 Server for VM Messages and Codes* manuals.

Except for a report or message file processing error, all Database Services Utility messages are written to the report or message file. If a report or message file error occurs, a Database Services Utility message is generated on the operator console describing the condition. In the report or message file, messages follow the commands that caused them to be generated.

Types of Errors

The Database Services Utility takes different actions depending on the cause of the error. The various kinds of errors are:

- **Database Manager or Operating System Failure**

  If the database manager (or the operating system) is terminated abnormally while the Database Services Utility is running, any database updates that occurred during the in-process Database Services Utility logical unit of work are rolled back by recovery processing when the database manager is restarted. No rollback occurs if the tables reside in nonrecoverable storage pools.
- **Database Services Utility Abnormal Termination Error Handling**

  If the Database Services Utility is terminated abnormally, database manager processing restores any database updates that occurred during the in-process logical unit of work unless updates are made to tables residing in nonrecoverable storage pools.
- **Database Services Utility Processing Errors**

  Database Services Utility processing errors are those errors that do not cause abnormal terminations. If a processing error occurs, and SET ERRORMODE is not reset to OFF, or SET ERRORMODE CONTINUE is not in effect, the Database Services Utility performs these actions:

  1. Writes SQL or Database Services Utility error messages to the report or message file except when message output is suppressed by either of the Database Services Utility control parameters MESSAGES(NONE) or MESSAGES(SQLONLY).
  2. Restores any database updates made during the in-process logical unit of work by executing an SQL ROLLBACK command, unless the updates were made to tables residing in nonrecoverable storage pools.
  3. Performs all possible Database Services Utility syntax checking on the remainder of the (input) control file. The utility does no further database manager processing.
  4. Writes a Database Services Utility message identifying the unsuccessful termination of Database Services Utility processing.
  5. Updates register 15 with a nonzero return code.
  6. Returns control to the invoking program with register linkage processing.

- **Report Errors**

  In DB2 Server for VSE, if the Database Services Utility encounters problems when it tries to open the report, it generates a message to the operator console and terminates processing after performing steps 5 and 6 above. If the utility encounters problems when it tries to write to the report, it generates a message and terminates processing after performing steps 2, 5, and 6 above.
- **Input Control Card File Errors**

  If the Database Services Utility encounters problems when it tries to open the input control card file, it terminates processing after performing steps 4, 5, and 6 described above. If the utility encounters problems when it tries to read from the input control card file, it terminates processing after performing steps 2, 4, 5, and 6 as described above.
- **Message File Errors**

  In DB2 Server for VM, if the Database Services Utility encounters problems when it tries to open the message file, it generates a WTO (write-to-operator) message and terminates processing after performing steps 5 and 6 above. If the utility encounters problems when it tries to write to the message file, it generates a WTO message and terminates processing after performing steps 2, 5, and 6 above.
- **Command File Errors**

  If the Database Services Utility encounters problems when it tries to open the command file, it terminates processing after performing steps 4, 5, and 6 described above. If the utility encounters problems when it tries to read from the command file, it terminates processing after performing steps 2, 4, 5, and 6 as described above.
- **Tape or DASD Data File Errors**

  If an incorrect *ddname* is specified in a Database Services Utility command, or incorrect data file job control or CMS FILEDEF statements are supplied, Database Services Utility processing is terminated. Database manager processing restores any database changes when the Database Services Utility job abnormally terminates because of a data-file-open error.

Return Codes

Although one or more of the following return codes may be encountered during Database Services Utility processing, one final return code is provided at the end of processing (the highest return code found). The following is a list of the return codes:

0 All commands processed successfully.

4 All requested processing completed successfully, and all changes were committed to the database. This return code indicates that an error occurred during Database Services Utility termination; no SQL or Database Services Utility commands need to be reprocessed.

6 One or more errors have occurred in command processing while SET ERRORMODE CONTINUE was in effect.

8 Database Services Utility processing error encountered. From the point of error, no further commands were executed, but subsequent Database Services Utility commands were checked for syntax errors.

If a Database Services Utility SET ERRORMODE OFF command is encountered before the end of the (input) control file is reached, normal Database Services Utility command processing is resumed.

12	Input control card or Control command file-open-error. No commands are processed.
16	Report or Message file-open-error. No commands are processed.
20	Initialization error. Sufficient processor storage was not available for Database Services Utility working storage areas. No commands are processed.

The type of ERRORMODE processing in effect at the time of a command processing error determines the final return code. If both ERRORMODE OFF and ERRORMODE CONTINUE processing are used within one (input) control file (either by default or through a SET ERRORMODE command), one of the following scenarios can exist:

- If an error occurs during ERRORMODE CONTINUE, the final return code is 6.
- If an error occurs during ERRORMODE OFF, the final return code is 8.
- If an error occurs during both ERRORMODE CONTINUE and ERRORMODE OFF, the final return code is 8.

Storage Dumps

Dumps Initiated by the Database Services Utility

The Database Services Utility initiates a storage dump if an illogical condition or critical error arises during its execution. Before a Database Services Utility storage dump of the partition or virtual machine is initiated, the message ARI804E is normally generated, and register 15 is set to a hexadecimal dump identification (DUMP ID) value. After a storage dump is generated, Database Services Utility processing continues.

The message ARI0804E identifies:
- The Database Services Utility module initiating the dump
- The reason code for the dump.

In two instances, Database Services Utility storage dumps are not preceded by the usual ARI0804E message. These are:

- The storage dump (DUMP ID = hex 811) is initiated as a result of the .DEBUG command.
- The storage dump (DUMP ID = hex 803) is initiated by the module ARIDBS before Database Services Utility processing is terminated, and the final Database Services Utility return code (register 15) value is 4 or is greater than 8.

The Database Services Utility modules initiating storage dumps, the reasons for the dumps, and the hexadecimal dump identification values are explained in the *DB2 Server for VSE Messages and Codes* and *DB2 Server for VM Messages and Codes* manuals.

Debugging

Processing for Debug Mode

In debug mode, if the sequence of commands described below is supplied in the DB2 Server for VSE Database Services Utility input control card file, a storage dump of a partition is taken following the next SQL error that occurs. And in DB2 Server for VM, if this sequence of commands is supplied in the Database Services Utility command file, a virtual machine dump is taken following the next SQL error that occurs. An SQL error is identified by an SQLCODE less than 0 or greater than +100 received after the execution of an SQL statement. The resulting storage dump reflects a register 15 value of hex 811 and is generated by a Database Services Utility call to entry point ARISYSDA.

The command sequence necessary to initiate the storage dump is:

```
.DEBUG
SET ERRORMODE OFF;
```

Note:
- .DEBUG must begin in command record column 1.
- An error condition occurs when the .DEBUG command is processed.
- A SET ERRORMODE OFF command must follow the .DEBUG command.

The storage dump is then taken after the next SQL error returned by the database manager (in the SQLCA) after the execution of an SQL statement. Database Services Utility processing continues after the dump is generated. If you want subsequent storage dumps during the same execution of the Database Services Utility, repeat the special command sequence described above.

Guidelines for DEBUG Storage Dump Analysis

Register 15 = X'811'   Register 13 + 4 = ARIDSQLA register save area address for last ARIPRDI CALL

ARIDSQLA save area address:
 + 12 = ARISYSDA (storage dump routine) return address within ARIDSQLA
 + 16 = ARISYSDA entry point
 + 32 = Address of SQLTIE (Register 3 contents at time of dump)
 + 36 = Contents of register 4 at time of dump.

Common processing area (CPA) address + X'0C' = the address of the special save area containing the register 0 through register 15 contents saved by the Database Services Utility before executing the dump request call to ARISYSDA.

Chapter 10. Improving Performance

This chapter adds to the information in previous chapters by raising issues you should consider to get the best performance from the Database Services Utility. Refer to the appropriate sections of earlier chapters if you want more basic information.

Note: All of the following UNLOAD and RELOAD command references apply to TABLE and DBSPACE unless PACKAGE is specified.

Nonrecoverable Storage Pool

With the database manager, you can define nonrecoverable storage pools. It provides limited recovery functions for dbspaces that are assigned to nonrecoverable storage pools. A problem may arise when you are inserting or updating tables stored in nonrecoverable storage pools, especially when you use the DATALOAD and RELOAD commands. See the *DB2 Server for VSE System Administration* or *DB2 Server for VM System Administration* manual for information about nonrecoverable storage pools.

Tape-File Support in DB2 Server for VM

Tape File Support Considerations

You use the CMS FILEDEF command and an optional CMS LABELDEF command to define input or output tape files processed by the Database Services Utility. Refer to the *DB2 Server for VM System Administration* manual for a complete description of the tape file support.

Locking Considerations

When running the Database Services Utility with multiple user mode to load (INSERT) or unload (SELECT) rows from a DB2 Server for VSE & VM database, you may encounter lock escalation, particularly when using isolation level CS or RR. Lock escalation reduces the ability to access the database and increases the likelihood of deadlock conditions, which terminate processing. SQL LOCK DBSPACE or LOCK TABLE statements override the database manager automatic locking mechanism; they can be used to reduce deadlock conditions during Database Services Utility processing. Using isolation level UR to unload rows from a DB2 Server for VSE & VM database may also reduce lock escalation and deadlock conditions; however, it is not recommended because it can cause data integrity problems.

A user-issued SQL LOCK statement is useful only during multiple user mode processing for table data in a public dbspace that is not defined with locking at the dbspace level. A user-acquired database lock remains in effect until the end of the logical unit of work in which it was issued. You cannot lock any database manager catalog tables—regardless of the database authority you have. To lock an eligible dbspace or table, you (the user connected to the database) must meet the requirements in Table 14 on page 228:

Table 14. Requirements to Lock a Dbspace or Table

To Lock:	You Must
DBSPACE	• Be the owner of the dbspace or • Have DBA authority
Table	• Be the owner of the TABLE or • Have DBA authority or • Have SELECT privilege on the table

DATALOAD and RELOAD Locking Considerations

If you insert many rows into the database with a RELOAD command or a DATALOAD command without the COMMITCOUNT option specified, consider using the SQL LOCK DBSPACE statement to eliminate or reduce lock escalation. An exclusive lock on the dbspace where the tables being loaded are defined does not appreciably increase lock contention and reduces the likelihood of deadlock with another user.

Note: An exclusive lock on a **table** being loaded does not prevent lock escalation and is not recommended.

To exclusively lock a dbspace, issue the following command before the DATALOAD or RELOAD command:

Format:

```
►►──LOCK DBSPACE──dbspace_name──IN EXCLUSIVE MODE;──────────────────────────►◄
```

You can also avoid lock escalation during multiple user mode DATALOAD processing by issuing a SET AUTOCOMMIT ON command before the DATALOAD command and specifying a sufficiently low COMMITCOUNT value in the DATALOAD INFILE subcommand. Use of DATALOAD COMMITCOUNT processing reduces the likelihood of the locking required by DATALOAD processing delaying other users accessing the table being loaded or other tables in the same dbspace where the table being defined resides. If the target table is in a dbspace defined with ROW level locking, a COMMITCOUNT value of approximately 200 should be sufficiently low. If the dbspace is defined with PAGE locking, the COMMITCOUNT value can be higher (1000, for example) and lock escalation is still avoided. Do not arbitrarily set the COMMITCOUNT value too low because frequent commit points increase DATALOAD run time.

SELECT, DATAUNLOAD, and UNLOAD Locking Considerations

If you are running the Database Services Utility with the isolation level setting of repeatable read (the default Database Services Utility processing mode) and you know that a particular SELECT, DATAUNLOAD, or UNLOAD operation is going to access many rows from one or more tables in the database, lock escalation then normally occurs. You should consider acquiring a SHARE lock on the table(s) being accessed. If all the tables being accessed reside in the same dbspace, you should consider acquiring a SHARE lock on the dbspace being accessed. This action can reduce lock contention and the likelihood that a SELECT, DATAUNLOAD, or UNLOAD causes a deadlock with another user. Other users can modify other tables in the same dbspace where the table being accessed resides.

To acquire a SHARE lock on a table or dbspace being accessed, issue the following command before the SELECT, DATAUNLOAD, or UNLOAD statement:

Format:

```
►►──LOCK TABLE──table_name──IN SHARE MODE;──────────────────────────────────►◄
or

►►──LOCK DBSPACE──dbspace_name──IN SHARE MODE;──────────────────────────────►◄
```

UNLOAD and RELOAD PACKAGE Considerations

To obtain the best performance when using the UNLOAD PACKAGE command and the RELOAD PACKAGE command, consider doing the following:

- Unload or reload large numbers of packages in your system's off-peak usage time or with single user mode.
- If you are unloading or reloading packages with multiple user mode, use blocking (by ensuring that the Database Services Utility was initialized with the BLOCK option).

These actions improve performance by preventing interruptions by other users.

PROGRAM is a synonym for PACKAGE. Therefore, UNLOAD or RELOAD PROGRAM, and UNLOAD or RELOAD PACKAGE are equivalent commands.

When unloading or reloading a modifiable package, an exclusive lock is held on the catalog table SYSACCESS. This may cause a performance deterioration for other users wanting to run the exclusively locked package.

See the *DB2 Server for VSE & VM Database Administration* manual for further information on locking.

Update Statistics Considerations

If you suppress automatic statistics collection by specifying SET UPDATE STATISTICS OFF in the (input) control file before issuing the DATALOAD command, you must issue an UPDATE STATISTICS statement to collect statistics. The UPDATE STATISTICS statement performs a dbspace scan, so it can be time-consuming if the number of active data pages in that dbspace is large. Consider suppressing statistics collection only if you know the statistics are not going to change significantly (for example, a small amount of data is being added to a large table). In this situation, you can postpone updating the statistics until more substantial changes have occurred.

Reorganizing Indexes

The REORGANIZE INDEX command corrects index fragmentation and corrects the skewing of index key values. REORGANIZE INDEX also revalidates an invalid index.

REORGANIZE INDEX automatically updates statistics while the index is being rebuilt. To calculate index statistics, the Database Services Utility has to have an exact count of the pages in the dbspace which contain rows from the indexed table. When the table is the only table in the dbspace, the database manager can find out how many pages contain rows from the dbspace directory. In other words, the number of used pages is the number of pages containing rows. If there is more than one table in the dbspace, the database manager has to scan each page to determine which ones are occupied by the indexed table. Hence, you get better performance for the REORGANIZE INDEX command when the indexed table is the only table in the dbspace. The DB2 Server for VSE & VM manuals recommend allocating one table per dbspace if the tables are large.

To reorganize a valid index, the database manager uses an internal dbspace as temporary storage to hold the keys of the index. The internal dbspace requirements to perform a REORGANIZE INDEX operation are one third of that required to perform the equivalent CREATE INDEX. If you do not have enough space, see the *DB2 Server for VSE System Administration* or *DB2 Server for VM System Administration* manual.

Packages that depend on an index are not invalidated when the index is reorganized. Therefore, using the REORGANIZE INDEX command instead of dropping and re-creating the index explicitly avoids the cost of the automatic preprocessing that reoccurs the next time an invalidated package is run. This benefit is realized whether the index you reorganize is valid or invalid.

Double-Byte Character Set

The database manager provides support for basic DBCS, while the Database Services Utility provides support for extended DBCS.

Basic Support

The Database Services Utility supports the use of all DB2 Server for VSE & VM data types, including the GRAPHIC data type for double-byte character set (DBCS) data. The following general rules apply to DBCS data in Database Services Utility input or output:

- DBCS data in SQL statements processed by the Database Services Utility can be supplied as a constant with the format:

  ```
  G'SOxx-xxSI'
  ```

  where:

  G' is the required constant prefix.
  SO is a shift-out delimiter (hex 0E).
  xx is a DBCS character in paired bytes.
  SI is a shift-in delimiter (hex 0F).
  ' is the character terminating the constant.

  Note: N' can be used as a synonym for G'
- DBCS data appearing in input data records is read into a graphic data type column by the Database Services Utility DATALOAD command processing. The DBCS data must be represented in paired bytes; the SO and SI delimiters are optional.
- The paired bytes of a DBCS character cannot be split across Database Services Utility (input) control file records, except for data records read by Database Services Utility DATALOAD command processing. A DBCS data string, as well

as the paired bytes representing a DBCS character, can be split across (input) control file data records processed using the DATALOAD continued record support.

- GRAPHIC data appearing in Database Services Utility DATAUNLOAD command output and GRAPHIC data in SELECT

  command print records has the format:

  ```
  SOxx-xxSI
  ```

  where:

  SO is a shift-out delimiter (hex 0E).
  xx is a DBCS character in paired bytes.
  SI is a shift-in delimiter (hex 0F).

  The SO and SI delimiters result in blank print positions.
- A DBCS data string appearing in a Database Services Utility COMMENT command must include shift-out and shift-in delimiters and cannot be continued across (input) control file records.

Extended Support

In DB2 Server for VSE, if the SQLOPTION column value is DBCS and the VALUE column value is *no* in the SYSOPTIONS catalog table, you can use the extended DBCS support of the Database Services Utility. The DBCS option value will be retrieved from the SQLGLOB file. The User DBCS SQLGLOB value, if it exists, will be the default setting for DBCS.

The DB2 Server for VM Database Services Utility retrieves information about the DBCS setting from the LASTING GLOBALV file on the database user machine. Extended DBCS supports the following characteristics:

- Any SQL identifier or character string constant in a command can contain DBCS/EBCDIC mixed data if the DBCS string and the shift-in or shift-out delimiters reside on the same line.
- The SQL SELECT statement ensures that the data printed for CHAR, VARCHAR, or long field columns contain matched pairs of shift-out and shift-in delimiters for each data line. A long field is a field that is either a LONG VARCHAR field, a LONG VARGRAPHIC field, a VARCHAR(n) field where *n* is greater than 254 but less than or equal to 32767, or a VARGRAPHIC(n) field where *n* is greater than 127 but less than or equal to 16383.
- The Database Services Utility assumes, when a command data record contains a shift-out delimiter without a shift-in delimiter, that all trailing command data positions within the command record contain DBCS data. The Database Services Utility inserts a shift-in delimiter after the last assumed nonblank DBCS character position before the record is written to the report or message file.

  Omitting a shift-in or shift-out delimiter causes unreadable (input) control file records in the report or message file. To suppress this display of unreadable DBCS data, set the LIST parameter of the INFILE subcommand to *no*.

If an error occurs, in DB2 Server for VSE, during access of the SYSOPTIONS catalog, or if an invalid DBCS option value is found, the Database Services Utility continues processing the input control card file as if the extended DBCS feature were not in effect.

Part 3. Appendixes

Appendix A. Sample Tables

The sample tables illustrated in this appendix are used in examples throughout the library. These tables simulate a database created for use in organization or project management applications. As a group, the tables include information that describes employees, departments, projects, and activities. Figure 100 shows the relationships among the tables. These relationships are established by referential constraints, where a foreign key in the dependent table references a primary key in the parent table. In the figure, the referential constraint is symbolized by lines joining the keys; the arrowheads point from the primary key to the foreign key. Only those columns named as foreign or primary keys are listed in the figure. All tables have additional columns. You can easily review the contents of any table by executing an SQL statement, such as `SELECT * FROM SQLDBA.DEPARTMENT`.

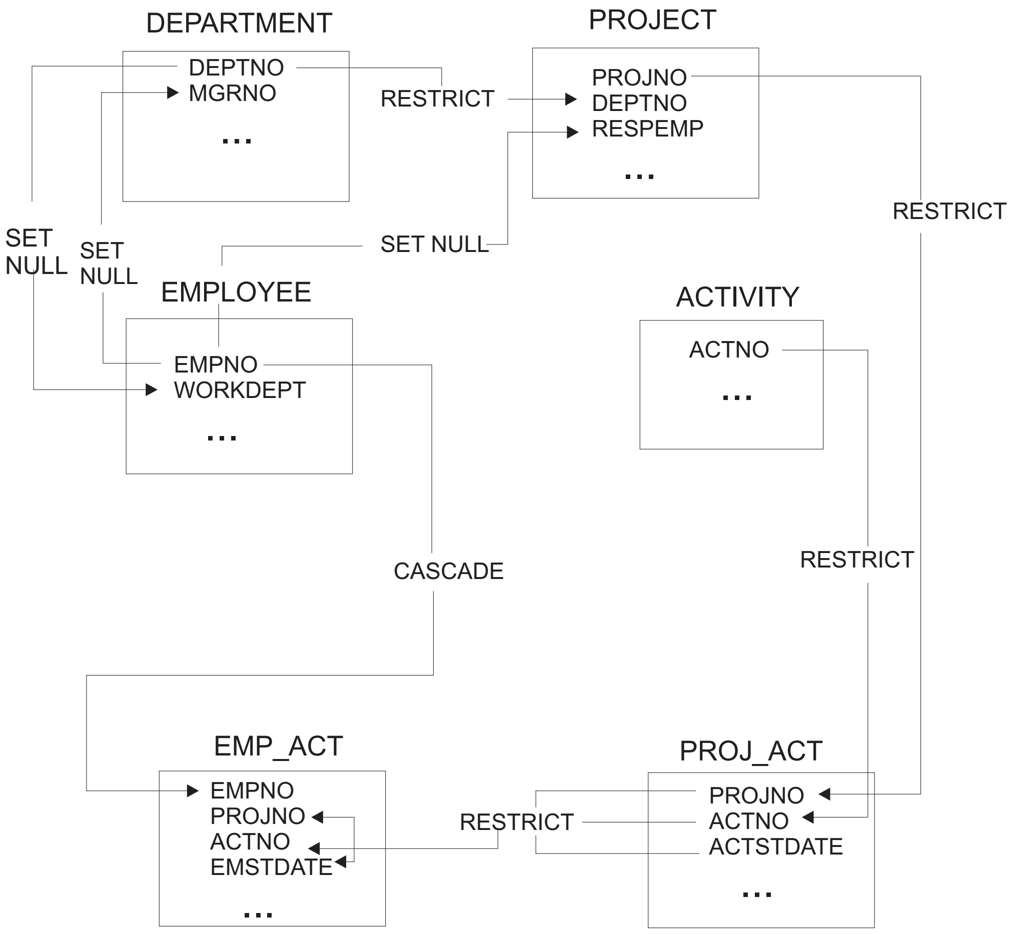


Figure 100. Relationships among Tables in the Sample Application

DEPARTMENT Table

The DEPARTMENT table describes each department in the business and identifies its manager and the department to which it reports. The table contents are shown in Figure 101 on page 236; a description of the columns is shown in Figure 102.

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER	?	A00
E01	SUPPORT SERVICES	000050	A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E11	OPERATIONS	000090	E01
E21	SOFTWARE SUPPORT	000100	E01

Figure 101. DEPARTMENT Table Contents

Column Name	Description
DEPTNO	Department number, the primary key
DEPTNAME	A name describing the general activities of the department
MGRNO	Employee number (EMPNO) of the department manager
ADMRDEPT	Number of the department to which this department reports; the department at the highest level reports to itself

Figure 102. Columns of the DEPARTMENT Table

The DEPARTMENT table is created with:

```
CREATE TABLE DEPARTMENT
    (DEPTNO    CHAR(3)           NOT NULL,
     DEPTNAME  VARCHAR(36)       NOT NULL,
     MGRNO     CHAR(6)                   ,
     ADMRDEPT  CHAR(3)           NOT NULL,
     PRIMARY KEY (DEPTNO)                )
```

After the EMPLOYEE table has been created, a foreign key is added to the DEPARTMENT table with this statement:

```
ALTER TABLE DEPARTMENT ADD
     FOREIGN KEY R_EMPLY1 (MGRNO) REFERENCES EMPLOYEE
          ON DELETE SET NULL
```

Relationship to Other Tables

DEPARTMENT is a parent of the EMPLOYEE and PROJECT tables.

The DEPARTMENT table is a dependent of the EMPLOYEE table; the MGRNO column is the foreign key in the DEPARTMENT table and references EMPNO, the primary key in the EMPLOYEE table.

EMPLOYEE Table

The EMPLOYEE table identifies all employees by an employee number and lists basic personnel information. The table in Table 15 on page 238 shows the contents of the EMPLOYEE table; Table 16 on page 240 shows a description of the columns.

Table 15. EMPLOYEE Table Contents

EMPNO	FIRSTNME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIREDATE	JOB	ED LEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
char(6) not null	varchar(12) not null	char(1) not null	varchar(15) not null	char(3)	char(4)	date	char(8)	smallint not null	char(1)	date	dec(9,2)	dec(9,2)	dec(9,2)
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01	PRES	18	F	1933-08-24	52750	1000	4220
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250	800	3300
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250	800	3060
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175	800	3214
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250	500	2580
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170	700	2893
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750	600	2380
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150	500	2092
000110	VINCENZO	G	LUCCHESSI	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500	900	3720
000120	SEAN		O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250	600	2340
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800	500	1904
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420	600	2274
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280	500	2022
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250	400	1780
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680	500	1974
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340	500	1707
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450	400	1636
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740	600	2217
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270	400	1462
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840	600	2387
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK	14	M	1935-05-30	22180	400	1774
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760	600	2301
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180	400	1534
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250	300	1380
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380	500	2190

Table 15. EMPLOYEE Table Contents (continued)

EMPNO	FIRSTNME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIREDATE	JOB	ED LEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250	500	2100
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340	300	1227
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750	400	1420
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900	300	1272
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FIELDREP	16	M	1932-08-11	19950	400	1596
000330	WING		LEE	E21	2103	1976-02-23	FIELDREP	14	M	1941-07-18	25370	500	2030
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840	500	1907

Table 16. Columns of the EMPLOYEE Table

Column Name	Description
EMPNO	Employee number (the primary key)
FIRSTNME	First name of the employee
MIDINIT	Middle initial of the employee
LASTNAME	Last name of the employee
WORKDEPT	Number of department in which the employee works
PHONENO	Employee telephone number
HIREDATE	Date of hire
JOB	Job held by the employee
EDLEVEL	Number of years of formal education
SEX	Sex of the employee (M or F)
BIRTHDATE	Date of birth
SALARY	Yearly salary
BONUS	Yearly bonus
COMM	Yearly commission

The EMPLOYEE table has a foreign key referencing the primary key in the DEPARTMENT table. The DEPARTMENT table must, therefore, be created first. The EMPLOYEE table is then created with:

```
CREATE TABLE EMPLOYEE
   (EMPNO     CHAR(6)         NOT NULL,
    FIRSTNME  VARCHAR(12)     NOT NULL,
    MIDINIT   CHAR(1)         NOT NULL,
    LASTNAME  VARCHAR(15)     NOT NULL,
    WORKDEPT  CHAR(3)                 ,
    PHONENO   CHAR(4)                 ,
    HIREDATE  DATE                    ,
    JOB       CHAR(8)                 ,
    EDLEVEL   SMALLINT        NOT NULL,
    SEX       CHAR(1)                 ,
    BIRTHDATE DATE                    ,
    SALARY    DECIMAL(9,2)            ,
    BONUS     DECIMAL(9,2)            ,
    COMM      DECIMAL(9,2)            ,
    PRIMARY KEY (EMPNO)               ,
    FOREIGN KEY R_DEPT1 (WORKDEPT) REFERENCES DEPARTMENT
          ON DELETE SET NULL       )
```

Relationship to Other Tables

The EMPLOYEE table is a parent of the DEPARTMENT table, the PROJECT table, and the EMP_ACT table.

The EMPLOYEE table is a dependent of the DEPARTMENT table; the foreign key on the WORKDEPT column in the EMPLOYEE table references the primary key on the DEPTNO column in the DEPARTMENT table.

PROJECT Table

The PROJECT table describes each project that the business is currently undertaking. Data contained in each row includes the project number, name, person responsible, and schedule dates as shown in Table 17; Table 18 describes the columns.

Table 17. PROJECT Table Contents

PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
AD3100	ADMIN SERVICES	D01	000010	6.5	1982-01-01	1983-02-01	?
AD3110	GENERAL ADMIN SYSTEMS	D21	000070	6	1982-01-01	1983-02-01	AD3100
AD3111	PAYROLL PROGRAMMING	D21	000230	2	1982-01-01	1983-02-01	AD3110
AD3112	PERSONNEL PROGRAMMING	D21	000250	1	1982-01-01	1983-02-01	AD3110
AD3113	ACCOUNT PROGRAMMING	D21	000270	2	1982-01-01	1983-02-01	AD3110
IF1000	QUERY SERVICES	C01	000030	2	1982-01-01	1983-02-01	?
IF2000	USER EDUCATION	C01	000030	1	1982-01-01	1983-02-01	?
MA2100	WELD LINE AUTOMATION	D01	000010	12	1982-01-01	1983-02-01	?
MA2110	W L PROGRAMMING	D11	000060	9	1982-01-01	1983-02-01	MA2100
MA2111	W L PROGRAM DESIGN	D11	000220	2	1982-01-01	1982-12-01	MA2110
MA2112	W L ROBOT DESIGN	D11	000150	3	1982-01-01	1982-12-01	MA2110
MA2113	W L PROD CONT PROGS	D11	000160	3	1982-02-15	1982-12-01	MA2110
OP1000	OPERATION SUPPORT	E01	000050	6	1982-01-01	1983-02-01	?
OP1010	OPERATION	E11	000090	5	1982-01-01	1983-02-01	OP1000
OP2000	GEN SYSTEMS SERVICES	E01	000050	5	1982-01-01	1983-02-01	?
OP2010	SYSTEMS SUPPORT	E21	000100	4	1982-01-01	1983-02-01	OP2000
OP2011	SCP SYSTEMS SUPPORT	E21	000320	1	1982-01-01	1983-02-01	OP2010
OP2012	APPLICATIONS SUPPORT	E21	000330	1	1982-01-01	1983-02-01	OP2010
OP2013	DB/DC SUPPORT	E21	000340	1	1982-01-01	1983-02-01	OP2010
PL2100	WELD LINE PLANNING	B01	000020	1	1982-01-01	1982-09-15	MA2100

Table 18. Columns of the PROJECT Table

Column Name	Description
PROJNO	Project number (the primary key)
PROJNAME	Project name

Table 18. Columns of the PROJECT Table (continued)

Column Name	Description
DEPTNO	Number of department responsible for the project
RESPEMP	Number of employee responsible for the project
PRSTAFF	Estimated mean project staffing (mean number of persons) needed between PRSTDATE and PRENDATE to achieve the whole project, including any subprojects
PRSTDATE	Estimated project start date
PRENDATE	Estimated project end date
MAJPROJ	Number of any major project of which the subject project may be a part

The PROJECT table has foreign keys referencing DEPARTMENT and EMPLOYEE. The EMPLOYEE and DEPARTMENT tables must be created before the PROJECT table. Once EMPLOYEE and DEPARTMENT are created, the following statement creates the PROJECT table:

```
CREATE TABLE PROJECT
    (PROJNO    CHAR(6)        NOT NULL,
     PROJNAME  VARCHAR(24)    NOT NULL,
     DEPTNO    CHAR(3)        NOT NULL,
     RESPEMP   CHAR(6)                ,
     PRSTAFF   DECIMAL(5,2)           ,
     PRSTDATE  DATE                   ,
     PRENDATE  DATE                   ,
     MAJPROJ   CHAR(6)                ,
     PRIMARY KEY (PROJNO)             ,
     FOREIGN KEY R_DEPT2 (DEPTNO) REFERENCES DEPARTMENT
          ON DELETE RESTRICT          ,
     FOREIGN KEY R_EMPLY2 (RESPEMP) REFERENCES EMPLOYEE
          ON DELETE SET NULL          )
```

Relationship to Other Tables

PROJECT is a parent of the PROJ_ACT table.

PROJECT is a dependent of:

- The DEPARTMENT table; the foreign key on the DEPTNO column in PROJECT references the primary key in the DEPARTMENT table.
- The EMPLOYEE table; the foreign key on the RESPEMP column in PROJECT references the primary key in the EMPLOYEE table.

ACTIVITY Table

The ACTIVITY tables describes the activities that can be performed during a project. The table acts as a master list of possible activities, identifying the activity number, and providing a description of the activity. Figure 103 on page 243 shows table contents; Figure 104 on page 243 shows a description of the columns.

ACTNO	ACTKWD	ACTDESC
160	ADMDB	Adm databases
170	ADMDC	Adm data comm
90	ADMQS	Adm query system
150	ADMSYS	Adm operating sys
70	CODE	Code programs
110	COURSE	Develop courses
30	DEFINE	Define specs
180	DOC	Document
20	ECOST	Estimate cost
40	LEADPR	Lead program/design
60	LOGIC	Describe logic
140	MAINT	Maint software sys
10	MANAGE	Manage/advise
130	OPERAT	Oper computer sys
50	SPECS	Write specs
120	STAFF	Pers and staffing
100	TEACH	Teach classes
80	TEST	Test programs

Figure 103. ACTIVITY Table Contents

Column Name	Description
ACTNO	Activity number (the primary key)
ACTKWD	Activity keyword (up to six characters)
ACTDESC	Activity description

Figure 104. Columns of the ACTIVITY Table

The ACTIVITY table is created with:

```
CREATE TABLE ACTIVITY
     (ACTNO    SMALLINT       NOT NULL,
      ACTKWD   CHAR(6)        NOT NULL,
      ACTDESC  VARCHAR(20)    NOT NULL,
      PRIMARY KEY (ACTNO)             )
```

Relationship to Other Tables

ACTIVITY is a parent of the PROJ_ACT table.

PROJ_ACT Table

The PROJ_ACT table lists the activities performed for each project. The table contains information on the start and completion dates of the project activity as well as staffing requirements as shown in Figure 105 on page 244. Figure 106 on page 245 shows a description of the columns.

PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
AD3100	10	0.50	1982-01-01	1982-07-01
AD3110	10	1.00	1982-01-01	1983-01-01
AD3111	60	0.80	1982-01-01	1982-04-15
AD3111	70	1.50	1982-02-15	1982-10-15
AD3111	80	1.25	1982-04-15	1983-01-15
AD3111	180	1.00	1982-10-15	1983-01-15
AD3112	60	0.75	1982-01-01	1982-05-15
AD3112	60	0.75	1982-12-01	1983-01-01
AD3112	70	0.75	1982-01-01	1982-10-15
AD3112	80	0.35	1982-08-15	1982-12-01
AD3112	180	0.50	1982-08-15	1983-01-01
AD3113	60	0.75	1982-03-01	1982-10-15
AD3113	70	1.25	1982-06-01	1982-12-15
AD3113	80	1.75	1982-01-01	1982-04-15
AD3113	180	0.75	1982-03-01	1982-07-01
IF1000	10	0.50	1982-01-01	1983-01-01
IF1000	90	1.00	1982-01-01	1983-01-01
IF1000	100	0.50	1982-01-01	1983-01-01
IF2000	10	0.50	1982-01-01	1983-01-01
IF2000	100	0.75	1982-01-01	1982-07-01
IF2000	110	0.50	1982-03-01	1982-07-01
IF2000	110	0.50	1982-10-01	1983-01-01
MA2100	10	0.50	1982-01-01	1982-11-01
MA2100	20	1.00	1982-01-01	1982-03-01
MA2110	10	1.00	1982-01-01	1983-02-01
MA2111	40	1.00	1982-01-01	1983-02-01
MA2111	50	1.00	1982-01-01	1092-06-01
MA2111	60	1.00	1982-06-01	1983-02-01
MA2112	60	2.00	1982-01-01	1982-07-01
MA2112	70	1.50	1983-02-01	1983-02-01
⋮	⋮	⋮	⋮	⋮

Figure 105. Partial Contents of PROJ_ACT Table

Column Name	Description
PROJNO	Project number
ACTNO	Activity number
ACSTAFF	Estimated mean number of employees needed to staff the activity
ACSTDATE	Estimated activity start date
ACENDATE	Estimated activity completion date

Figure 106. Columns of the PROJ_ACT Table

The table has a composite primary key and was created with:

```
CREATE TABLE PROJ_ACT
     (PROJNO     CHAR(6)      NOT NULL,
      ACTNO      SMALLINT     NOT NULL,
      ACSTAFF    DECIMAL(5,2)         ,
      ACSTDATE   DATE         NOT NULL,
      ACENDATE   DATE                 ,
      PRIMARY KEY (PROJNO, ACTNO, ACTSTDATE),
      FOREIGN KEY R_PROJ2 (PROJNO) REFERENCES PROJECT
              ON DELETE RESTRICT,
      FOREIGN KEY R_ACTIV (ACTNO) REFERENCE ACTIVITY
              ON DELETE RESTRICT)
```

Relationship to Other Tables

PROJ_ACT is a parent of the EMP_ACT table.

It is a dependent of:

- The ACTIVITY table; the foreign key on ACTNO in the PROJ_ACT table references the primary key, ACTNO, in the ACTIVITY table.
- The PROJECT table; the foreign key on PROJNO in the PROJ_ACT table references the primary key, PROJNO, in the PROJECT table.

EMP_ACT Table

The EMP_ACT table identifies the employee performing each activity listed for each project. The table in Figure 107 on page 246 shows some of the rows in this table. Figure 108 on page 246 shows a description of the columns.

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000130	IF1000	90	1.00	1982-01-01	1982-10-01
000130	IF1000	100	.50	1982-10-01	1983-01-01
000140	IF1000	90	.50	1982-10-01	1983-01-01
000030	IF1000	10	.50	1982-06-01	1983-01-01
000030	IF2000	10	.50	1982-01-01	1983-01-01
000140	IF2000	100	1.00	1982-01-01	1982-03-01
000140	IF2000	100	.50	1982-03-01	1982-07-01
000140	IF2000	110	.50	1982-03-01	1982-07-01
000140	IF2000	110	.50	1982-10-01	1983-01-01
000010	MA2100	10	.50	1982-01-01	1982-11-01
000110	MA2100	20	1.00	1982-01-01	1982-03-01
000020	PL2100	30	1.00	1982-01-01	1982-09-15
000010	MA2110	10	1.00	1982-01-01	1983-02-01
000220	MA2111	40	1.00	1982-01-01	1983-02-01
⋮	⋮	⋮	⋮	⋮	⋮

Figure 107. Partial Contents of EMP_ACT Table

Column Name	Description
EMPNO	Employee number
PROJNO	Project number of the project to which the employee is assigned
ACTNO	Activity number within a project to which an employee is assigned
EMPTIME	A proportion of the employee’s full time (between 0.00 and 1.00) to be spent on the project from EMSTDATE to EMENDATE
EMSTDATE	Date the activity starts
EMENDATE	Completion date of the activity

Figure 108. Columns of the EMP_ACT Table

Since the table has foreign keys referencing EMPLOYEE and PROJ_ACT, those tables must be created first.

This table was created with:

```
CREATE TABLE EMP_ACT
      (EMPNO     CHAR(6)          NOT NULL,
       PROJNO    CHAR(6)          NOT NULL,
       ACTNO     SMALLINT         NOT NULL,
       EMPTIME   DECIMAL(5,2)             ,
       EMSTDATE  DATE                     ,
       EMENDATE  DATE                     ,
       FOREIGN KEY R_PROACT (PROJNO,ACTNO,EMSTDATE)
```

```
        REFERENCES PROJ_ACT ON DELETE RESTRICT,
FOREIGN KEY R_EMPLY3 (EMPNO) REFERENCES EMPLOYEE
        ON DELETE CASCADE          )
```

Relationship to Other Tables

The EMP_ACT table is a dependent of:

- The EMPLOYEE table; the foreign key on EMPNO in the EMP_ACT table references the primary key, EMPNO, in the EMPLOYEE table.
- The PROJ_ACT table; the foreign key on the set of PROJNO, ACTNO, EMSTDATE in the EMP_ACT table references the primary key, PROJNO, ACTNO, ACSTDATE, in the PROJ_ACT table.

IN_TRAY Table

The IN_TRAY table contains a person's note log. The table contents are shown in Figure 109; a description of the columns is shown in Figure 110.

RECEIVED	SOURCE	SUBJECT	NOTE_TEXT
1965-01-01-07.00.00	SQLDBA	English	Here is a note from your DBA.

Figure 109. IN_TRAY Table Contents

Column Name	Description
RECEIVED	Date and time note was received
SOURCE	User id of person sending note
SUBJECT	Brief description
NOTE_TEXT	The text of the note

Figure 110. Columns of the IN_TRAY Table

This table was created with:

```
CREATE TABLE IN_TRAY
     (RECEIVED     TIMESTAMP   NOT NULL,
      SOURCE       CHAR(8)     NOT NULL,
      SUBJECT      CHAR(64)            ,
      NOTE_TEXT    VARCHAR(4000)       )
```

CL_SCHED Table

The CL_SCHED table describes a classroom schedule. The table contents are shown in Figure 111; a description of the columns is shown in Figure 112 on page 248.

CLASS_CODE	DAY	STARTING	ENDING
101:KAR	2	14.10.00	16.10.00
202:LMM	3	14.40.00	16.40.00
303:RAR	4	09.00.00	09.40.00

Figure 111. CL_SCHED Table Contents

CL_SCHED Table

Column Name	Description
CLASS_CODE	Class Code (room:teacher)
DAY	Day number of four day schedule
STARTING	Class start time
ENDING	Class end time

Figure 112. Columns of the CL_SCHED Table

This table was created with:

```
CREATE TABLE CL_SCHED
     (CLASS_CODE      CHAR(7)  NOT NULL,
      DAY             SMALLINT NOT NULL,
      STARTING        TIME     NOT NULL,
      ENDING          TIME     NOT NULL)
```

Note: For more information about data types, refer to the *DB2 Server for VSE & VM Application Programming* manual.

Appendix B. FILEDEF Command Syntax and Notes

Whenever you run the Database Services Utility under CMS, first identify the files to CMS with the FILEDEF command.

Note: Use the SQLDBSU EXEC, which generates standard FILEDEF statements, to define the control and message files. Create a FILEDEF statement for all additional input and output files.

The FILEDEF command in CMS performs the same functions as the data definition (DD) record in OS job control language (JCL). When you enter a FILEDEF command, specify:
- A *ddname*
- The device type
- A file identification if the device type is DISK
- Options (as required).

The format of the FILEDEF command is:

Format:

```
►►──FIledef──ddname──┬─Terminal───────┬──┬─────────────────┬────────────────────────────►◄
                     ├─PRinter────────┤  └─(──Options──┬───┬┘
                     ├─Reader─────────┤                └─)─┘
                     ├─DISK──fn_ft_fm─┤
                     └─TAPn───────────┘
```

ddname **(data definition name)**
: identifies the name used in your Database Services Utility command that refers to the input or output file.

Terminal
: your workstation

PRinter
: the spooled printer available to you

Reader
: the spooled reader available to you

DISK *fn ft fm*
: virtual direct access storage device (DASD) CMS file

TAP*n*
: magnetic tape drive, where *n* can be *1*, *2*, *3*, or *4*, representing virtual units 181, 182, 183, and 184, respectively.

Options
: to avoid error messages, specify only those options that are valid for a particular device. Table 19 on page 251 shows valid options for each device type.

The following diagram illustrates the FILEDEF options available when the device is a workstation or a tape drive:

Format:

```
►►──(──┬──────┬──┬──────────┬──┬────────────────┬──┬─────────────┬──────────────────────────────────────►
       └─PERM─┘  ├─CHANGE───┤  └─RECFM──────────┘  └─LRECL──nnnn─┘
                 └─NOCHANGE─┘          ├─F───┤
                                       ├─FB──┤
                                       ├─V───┤
                                       ├─VB──┤
                                       ├─FBS─┤
                                       └─VBS─┘

►─BLOCK──nnnn──┬────────────┬──┬────────────┬──┬─────┬──────────────────────────────────────────────────►◄
               │       (1)  │  │       (2)  │  └─)───┘
               ├─UPCASE─────┤  ├─7TRACK─────┤
               │        (1) │  │       (2)  │
               └─LOWCASE────┘  └─9TRACK─────┘
```

Notes:

1 Terminal only.

2 Tape only.

The following diagram illustrates the FILEDEF options available when the device specified is DISK:

Format:

```
►►──(──┬──────┬──┬──────────┬──┬─────────────────┬──┬─────────────┬──┬───────────────┬─────────────────►
       └─PERM─┘  ├─CHANGE───┤  └─RECFM──┬─F───┬─┘  └─LRECL──nnnn─┘  ├─*─────────────┤
                 └─NOCHANGE─┘           ├─FB──┤                     └─BLOCK──nnnn───┘
                                        ├─V───┤
                                        ├─VB──┤
                                        ├─FBS─┤
                                        └─VBS─┘

►─┬──────────────┬──┬──────────┬──┬────────────────┬──┬────────┬──┬──────────────┬──┬─────┬──────────►◄
  ├─XTENT50──────┤  └─DISP MOD─┘  └─MEMBER─mbrname─┘  └─CONCAT─┘  └─DSORG──┬─PS─┤  └─)───┘
  └─XTENT──nnnn──┘                                                         ├─PO─┤
                                                                           └─DA─┘
```

Table 19. FILEDEF Options and Parameters

OPTION NAME	DISK	READER/PRINTER	TAPn	TERMINAL
BLOCK, BLOCKSIZE	X	X	X	X
CHANGE, NOCHANGE	X	X	X	X
CONCAT	X			
DEN			X	
DISP MOD	X		X	
DSORG	X			
LOWERCASE/ UPCASE				X
LRECL	X	X	X	X
MEMBER	X			
PERM	X	X	X	X
RECFM	X	X	X	X
7TRACK/9TRACK			X	

Some guidelines for entering FILEDEF specifications are given below.

Specifying *ddname*

If the FILEDEF command is issued for a program input or output file, the *ddname* must be the same as the *ddname* or file name specified for the file in the source program. For example, you have an Assembler language source program that contains the line:

```
INFILE   DCB   ddname=INPUTDD,MACRF=GL,DSORG=PS,RECFM=F,LRECL=80
```

For a particular execution of this program, you want to use as your input file a CMS file on your A-disk that is named MYINPUT FILE. You must issue a FILEDEF like this before executing the program:

```
FILEDEF INPUTDD DISK MYINPUT FILE A1
```

CMS FILEDEF command information for RELOAD processing should be identical to the information in the FILEDEF command used when the file was created by the package's UNLOAD command processing.

If the input data file was created by DATAUNLOAD processing, then the CMS FILEDEF command that defines the DATALOAD input data file should be identical to the information in the FILEDEF command used when the file was created by DATAUNLOAD processing.

Specifying Device Type

For input files, the device type you enter on the FILEDEF command indicates the device from which you want records read. It can be DISK, TERMINAL, READER (for input from real cards or virtual cards), or TAP*n* (for tape). Using the above example, if your input file is to be read from your virtual card reader, the FILEDEF command might be as follows:

```
FILEDEF INPUTDD READER
```

Or, if you were reading from a tape attached to your virtual machine at virtual address 181 (TAP1):

```
FILEDEF INPUTDD TAP1
```

For output files, the device you specify can be DISK, PRINTER, TAP*n* (tape), or TERMINAL.

Entering File Identifiers

If you are using a CMS disk file for your input or output, specify:

```
FILEDEF ddname DISK filename filetype filemode
```

Note: If an asterisk (*) is used for the file mode of an output file, the results are unpredictable. The file mode field is optional; your A-disk is the default assumed.

If you want an output file to be constructed in OS simulated data set format, you must specify the file mode number as 4. For example, a program contains a dbspace for an output file with the *ddname* OUTPUTDD, and you are using it to create a CMS file named DTABSE OUTPUT on your B-disk:

```
FILEDEF OUTPUTDD DISK DTABSE OUTPUT B4
```

If you enter only the *ddname* and device type on the FILEDEF command, such as:

```
FILEDEF ddname DISK
```

where *ddname* is the name of the output file you assigned as the parameter of the FILEDEF command, you have then created a file on your A-disk. For example, if you assign a *ddname* of OSCAR to an output file and do not issue a FILEDEF command before you execute the program, the CMS file FILE OSCAR A1 is created when you execute the program.

Specifying CMS Tape Label Processing

You can use the label operands on the FILEDEF command to indicate that CMS tape label processing is not desired. (This is the default.) If CMS tape label processing is desired, you can use the label operands on the FILEDEF command to indicate the types of labels on your tape.

Specifying Options

The FILEDEF command has many options; those mentioned below are a sampling only. For complete descriptions of all the options of the FILEDEF command, see the *VM/ESA: CMS Command Reference.*

Note: If a SET ERRORMODE CONTINUE command is in effect during Database Services Utility command processing, which requires tape file operation involving multifile volume, the use of the LEAVE option in the FILEDEF may cause a tape positioning error. If a Database Services Utility command processing involving tape file operation fails, the subsequent command processing requiring access to the same tape will get a tape file open error. This error results from the wrong tape positioning caused by the use of the LEAVE option in the FILEDEF.

BLOCK, LRECL, RECFM, DSORG

If you are using the FILEDEF command to relate a data control block (DCB) in a program to an input or output file, you need to supply some of the file format information, such as the record length and block size, on the FILEDEF command line. For example, you have coded a DCB macro for an output file as follows:

```
OUTFILE  DCB   ddname=OUT,MACRF=PM,DSORG=PS
```

When you are issuing a FILEDEF for this *ddname*, you must specify the format of the file. To create an output file on disk, blocked in OS-simulated data set format, you could issue:

```
FILEDEF OUT DISK fn ft A (RECFM FB LRECL 80 BLOCK 1600
```

Note the following command-specific information for the RECFM, BLOCK, and LRECL parameters:

- DATALOAD

  If the DATALOAD input data file contains records with more than 32 760 positions of data, you can do one of the following:

  1. Use VS or VBS records. Specify (as options) only the RECFM and block size (BLOCK or BLKSIZE) parameters in the FILEDEF command defining the data file. (The LRECL specification does not apply and will be overridden if specified.)
  2. Use F or V records if you are using CMS 15 or later, and the DATALOAD input data file contains records with less than 65 536 positions of data.
- UNLOAD DBSPACE and UNLOAD TABLE

  You should always specify a record format (RECFM) of VBS for UNLOAD processing. UNLOAD processing changes the record format to U if the system-required logical record length is greater than the specified block size (BLOCK) value minus 4. Otherwise, UNLOAD processing changes the record format to VB. See below for more information about undefined (U) record format usage.

  A block size greater than 8 244 is recommended for tape output files created by UNLOAD processing.
- UNDEFINED RECORD FORMAT and UNLOAD DBSPACE / UNLOAD TABLE / DATAUNLOAD

  UNLOAD processing changes the RECFM=VBS, specified on the FILEDEF command, to either VB or U. It changes the RECFM to VB if the record length required to unload the data will fit within the specified or default block size (minus 4), so that there are no spanned records. However, when the record length exceeds the block size (minus 4), UNLOAD processing produces spanned records (records that span more than one block). Likewise, DATAUNLOAD processing changes the RECFM=VS or RECFM=VBS, specified on the FILEDEF command, to U, producing spanned records. Note, however, that this VBS-like or VS-like file may NOT be acceptable to other programs that use the OS VBS access method (CMS OS simulation on OS/390 and VSE/ESA), because the logical record length may exceed the OS-defined maximum of 65 535. Each table row is written out as a single spanned record, and tables with very long rows (especially containing long fields) can produce a record exceeding 65 535 positions. This is why UNLOAD / DATAUNLOAD changes VBS or VS to U, since CMS OS simulation cannot handle records longer than 65 535. In that case, the spanning of records is done by the DBS Utility and not by CMS OS simulation.
- RELOAD DBSPACE and RELOAD TABLE

  You should always specify a record format (RECFM) of VBS for RELOAD processing. If a RECFM value other than VBS, or an LRECL value, is specified it is ignored. RELOAD processing changes the record format to VB. Always use the same RECFM and BLKSIZE values on the FILEDEF for a RELOAD as were used on the FILEDEF for the corresponding UNLOAD.
- UNLOAD and RELOAD PROGRAM

If you specify a RECFM other than FB, or specify an LRECL value, the value is ignored.

- SCHEMA

  If you specify a RECFM other than FB, or specify an LRECL value, the value is ignored.

PERM

Usually, when you execute one of the language processors, all existing file definitions are cleared. If the development of a program requires you to recompile and reexecute it frequently, you might want to use the PERM option when you issue file definitions for your input and output files. For example:

```
CP SPOOL PUNCH TO *
FILEDEF INDD DISK TEST FILE A1 (LRECL 80 PERM
FILEDEF OUTDD PUNCH (LRECL 80 PERM
```

In this example, because you spooled your virtual punch to your own virtual card reader, output files are placed in your virtual reader. You can either read or delete them.

All file definitions issued with the PERM option stay in effect until you log off; therefore, specifically clear those definitions or redefine them:

```
FILEDEF INDD CLEAR
FILEDEF OUTDD TAP1 (LRECL 80
```

In the above example, the definition for INDD is cleared; OUTDD is redefined as a tape file.

When you issue the command:

```
FILEDEF * CLEAR
```

all file definitions are cleared, except those you enter with the PERM option.

Note: When a program ends abnormally, or when you issue the HX immediate command, all file definitions are cleared, including those entered with the PERM option.

DISP MOD

Suppose you issue a FILEDEF command for an output file and assign it a CMS file identifier that is identical to that of an existing CMS file; then, when anything is written to that *ddname*, the existing file is replaced by the new output file. If you want, instead, to have new records added to the end of the existing file, you can use the DISP MOD option as follows:

```
FILEDEF ddname DISK fn ft fm (DISP MOD
```

Note: To see the file characteristics used in the Database Services Utility's processing, look at message ARI08681 in the message file.

Notices

Programming Interface Information

This book documents intended Programming Interfaces that allow the customer to write programs to obtain services of DB2 Server for VSE & VM.

Trademarks

Bibliography

This bibliography lists publications that are referenced in this manual or that may be helpful.

DB2 Server for VM Publications

- *DB2 Server for VSE & VM Application Programming*, SC09-2889
- *DB2 Server for VSE & VM Database Administration*, SC09-2888
- *DB2 Server for VSE & VM Database Services Utility*, SC09-2983
- *DB2 Server for VSE & VM Diagnosis Guide and Reference*, LC09-2907
- *DB2 Server for VSE & VM Overivew*, GC09-2995
- *DB2 Server for VSE & VM Interactive SQL Guide and Reference*, SC09-2990
- *DB2 Server for VSE & VM Master Index and Glossary*, SC09-2890
- *DB2 Server for VM Messages and Codes*, GC09-2984
- *DB2 Server for VSE & VM Operation*, SC09-2986
- *DB2 Server for VSE & VM Quick Reference*, SC09-2988
- *DB2 Server for VM System Administration*, SC09-2980
- *DB2 Server for VSE & VM Performance Tuning Handbook*, GC09-2987
- *DB2 Server for VSE & VM SQL Reference*, SC09-2989

DB2 Server for VSE Publications

- *DB2 Server for VSE & VM Application Programming*, SC09-2889
- *DB2 Server for VSE & VM Database Administration*, SC09-2888
- *DB2 Server for VSE & VM Database Services Utility*, SC09-2983
- *DB2 Server for VSE & VM Diagnosis Guide and Reference*, LC09-2907
- *DB2 Server for VSE & VM Overivew*, GC09-2995
- *DB2 Server for VSE & VM Interactive SQL Guide and Reference*, SC09-2990
- *DB2 Server for VSE & VM Master Index and Glossary*, SC09-2890
- *DB2 Server for VSE Messages and Codes*, GC09-2985
- *DB2 Server for VSE & VM Operation*, SC09-2986
- *DB2 Server for VSE System Administration*, SC09-2981
- *DB2 Server for VSE & VM Performance Tuning Handbook*, GC09-2987
- *DB2 Server for VSE & VM SQL Reference*, SC09-2989

Related Publications

- *DB2 Server for VSE & VM Data Restore*, SC09-2991
- *DRDA: Every Manager's Guide*, GC26-3195
- *IBM SQL Reference, Version 2, Volume 1*, SC26-8416
- *IBM SQL Reference*, SC26-8415

VM/ESA Publications

- *VM/ESA: General Information*, GC24-5745
- *VM/ESA: VMSES/E Introduction and Reference*, GC24-5837
- *VM/ESA: Installation Guide*, GC24-5836
- *VM/ESA: Service Guide*, GC24-5838
- *VM/ESA: Planning and Administration*, SC24-5750
- *VM/ESA: CMS File Pool Planning, Administration, and Operation*, SC24-5751
- *VM/ESA: REXX/EXEC Migration Tool for VM/ESA*, GC24-5752
- *VM/ESA: Conversion Guide and Notebook*, GC24-5839
- *VM/ESA: Running Guest Operating Systems*, SC24-5755
- *VM/ESA: Connectivity Planning, Administration, and Operation*, SC24-5756
- *VM/ESA: Group Control System*, SC24-5757
- *VM/ESA: System Operation*, SC24-5758
- *VM/ESA: Virtual Machine Operation*, SC24-5759
- *VM/ESA: CP Programming Services*, SC24-5760
- *VM/ESA: CMS Application Development Guide*, SC24-5761
- *VM/ESA: CMS Application Development Reference*, SC24-5762
- *VM/ESA: CMS Application Development Guide for Assembler*, SC24-5763
- *VM/ESA: CMS Application Development Reference for Assembler*, SC24-5764

- *VM/ESA: CMS Application Multitasking*, SC24-5766
- *VM/ESA: CP Command and Utility Reference*, SC24-5773
- *VM/ESA: CMS Primer*, SC24-5458
- *VM/ESA: CMS User's Guide*, SC24-5775
- *VM/ESA: CMS Command Reference*, SC24-5776
- *VM/ESA: CMS Pipelines User's Guide*, SC24-5777
- *VM/ESA: CMS Pipelines Reference*, SC24-5778
- *VM/ESA: XEDIT User's Guide*, SC24-5779
- *VM/ESA: XEDIT Command and Macro Reference*, SC24-5780
- *VM/ESA: Quick Reference*, SX24-5290
- *VM/ESA: Performance*, SC24-5782
- *VM/ESA: Dump Viewing Facility*, GC24-5853
- *VM/ESA: System Messages and Codes*, GC24-5841
- *VM/ESA: Diagnosis Guide*, GC24-5854
- *VM/ESA: CP Diagnosis Reference*, SC24-5855
- *VM/ESA: CP Diagnosis Reference Summary*, SX24-5292
- *VM/ESA: CMS Diagnosis Reference*, SC24-5857
- CP and CMS control block information is not provided in book form. This information is available on the IBM VM/ESA operating system home page (http://www.ibm.com/s390/vm).
- *IBM VM/ESA: CP Exit Customization*, SC24-5672
- *VM/ESA REXX/VM User's Guide*, SC24-5465
- *VM/ESA REXX/VM Reference*, SC24-5770

C for VM/ESA Publications

- *IBM C for VM/ESA Diagnosis Guide*, SC09-2149
- *IBM C for VM/ESA Language Reference*, SC09-2153
- *IBM C for VM/ESA Compiler and Run-Time Migration Guide*, SC09-2147
- *IBM C for VM/ESA Programming Guide*, SC09-2151
- *IBM C for VM/ESA User's Guide*, SC09-2152

Virtual Storage Extended/Enterprise Systems Architecture (VSE/ESA) Publications

- *IBM VSE/ESA Administration*, SC33-6505
- *IBM VSE/ESA Diagnosis Tools*, SC33-6514
- *IBM VSE/ESA General Information*, GC33-6501
- *IBM VSE/ESA Guide for Solving Problems*, SC33-6510
- *IBM VSE/ESA Guide to System Functions*, SC33-6511
- *IBM VSE/ESA Installation*, SC33-6504
- *IBM VSE/ESA Messages & Codes*, SC33-6507
- *IBM VSE/ESA Networking Support*, SC33-6508
- *IBM VSE/ESA Operation*, SC33-6506
- *IBM VSE/ESA Planning*, SC33-6503
- *IBM VSE/ESA System Control Statements*, SC33-6513
- *IBM VSE/ESA System Macros User's Guide*, SC33-6515
- *IBM VSE/ESA System Macros Reference*, SC33-6516
- *IBM VSE/ESA System Utilities*, SC33-6517
- *IBM VSE/ESA Unattended Node Support*, SC33-6512
- *IBM VSE/ESA Using IBM Workstations*, SC33-6509

CICS/VSE Publications

- *CICS/VSE Application Programming Reference*, SC33-0713
- *CICS/VSE Application Programming Guide*, SC33-0712
- *CICS Application Programming Primer (VS COBOL II)*, SC33-0674
- *CICS/VSE CICS-Supplied Transactions*, SC33-0710
- *CICS/VSE Customization Guide*, SC33-0707
- *CICS/VSE Facilities and Planning Guide*, SC33-0718
- *CICS/VSE Intercommunication Guide*, SC33-0701
- *CICS/VSE Performance Guide*, SC33-0703
- *CICS/VSE Problem Determination Guide*, SC33-0716
- *CICS/VSE Recovery and Restart Guide*, SC33-0702
- *CICS/VSE Release Guide*, GC33-1645
- *CICS/VSE Report Controller User's Guide*, SC33-0705
- *CICS Transaction Server for VSE/ESA V1R1.0 Resource Definition Guide*, SC33-0709
- *CICS/VSE Resource Definition (Online)*, SC33-0708
- *CICS/VSE System Definition and Operations Guide*, SC33-0706
- *CICS/VSE System Programming Reference*, SC33-0711
- *CICS/VSE User's Handbook*, SX33-6079
- *CICS/VSE XRF Guide*, SC33-0704

CICS/ESA Publications

- *CICS/ESA General Information*, GC33-0803

VSE/Virtual Storage Access Method (VSE/VSAM) Publications

- *VSE/VSAM Commands and Macros*, SC33-6532
- *VSE/VSAM Introduction*, GC33-6531
- *VSE/VSAM Messages and Codes*, SC24-5146
- *VSE/VSAM Programmer's Reference*, SC33-6535

VSE/Interactive Computing and Control Facility (VSE/ICCF) Publications

- *VSE/ICCF Administration and Operation*, SC33-6562
- *VSE/ICCF Primer*, SC33-6561
- *VSE/ICCF User's Guide*, SC33-6563

VSE/POWER Publications

- *VSE/POWER Administration and Operation*, SC33-6571
- *VSE/POWER Application Programming*, SC33-6574
- *VSE/POWER Networking*, SC33-6573
- *VSE/POWER Remote Job Entry*, SC33-6572

Distributed Relational Database Architecture (DRDA) Library

- *Application Programming Guide*, SC26-4773
- *Architecture Reference*, SC26-4651
- *Connectivity Guide*, SC26-4783
- *DRDA: Every Manager's Guide*, GC26-3195
- *Planning for Distributed Relational Database*, SC26-4650
- *Problem Determination Guide*, SC26-4782

C/370 for VSE Publications

- *IBM C/370 General Information*, GC09-1386
- *IBM C/370 Programming Guide for VSE*, SC09-1399
- *IBM C/370 Installation and Customization Guide for VSE*, GC09-1417
- *IBM C/370 Reference Summary for VSE*, SX09-1246
- *IBM C/370 Diagnosis Guide and Reference for VSE*, LY09-1805

VSE/REXX Publication

- *VSE/REXX Reference*, SC33-6642

Other Distributed Data Publications

- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 4*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Implementation Programmer's Guide*, SC21-9529
- *VM/Directory Maintenance Licensed Program Specification*, GC20-1836
- *IBM Distributed Relational Database Architecture Reference*, SC26-4651
- *IBM Systems Network Architecture, Format and Protocol Reference*, SC30-3112
- *SNA LU 6.2 Reference: Peer Protocols*, SC31-6808
- *Reference Manual: Architecture Logic for LU Type 6.2*, SC30-3269
- *IBM Systems Network Architecture, Logical Unit 6.2 Reference: Peer Protocols*, SC31-6808
- *Distributed Data Management (DDM) General Information*, GC21-9527

CCSID Publications

- *Character Data Representation Architecture, Executive Overview*, GC09-2207
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

DB2 Server RXSQL Publications

- *DB2 REXX SQL for VM/ESA Installation and Reference*, SC09-2891

C/370 Publications

- *IBM C/370 Installation and Customization Guide*, GC09-1387
- *IBM C/370 Programming Guide*, SC09-1384

Communication Server for OS/2 Publications

- *Up and Running!*, GC31-8189
- *Network Administration and Subsystem Management Guide*, SC31-8181
- *Command Reference*, SC31-8183
- *Message Reference*, SC31-8185
- *Problem Determination Guide*, SC31-8186

Distributed Database Connection Services (DDCS) Publications

- *DDCS User's Guide for Common Servers*, S20H-4793
- *DDCS for OS/2 Installation and Configuration Guide*, S20H-4795

VTAM Publications

- *VTAM Messages and Codes*, SC31-6493
- *VTAM Network Implementation Guide*, SC31-6494
- *VTAM Operation*, SC31-6495
- *VTAM Programming*, SC31-6496
- *VTAM Programming for LU 6.2*, SC31-6497
- *VTAM Resource Definition Reference*, SC31-6498
- *VTAM Resource Definition Samples*, SC31-6499

CSP/AD and CSP/AE Publications
- *Developing Applications*, SH20-6435
- *CSP/AD and CSP/AE Installation Planning Guide*, GH20-6764
- *Administering CSP/AD and CSP/AE on VM*, SH20-6766
- *Administering CSP/AD and CSP/AE on VSE*, SH20-6767
- *CSP/AD and CSP/AE Planning*, SH20-6770
- *Cross System Product General Information*, GH23-0500

Query Management Facility (QMF) Publications
- *Introducing QMF*, GC27-0714
- *Installing and Managing QMF for VSE*, GC27-0721
- *QMF Reference*, SC27-0715
- *Installing and Managing QMF for VM*, GC27-0720
- *Developing QMF Applications*, SC27-0718
- *QMF Messages and Codes*, GC27-0717
- *Using QMF*, SC27-0716

Query Management Facility (QMF) for Windows Publications
- *Getting Started with QMF for Windows*, SC27-0723
- *Installing and Managing QMF for Windows*, GC27-0722

DL/I DOS/VS Publications
- *DL/I DOS/VS Application Programming*, SH24-5009

COBOL Publications
- *VS COBOL II Migration Guide for VSE*, GC26-3150
- *VS COBOL II Migration Guide for MVS and CMS*, GC26-3151
- *VS COBOL II General Information*, GC26-4042
- *VS COBOL II Language Reference*, GC26-4047
- *VS COBOL II Application Programming Guide*, SC26-4045
- *VS COBOL II Application Programming Debugging*, SC26-4049
- *VS COBOL II Installation and Customization for CMS*, SC26-4213
- *VS COBOL II Installation and Customization for VSE*, SC26-4696
- *VS COBOL II Application Programming Guide for VSE*, SC26-4697

Data Facility Storage Management Subsystem/VM (DFSMS/VM) Publications
- *DFSMS/VM RMS User's Guide and Reference*, SC35-0141

Systems Network Architecture (SNA) Publications
- *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084
- *SNA Format and Protocol Reference: Architecture Logic for LU Type 6.2*, SC30-3269
- *SNA LU 6.2 Reference: Peer Protocols*, SC31-6808
- *SNA Synch Point Services Architecture Reference*, SC31-8134

Miscellaneous Publications
- *IBM 3990 Storage Control Planning, Installation, and Storage Administration Guide*, GA32-0100
- *Dictionary of Computing*, ZC20-1699
- *APL2 Programming: Using Structured Query Language*, SH21-1056
- *ESA/390 Principles of Operation*, SA22-7201

Related Feature Publications
- *DB2 for VM Control Center Operations Guide*, GC09-2993
- *DB2 for VSE Control Center Operations Guide*, GC09-2992
- *DB2 Replication Guide and Reference*, SC26-9920

Index

Contacting IBM

Before you contact DB2 customer support, check the product manuals for help with your specific technical problem.

For information or to order any of the DB2 Server for VSE & VM products, contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-237-5511 for customer support
- 1-888-426-4343 to learn about available service options

Product information

DB2 Server for VSE & VM product information is available by telephone or by the World Wide Web at http://www.ibm.com/software/data/db2/vse-vm

This site contains the latest information on the technical library, product manuals, newsgroups, APARs, news, and links to web resources.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) to order products or to obtain general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, go to the IBM Worldwide page at http://www.ibm.com/planetwide

In some countries, IBM-authorized dealers should contact their dealer support structure for information.

File Number: S370/4300-50
Program Number: 5697-F42

Printed in USA

SC09-2983-03

Spine information:

DB2 Server for VSE & VM   Database Services Utility   Version 7 Release 5