

# **INFORMIX®- OnLine/Optical**

## **User Manual**

Version 9.12  
July 1997  
Part No. 000-4079

Published by INFORMIX® Press

Informix Software, Inc.  
4100 Bohannon Drive  
Menlo Park, CA 94025

Copyright © 1981-1997 by Informix Software, Inc. or their subsidiaries, provided that portions may be copyrighted by third parties, as set forth in documentation. All rights reserved.

The following are worldwide trademarks of Informix Software, Inc., or its subsidiaries, registered in the United States of America as indicated by “®,” and in numerous other countries worldwide:

INFORMIX®; INFORMIX®-OnLine Dynamic Server™; DataBlade®

The following are worldwide trademarks of the indicated owners or their subsidiaries, registered in the United States of America as indicated by “®,” and in numerous other countries worldwide:

Adobe Systems Incorporated: PostScript®  
X/OpenCompany Ltd.: UNIX®; X/Open®

All other marks or symbols are registered trademarks or trademarks of their respective owners.

Documentation Team: Bob Berry, Twila Booth, Jennifer Leland

To the extent that this software allows the user to store, display, and otherwise manipulate various forms of data, including, without limitation, multimedia content such as photographs, movies, music and other binary large objects (blobs), use of any single blob may potentially infringe upon numerous different third-party intellectual and/or proprietary rights. It is the user's responsibility to avoid infringements of any such third-party rights.

#### RESTRICTED RIGHTS/SPECIAL LICENSE RIGHTS

Software and documentation acquired with US Government funds are provided with rights as follows: (1) if for civilian agency use, with Restricted Rights as defined in FAR 52.227-19; (2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by negotiated vendor license as prescribed in DFAR 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce the legend.

# Table of Contents

## Introduction

About This Manual . . . . .	3
Organization of This Manual . . . . .	3
Types of Users . . . . .	4
Software Dependencies . . . . .	4
Assumptions About Your Locale. . . . .	4
Demonstration Database . . . . .	5
Major Features . . . . .	5
Documentation Conventions . . . . .	6
Typographical Conventions . . . . .	6
Icon Conventions . . . . .	7
Syntax Conventions . . . . .	8
Sample-Code Conventions. . . . .	14
Additional Documentation . . . . .	14
On-Line Manuals . . . . .	15
Printed Manuals . . . . .	15
Error Message Files . . . . .	15
Documentation Notes, Release Notes, Machine Notes . . . . .	16
Compliance with Industry Standards . . . . .	16
Informix Welcomes Your Comments . . . . .	17

## Chapter 1

### An Overview of OnLine/Optical

The Advantages of Optical Media . . . . .	1-4
Optical Media and Simple Large Objects . . . . .	1-4
Components of the Optical Storage Subsystem . . . . .	1-5
Optical Platter Organization . . . . .	1-6
Storage of Simple Large Objects . . . . .	1-7
Storing Simple Large Objects Sequentially . . . . .	1-7
Storing Simple Large Objects in a Cluster. . . . .	1-7
Placing Simple Large Objects in Optical Clusters . . . . .	1-9

	The OnLine/Optical API . . . . .	1-11
	The Memory Cache and Staging-Area Blobspace . . . . .	1-13
	The API Internal Layer . . . . .	1-17
	The API External Layer . . . . .	1-19
	The OnLine/Optical Message File . . . . .	1-19
<b>Chapter 2</b>	<b>Installation and Initial Configuration</b>	
	Prerequisites . . . . .	2-4
	Installing OnLine/Optical . . . . .	2-5
	Creating the Staging Area . . . . .	2-8
	Naming the Staging Area on the STAGEBLOB Parameter . . . . .	2-8
	Initializing INFORMIX-OnLine/Optical . . . . .	2-8
	Creating the Staging-Area Blobspace . . . . .	2-9
	Verifying the Presence of the Optical Subsystem . . . . .	2-10
	Creating Optical Families . . . . .	2-10
	Testing the Connection . . . . .	2-10
	Test 1 . . . . .	2-10
	Test 2 . . . . .	2-11
<b>Chapter 3</b>	<b>Using OnLine/Optical</b>	
	Assigning Simple-Large-Object Columns to an Optical Platter . . . . .	3-4
	Reading Simple-Large-Object Columns from an Optical Platter . . . . .	3-5
	Clustering Simple Large Objects . . . . .	3-5
	Managing Volumes on the Optical Drives . . . . .	3-9
	Using Simple-Large-Object Descriptors . . . . .	3-12
	Locating Columns Stored in Optical Families . . . . .	3-14
	Locating the Optical Volume Where a Simple Large Object Is Stored . . . . .	3-15
	Migrating a Database with Simple Large Objects on an Optical Platter . . . . .	3-16
<b>Chapter 4</b>	<b>SQL for OnLine/Optical</b>	
	ALTER OPTICAL CLUSTER . . . . .	4-4
	CREATE OPTICAL CLUSTER . . . . .	4-6
	DROP OPTICAL CLUSTER . . . . .	4-10
	RELEASE . . . . .	4-12
	RESERVE . . . . .	4-14
	SET MOUNTING TIMEOUT . . . . .	4-16
	Function Expressions . . . . .	4-17

## Index

# Introduction

About This Manual . . . . .	3
Organization of This Manual . . . . .	3
Types of Users . . . . .	4
Software Dependencies . . . . .	4
Assumptions About Your Locale . . . . .	4
Demonstration Database . . . . .	5
Major Features . . . . .	5
Documentation Conventions . . . . .	6
Typographical Conventions . . . . .	6
Icon Conventions . . . . .	7
Comment Icons . . . . .	7
Syntax Conventions . . . . .	8
Elements That Can Appear on the Path . . . . .	9
How to Read a Syntax Diagram . . . . .	12
Sample-Code Conventions . . . . .	14
Additional Documentation . . . . .	14
On-Line Manuals . . . . .	15
Printed Manuals . . . . .	15
Error Message Files . . . . .	15
Documentation Notes, Release Notes, Machine Notes . . . . .	16
Compliance with Industry Standards . . . . .	16
Informix Welcomes Your Comments . . . . .	17



# R

ead this introduction for an overview of the information provided in this manual and for an understanding of the documentation conventions used.

---

## About This Manual

The *INFORMIX-OnLine/Optical User Manual* describes INFORMIX-OnLine/Optical, a configuration of INFORMIX-Universal Server that supports the storage of TEXT and BYTE data types, also known as *simple large objects*, on optical platters. It describes the OnLine/Optical interface for an optical storage subsystem and the set of SQL statements that you use exclusively with optical platters.

## Organization of This Manual

This manual includes the following chapters:

- This Introduction provides an overview of the manual and describes the documentation conventions used.
- [Chapter 1, “An Overview of OnLine/Optical,”](#) describes the components of an optical storage subsystem and explains how data is organized on optical platters.
- [Chapter 2, “Installation and Initial Configuration,”](#) describes how to install OnLine/Optical, how to create the staging area blob space, and how to initialize the OnLine/Optical system.

- [Chapter 3, “Using OnLine/Optical,”](#) illustrates the use of the SQL extensions that Informix provides to support access to WORM optical media.
- [Chapter 4, “SQL for OnLine/Optical,”](#) provides a reference to the six SQL statements and three special functions that support storage of simple large objects on optical media.

## Types of Users

This manual is written for Universal Server administrators who wish to take advantage of WORM optical media for storage of simple large objects.

If you have limited experience with relational databases, SQL, or your operating system, refer to [Getting Started with INFORMIX-Universal Server](#) for a list of introductory texts.

## Software Dependencies

This manual assumes that you are using INFORMIX-Universal Server, Version 9.1x, as your database server.

In this manual, all instances of Universal Server refer to INFORMIX-Universal Server.

## Assumptions About Your Locale

Informix products can support many languages, cultures, and code sets. All culture-specific information is brought together in a single environment, called a GLS (Global Language Support) locale.

This manual assumes that you are using the default locale, **en\_us.8859-1**. This locale supports U.S. English format conventions for dates, times, and currency. In addition, this locale supports the ISO 8859-1 code set, which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.



If you plan to use nondefault characters in your data or your SQL identifiers, or if you want to conform to the nondefault collation rules of character data, you need to specify the appropriate nondefault locale(s). For instructions on how to specify a nondefault locale, additional syntax, and other considerations related to GLS locales, see the [Guide to GLS Functionality](#).

## Demonstration Database

The DB-Access utility, which is provided with your Informix database server products, includes a demonstration database called **stores7** that contains information about a fictitious wholesale sporting-goods distributor. Sample command files are also included.

Many examples in Informix manuals are based on the **stores7** demonstration database. The **stores7** database is described in detail and its contents are listed in Appendix A of the [Informix Guide to SQL: Reference](#).

The script that you use to install the demonstration database is called **dbaccessdemo7** and is located in the **\$INFORMIXDIR/bin** directory. For a complete explanation of how to create and populate the demonstration database on your database server, refer to the [DB-Access User Manual](#).

---

## Major Features

The Introduction to each Version 9.1x product manual contains a list of major features for that product. The Introduction to each manual in the Version 9.1 *Informix Guide to SQL* series contains a list of new SQL features.

Major features for Version 9.1x Informix products also appear in release notes.

# Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other Informix manuals.

The following conventions are covered:

- Typographical conventions
- Icon conventions
- Syntax conventions
- Sample-code conventions

## Typographical Conventions

This manual uses the following standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	All keywords appear in uppercase letters in a serif font.
<i>italics</i>	Within text, new terms and emphasized words appear in italics. Within syntax diagrams, values that you are to specify appear in italics.
<b>boldface</b>	Identifiers (names of classes, objects, constants, events, functions, program variables, forms, labels, and reports), environment variables, database names, filenames, table names, column names, icons, menu items, command names, and other similar terms appear in boldface.
<code>monospace</code>	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
◆	This symbol indicates the end of feature-, product-, platform-, or compliance-specific information.




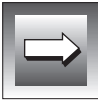

**Tip:** When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.

## Icon Conventions

Throughout the documentation, you will find text that is identified by several different types of icons. This section describes these icons.

### Comment Icons

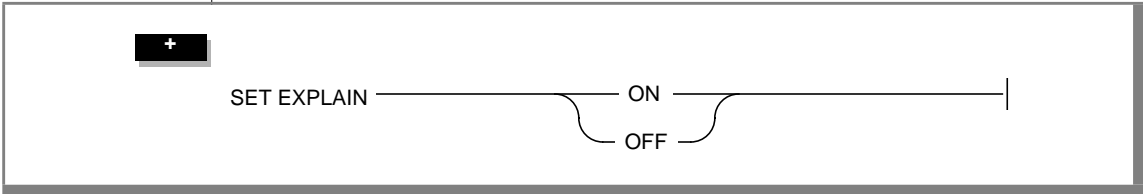
Comment icons identify warnings, important notes, or tips. This information is always displayed in italics.

Icon	Description
	The <i>warning</i> icon identifies vital instructions, cautions, or critical information.
	The <i>important</i> icon identifies significant information about the feature or operation that is being described.
	The <i>tip</i> icon identifies additional details or shortcuts for the functionality that is being described.

## Syntax Conventions

This section describes conventions for syntax diagrams. Each diagram displays the sequences of required and optional keywords, terms, and symbols that are valid in a given statement or segment, as Figure 1 shows.

**Figure 1**  
*Example of a Simple Syntax Diagram*











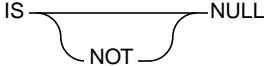
Each syntax diagram begins at the upper-left corner and ends at the upper-right corner with a vertical terminator. Between these points, any path that does not stop or reverse direction describes a possible form of the statement.

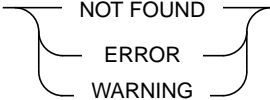
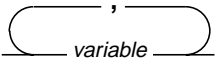
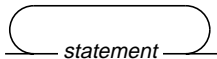
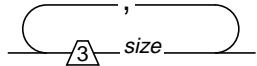

Syntax elements in a path represent terms, keywords, symbols, and segments that can appear in your statement. The path always approaches elements from the left and continues to the right, except in the case of separators in loops. For separators in loops, the path approaches counterclockwise from the right. Unless otherwise noted, at least one blank character separates syntax elements.

Elements That Can Appear on the Path

You might encounter one or more of the following elements on a path.

Element	Description
KEYWORD	A word in UPPERCASE letters is a keyword. You must spell the word exactly as shown; however, you can use either uppercase or lowercase letters.
( , ; @ + * - / )	Punctuation and other nonalphanumeric characters are literal symbols that you must enter exactly as shown.
' '	Single quotes are literal symbols that you must enter as shown.
<i>variable</i>	A word in <i>italics</i> represents a value that you must supply. A table immediately following the diagram explains the value.
<div>ADD Clause p. 1-14</div> <div>ADD Clause</div>	A reference in a box represents a subdiagram. Imagine that the subdiagram is spliced into the main diagram at this point. When a page number is not specified, the subdiagram appears on the same page.
<div>Table Name see SQLS</div>	A reference to SQLS in this manual refers to the <i>Informix Guide to SQL: Syntax</i> . Imagine that the subdiagram is spliced into the main diagram at this point.
<div>E/C</div>	<p>An icon is a warning that this path is valid only for some products, or only under certain conditions. Characters on the icons indicate what products or conditions support the path.</p> <p>These icons might appear in a syntax diagram:</p> <div><div>D/B</div> This path is valid only for DB-Access.</div> <div><div>E/C</div> This path is valid only for INFORMIX-ESQL/C.</div> <div><div>EXT</div> This path is valid for external routines.</div>

Element	Description
	This path is valid only if you are using Informix Stored Procedure Language (SPL).
	This path is valid for the SQL Editor.
	This path is valid only for INFORMIX-OnLine/Optical.
	This path is an Informix extension to ANSI SQL-92 entry-level standard SQL. If you initiate Informix extension checking and include this syntax branch, you receive a warning. If you have set the <b>DBANSIWARN</b> environment variable at compile time, or have used the <b>-ansi</b> compile flag, you receive warnings at compile time. If you have <b>DBANSIWARN</b> set at runtime, or if you compiled with the <b>-ansi</b> flag, warning flags are set in the <b>sqlwarn</b> structure.
	This path is valid only if your database or application uses a nondefault GLS locale.
	A shaded option is the default action.
	Syntax that is enclosed between a pair of arrows is a subdiagram.
	The vertical line terminates the syntax diagram.
	A branch below the main path indicates an optional path. (Any term on the main path is required, unless a branch can circumvent it.)

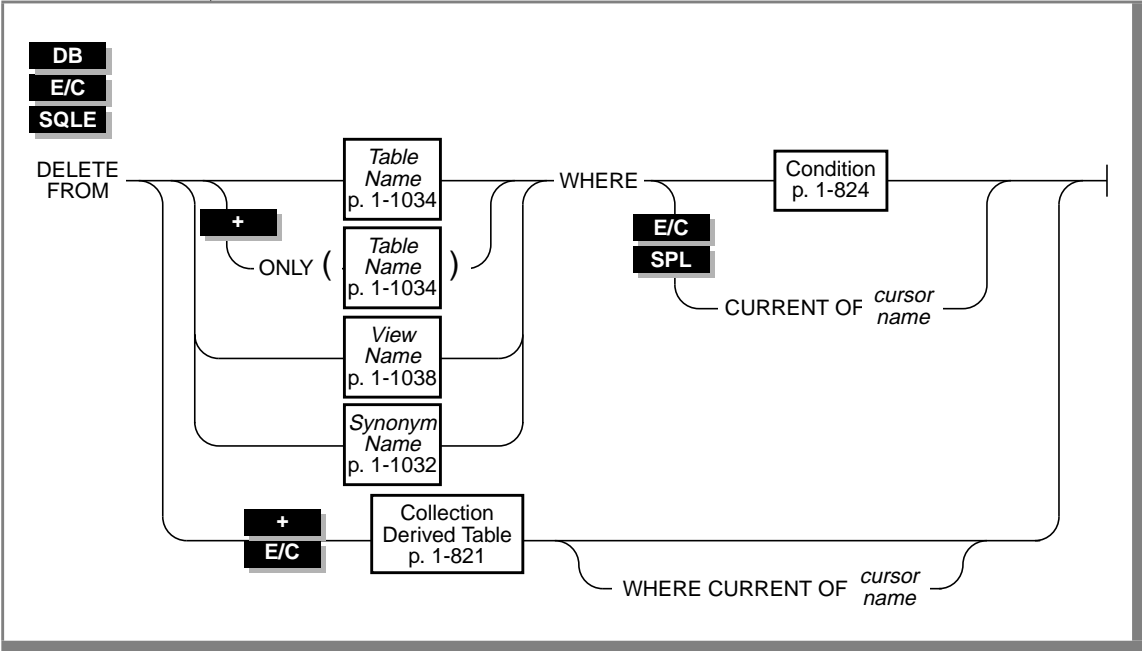
Element	Description
	A set of multiple branches indicates that a choice among more than two different paths is available.
 	A loop indicates a path that you can repeat. Punctuation along the top of the loop indicates the separator symbol for list items. If no symbol appears, a blank space is the separator.
	A gate (  ) on a path indicates that you can only use that path the indicated number of times, even if it is part of a larger loop. Here you can specify <i>size</i> no more than three times.

(3 of 3)

**How to Read a Syntax Diagram**

Figure 2 shows a syntax diagram that uses many of the elements that are listed in the previous table.

**Figure 2**  
*Example of a Syntax Diagram*



The three icons at the top left of this diagram indicate that you can construct this statement if you are using DB-Access, ESQL/C, or the SQL Editor. To use the diagram to construct a statement, begin at the far left with the keywords **DELETE FROM**. Then follow the diagram to the right, proceeding through the options that you want.



**To construct a DELETE statement**

1. You must type the words `DELETE FROM`.
2. If you are using DB-Access, ESQ/C, or the SQL Editor, you can delete a table, view, or synonym:
  - Follow the diagram by typing the table name, view name, or synonym, as desired. Refer to the appropriate segment for available syntax options.
  - You must type the keyword `WHERE`.
  - If you are using DB-Access or the SQL Editor, you must include the Condition clause to specify a condition to delete. To find the syntax for deleting a condition, go to the “Condition” segment on page 1-803.
  - If you are using ESQ/C or SPL, you can include either the Condition clause to delete a specific condition or the `CURRENT OF cursorname` clause to delete a row from the table.
3. If you are using ESQ/C, you can also choose to delete from a collection-derived table:
  - Follow the diagram by going to the segment “Collection Derived Table” on page 1-800. Follow the syntax for the segment.
  - You can stop, taking the direct route to the terminator, or you can include the `WHERE CURRENT OF cursorname` clause to delete a row from a collection-derived table.
4. Follow the diagram to the terminator. Your `DELETE` statement is complete.

## Sample-Code Conventions

Examples of SQL code occur throughout this manual. Except where noted, the code is not specific to any single Informix application development tool. If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores7
:
:
DELETE FROM customer
      WHERE customer_num = 121
:
:
COMMIT WORK
DISCONNECT CURRENT
```

To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using the Query-language option of DB-Access, you must delimit multiple statements with semicolons. If you are using an SQL API, you must use EXEC SQL at the start of each statement and a semicolon (or other appropriate delimiter) at the end of the statement.



***Tip:** Ellipsis points in a code example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.*

For detailed directions on using SQL statements for a particular application development tool or SQL API, see the manual for your product.

---

## Additional Documentation

For additional information, you might want to refer to the following types of documentation:

- On-line manuals
- Printed manuals
- Error message files
- Documentation notes, release notes, and machine notes

## On-Line Manuals

A CD that contains Informix manuals in electronic format is provided with your Informix products. You can install the documentation or access it directly from the CD. For information about how to install, read, and print on-line manuals, see either the installation guide for your product or the installation insert that accompanies the documentation CD.

The documentation set that is provided on the CD describes Universal Server, its implementation of SQL, and its associated application-programming interfaces. For an overview of the manuals in the Universal Server documentation set, see [Getting Started with INFORMIX-Universal Server](#).

## Printed Manuals

The Universal Server documentation set describes Universal Server, its implementation of SQL, and its associated application-programming interfaces. For an overview of the manuals in the Universal Server documentation set, see [Getting Started with INFORMIX-Universal Server](#).

To order printed manuals, call 1-800-331-1763 or send email to [moreinfo@informix.com](mailto:moreinfo@informix.com).

Please provide the following information:

- The documentation that you need
- The quantity that you need
- Your name, address, and telephone number

## Error Message Files

Informix software products provide ASCII files that contain all the Informix error messages and their corrective actions. To read the error messages in the ASCII file, Informix provides scripts that let you display error messages on the screen (**finderr**) or print formatted error messages (**rofferr**). For a detailed description of these scripts, see the Introduction to the [Informix Error Messages](#) manual.

## Documentation Notes, Release Notes, Machine Notes

In addition to printed documentation, the following on-line files, located in the `$INFORMIXDIR/release/en_us/0333` directory, supplement the information in this manual.

On-Line File	Purpose
<b>OPTICALDOC_9.1</b>	The documentation-notes file describes features that are not covered in this manual or that have been modified since publication.
<b>SERVERS_9.1</b>	The release-notes file describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.
<b>IUNIVERSAL_9.1</b>	The machine-notes file describes any special actions that are required to configure and use Informix products on your computer. Machine notes are named for the product described.

Please examine these files because they contain vital information about application and performance issues.

## Compliance with Industry Standards

The American National Standards Institute (ANSI) has established a set of industry standards for SQL. Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992, on INFORMIX-Universal Server. In addition, many features of Universal Server comply with the SQL-92 Intermediate and Full Level and X/Open SQL CAE (common applications environment) standards.

---

## **Informix Welcomes Your Comments**

Please tell us what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about corrections or clarifications that you would find useful. Include the following information:

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

Write to us at the following address:

Informix Software, Inc.  
SCT Technical Publications Department  
4100 Bohannon Drive  
Menlo Park, CA 94025

If you prefer to send email, our address is:

`doc@informix.com`

Or send a facsimile to the Informix Technical Publications Department at:

415-926-6571

We appreciate your feedback.



# An Overview of OnLine/Optical

The Advantages of Optical Media. . . . .	1-4
Optical Media and Simple Large Objects . . . . .	1-4
Components of the Optical Storage Subsystem . . . . .	1-5
Optical Platter Organization. . . . .	1-6
Storage of Simple Large Objects . . . . .	1-7
Storing Simple Large Objects Sequentially . . . . .	1-7
Storing Simple Large Objects in a Cluster. . . . .	1-7
Placing Simple Large Objects in Optical Clusters . . . . .	1-9
The Simple-Large-Object Column . . . . .	1-9
The Cluster-Key Column . . . . .	1-9
How Optical Clustering Occurs. . . . .	1-10
The OnLine/Optical API. . . . .	1-11
The Memory Cache and Staging-Area BlobSpace . . . . .	1-13
Memory Cache . . . . .	1-14
The Staging Area . . . . .	1-15
The STAGEBLOB Parameter . . . . .	1-16
The onstat -O option. . . . .	1-16
The API Internal Layer . . . . .	1-17
The API External Layer . . . . .	1-19
The OnLine/Optical Message File . . . . .	1-19







**I**NFORMIX-OnLine/Optical is a configuration of INFORMIX-Universal Server that supports the storage of TEXT and BYTE data, also known as simple large objects, on an optical platter.

**Important:** Version 9.1x of OnLine/Optical uses the term “simple large object” for the BYTE and TEXT data types that are known as “blobs” in previous versions of OnLine/Optical.

For more information on the TEXT and BYTE data types, refer to the [Informix Guide to SQL: Reference](#).

OnLine/Optical includes an interface for an optical storage subsystem and supports a set of SQL statements that are exclusively for use with optical platters. These SQL statements support the efficient storage and retrieval of data to and from the optical storage subsystem.

OnLine/Optical also includes all the features and capabilities of INFORMIX-Universal Server.



**Important:** OnLine/Optical does not store the Universal Server CLOB (Character Large object) and BLOB (Binary Large Object) data types, also referred to as smart large objects. Smart large objects use a file interface to support the optical storage subsystem. For more information on how to store CLOB and BLOB data types on optical disc, refer to the “[INFORMIX-ESQL/C Programmer's Manual](#).”

This chapter describes the purpose and implementation of OnLine/Optical. The following topics are covered:

- The advantages offered by optical media
- The components of an optical storage subsystem
- How data is organized on optical platters
- The sequential and clustered methods of storing TEXT and BYTE data on optical platters
- The operation of the OnLine/Optical application-programming interface (API), which supports the optical storage subsystem

---

## The Advantages of Optical Media

Optical media offer the following advantages for storing data over conventional magnetic disks:

- Mass storage capacity (on the order of gigabytes)
- Mountable/unmountable storage units
- Low cost per bit of storage
- Long media life
- High data stability

Because optical media are usually written to and read with a laser, a bit of data is written to a much smaller area on the optical *platter* (disc) than the area that is required to record a bit on a conventional magnetic disk. Optical platters offer a very high degree of stability because they are far less susceptible than magnetic disks to corruption by environmental influences such as electromagnetic fields. Furthermore, the possibility of a *head crash* is extremely remote because the optical disc read/write head does not come as close to the platter as the magnetic disk read/write head.

OnLine/Optical supports only the write-once-read-many (WORM) type of optical media. When data is written to WORM optical media, a permanent, virtually incorruptible mark is made on the platter. Thereafter, the data can be read an unlimited number of times. When data is updated to a WORM optical platter, it is written to a new location on the platter; the original location cannot be reused.

## Optical Media and Simple Large Objects

In a database environment, the vast storage capacity of optical media make it particularly attractive for storing simple-large-object data such as text documents, source and object programs, and scanned images. Simple large objects theoretically have no maximum size, although in practice OnLine/Optical allows a maximum size of about 2 gigabytes per simple large object. The number of locks available in the system may impose an additional constraint on the size.

The trade-off for the capacity, lower cost, integrity, and security that optical media provide, however, is slower access. In general, accessing optical media is slower than accessing magnetic disks. Typically, an optical drive accesses data at speeds that are 25 to 50 percent slower than a magnetic disk drive.

---

## Components of the Optical Storage Subsystem

The optical storage subsystem has the following components:

- The platter
- The jukebox (autochanger) or stand-alone drive
- The shelf
- Subsystem management software

The optical media are the removable optical platters that contain data.

Optical platters can be kept in an automated library called a *jukebox* or *autochanger*. The jukebox is a cabinet that contains one or more optical platter drives, slots that store the optical platters when they are not mounted, and a robotic arm that transfers platters between the slots and the drives.

Subsystem commands initiate transport of the platters to and from the drives and manage the input/output (I/O) operations within the subsystem.

Jukeboxes are not essential to the subsystem. A supported subsystem can consist of a stand-alone optical drive that holds only one platter, or several stand-alone drives.

Optical platters can be stored off-line on *shelves*. When a platter is known to the subsystem, but is not stored in the jukebox where it can be accessed, the platter is said to be *on the shelf*. The operator is responsible for storage and retrieval of platters to and from the shelf.

Subsystem management software performs various tasks related to the subsystem. It creates optical families and volumes and tracks information about them, chooses which optical drives to use and the optical platters to mount, issues commands to the jukebox, and reads and writes data to the optical platter.

## Optical Platter Organization

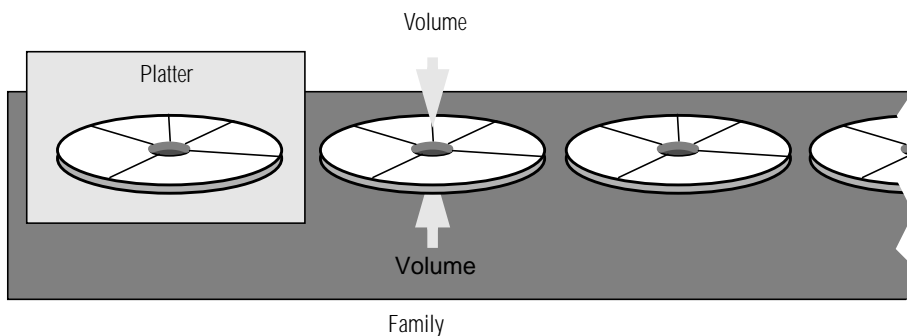
In a WORM optical subsystem supported by OnLine/Optical, the optical media are organized into units of storage called the *family* and the *volume*. Each side of the removable platter is called a volume. An optical family is theoretically an unlimited collection of volumes, although in practice the constraints of the subsystem impose a limit on its size. When a volume becomes full, the subsystem automatically allocates another volume from the family.

The optical storage administrator creates an optical family with a utility that the subsystem vendor provides. The administrator also uses subsystem commands to add volumes to a family.

In a manner that is similar to an automated tape library, the optical subsystem tracks the location of each simple large object on a volume, each volume on a platter, and each platter in a family. Subsystem software schedules the optical drives and mounts platters as needed.

Figure 1-1 shows the relationship between the optical family, the volume, and the optical platter.

**Figure 1-1**  
*The Optical Family, the Volume, and the Optical Platter*



---

## Storage of Simple Large Objects

You assign a TEXT or BYTE column to the optical storage subsystem with the CREATE TABLE statement, which allows you to place the column data in an optical family. Prior to executing the CREATE TABLE statement, the subsystem administrator must create an optical family name with a utility provided by the subsystem vendor. You can then use the column-definition portion of the CREATE TABLE statement to assign the column to the optical family. For more information, see [“Assigning Simple-Large-Object Columns to an Optical Platter” on page 3-4](#).

Simple large objects stored on optical media are not backed up (or archived) by the OnLine/Optical backup and restore utilities. Consequently, you cannot restore them from backup media.

OnLine/Optical can store simple large objects on an optical volume either sequentially or in a cluster. The following sections describe each of these methods.

### Storing Simple Large Objects Sequentially

Sequential storage means that OnLine/Optical stores simple large objects on the current volume of the family in the same sequence that they are inserted. When a volume becomes full, the subsystem allocates the next volume in the family; no attempt is made to keep logically related large objects together on the same volume or even on the same platter.

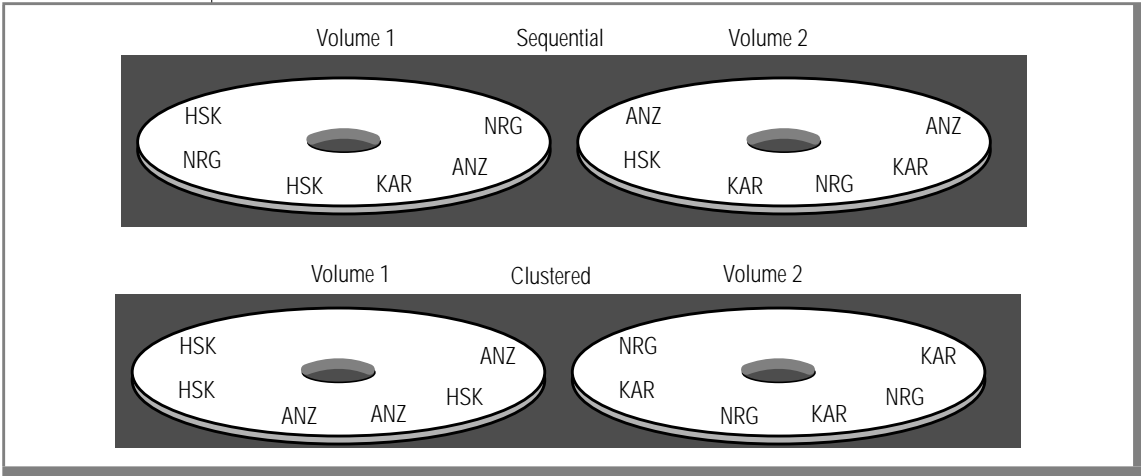
### Storing Simple Large Objects in a Cluster

Optical clustering attempts to store logically related large objects together on the same volume. In applications where related large objects are often retrieved together, optical clustering minimizes time-consuming platter exchanges on the drives. This practice improves performance.

Optical clustering does not imply that large objects that share a particular key value are stored next to each other on the optical platter. Instead, each optical cluster represents a reservation of some number of kilobytes (the clustersize) on a volume. However, the reserved kilobytes and the large objects that are eventually stored there might not be contiguous. The benefit of clustering is that it is far faster to find related large objects on the same volume than it is to find them on separate volumes or platters.

Figure 1-2 illustrates the difference between sequential and clustered organization. In this example, the optical platters store large objects that are scanned pictures of merchandise. The pictures belong to the **cat\_desc** column in the **stores7** database. They are represented by another column in the table, a manufacturer code, because the pictures do not inherently provide any logical ordering. When stored sequentially, the pictures are written to the optical platter in the same order that they are inserted. When they are clustered by the manufacturer code (the cluster key), the pictures for a given manufacturer are stored together. Two volumes are shown to illustrate how clustering reduces platter exchanges on the optical drives.

**Figure 1-2**  
*Sequential Storage Versus Clustering by manu\_code*



## Placing Simple Large Objects in Optical Clusters

You create an optical cluster with the `CREATE OPTICAL CLUSTER` statement. For information on the syntax of the `CREATE OPTICAL CLUSTER` statement, refer to “[CREATE OPTICAL CLUSTER](#)” on page 4-6. As part of the syntax, you define the cluster by specifying the following column lists:

*column list* is one or more logically related simple-large-object columns from a table. These are the columns that you want to store together in the same cluster.

*cluster-key column list* is one or more columns that provide a logical order for the simple-large-object columns. The cluster-key columns must be from the same table as the simple large object columns.

For example, suppose you are a sporting-goods distributor and you have photographs taken of your stock to include in an electronic catalog. With OnLine/Optical, you create a table called **catalog**, which is similar to the **catalog** table provided with the **stores7** database but which stores the **cat\_picture** column on optical platter rather than magnetic disk.

### *The Simple-Large-Object Column*

In the `CREATE TABLE` statement for **catalog**, you define a `BYTE` column called **cat\_picture** and use the `IN` portion of the column definition to place the column in the optical family **stock\_photos**. For the complete syntax of the `CREATE TABLE` statement, see the [Informix Guide to SQL: Syntax](#).

### *The Cluster-Key Column*

Next, you decide that you want to keep the photographs for a given manufacturer together to avoid unnecessary platter exchanges during retrieval. To do this, you use the `CREATE OPTICAL CLUSTER` statement and make the **manu\_code** column (also a column in the **catalog** table) the *cluster-key* column. Because **manu\_code** contains nine unique values (ANZ, HRO, HSK, KAR, NKL, NRG, PRC, SHM, SMT), the photographs are stored in nine unique clusters, each containing the photographs for a single manufacturer. You assign a cluster size of 18,000 kilobytes to the cluster for the following reason: each photo requires 60 kilobytes of storage, and you want to allow for as many as 300 photos per manufacturer.

### ***How Optical Clustering Occurs***

Insert the first photo into the database. As the insert operation begins, OnLine/Optical checks whether the large object is stored on an optical platter and whether an optical cluster exists for it. If so, OnLine/Optical uses an internal optical-cluster table to find the current volume for the cluster-key value. If the cluster-key (**manu\_code**) value for the first insert is HSK, OnLine/Optical asks the subsystem to reserve 18,000 kilobytes on the current volume for this cluster key. This first photo is then written on to the volume.

OnLine/Optical adds the cluster-key value, HSK, and the location of the HSK cluster, to the cluster-size information that is already recorded in its internal optical-cluster table. OnLine/Optical also records the fact that 60 kilobytes in the HSK cluster are now used. When the next unique cluster-key value, NRG for example, is inserted, the subsystem reserves another 18,000 kilobytes on the current volume for large objects associated with this cluster key. This process is repeated each time the subsystem encounters a unique cluster-key value.

As HSK and NRG photos fill their respective clusters, the internal optical cluster table tracks the amount of space used in each cluster. Before a photo is written to optical platter, OnLine/Optical compares the size of the object with the amount of space remaining in its assigned cluster. If the cluster is too full to receive the photo, the subsystem creates a new cluster on the volume.

If all space on the current volume is filled or reserved, the subsystem finds the next available volume in the family (a stand-alone subsystem prompts you to mount an empty platter), and creates a new cluster on it for the cluster-key value. When the new cluster is created, the internal optical-cluster table is updated with the new volume number and the new value for space remaining in the current cluster. If an additional volume is not available in the family, an error is returned.



---

## The OnLine/Optical API

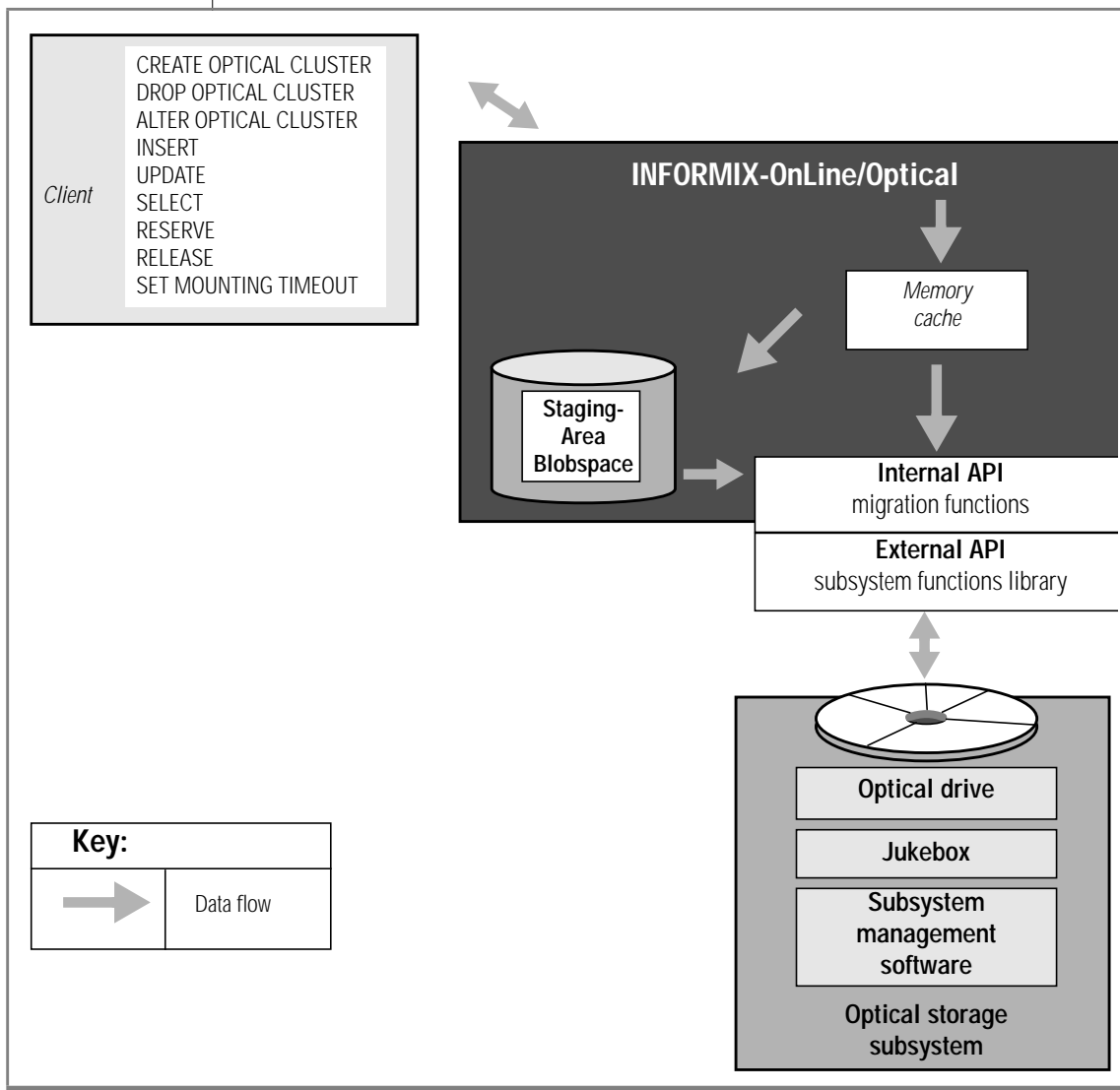
The API between OnLine/Optical and the optical storage subsystem consists of the following layers:

- An *internal* layer that represents the interaction between OnLine/Optical and the API
- An *external* layer that represents the interaction between the API and the optical storage subsystem

The subsystem vendor provides the external layer of the interface in the form of a library that is linked into OnLine/Optical during installation. For information on installing the vendor library and linking it with OnLine/Optical, see [Chapter 2, “Installation and Initial Configuration.”](#)

Figure 1-3 illustrates the relationships between OnLine/Optical, the API, and the optical storage subsystem.

**Figure 1-3**  
*INFORMIX-OnLine/Optical, the API, the Staging-Area Blobspace, the Memory Cache, and the Optical Storage Subsystem*



## **The Memory Cache and Staging-Area BlobSpace**

The memory cache improves the performance of writing simple large objects that are smaller than the cache size. OnLine/Optical receives simple large objects from the client application in 1-kilobyte pieces for a single row at a time. If the memory cache is full and cannot hold all pieces of a large object, or if the cache is not being used, then OnLine/Optical writes the large object to the staging-area blobSpace. OnLine/Optical uses the same memory cache for all client applications that are writing simple large objects.

The system configuration parameter **OPCACHEMAX**, located in the **ONCONFIG** file, specifies the size of the memory cache in kilobytes. If the value is 0, OnLine/Optical does not use the cache. You can set the client environment variable, **INFORMIXOPCACHE**, to restrict the amount of memory cache that the client application uses. The value for **INFORMIXOPCACHE** is specified in kilobytes and can be any value less than or equal to **OPCACHEMAX**. If **INFORMIXOPCACHE** is not set, the client application can use as much of the memory cache as is available.

OnLine/Optical stores all simple large objects for a given row in the memory cache until the last large object in the row has been received or until the cache is filled. If a simple large object in the row does not fit into the memory cache, it is flushed out of the cache into the staging-area blobSpace. If the next large object in the row fits into the available space in the memory cache, it stays there.

For example, imagine that you have a memory cache of 150 kilobytes, that the setting for the configuration parameter **OPCACHEMAX** is 150 kilobytes, and that the environment variable **INFORMIXOPCACHE** is not set. Your current row has three large objects of the following sizes: 30 kilobytes, 180 kilobytes, and 70 kilobytes. The first large object is 30 kilobytes and OnLine/Optical writes it to the memory cache. The second large object is 180 kilobytes and OnLine/Optical tries to use the memory cache but the large object is too large. OnLine/Optical writes this large object to the staging-area blobSpace. The second large object is 70 kilobytes and fits in the memory cache. OnLine/Optical now outmigrates all the large objects in the row to the optical subsystem in 50-kilobyte pieces in the order the objects were processed: the first large object from the memory cache; the second large object from the staging-area blobSpace; and the third large object from the memory cache.

If you want to outmigrate a different number of bytes than the default of 50 kilobytes, you can configure the outmigration amount using the storage manager for the optical storage subsystem. [Figure 1-3 on page 1-12](#) illustrates the use of the memory cache and the staging area when simple large objects are outmigrated to the subsystem.

You create the staging-area blobSpace through ON-Monitor, or with the **onspaces** utility, in the same way as any other OnLine/Optical blobSpace. Before you use the optical storage subsystem, you must perform the following tasks:

- Use ON-Monitor or a text editor to edit the ONCONFIG file and set the STAGEBLOB parameter to the name of the staging-area blobSpace.
- Create the staging-area blobSpace.
- Restart **oninit** to enable OnLine/Optical to recognize the optical subsystem.

Before you use the optical storage subsystem, you might also want to perform the following tasks:

- Adjust the configuration parameter OPCACHEMAX to something other than the default (128 kilobytes).
- Specify the size for the **INFORMIXOPCACHE** environment variable in client environments.

## ***Memory Cache***

Memory for the memory cache is allocated as it is used. If OPCACHEMAX is set at 500 (500 kilobytes) and the largest simple large object is 50 kilobytes, OnLine/Optical uses only 50 kilobytes for the memory cache.

Because the setting for the configuration parameter OPCACHEMAX is system wide, 100 percent of the memory cache is available if you are the only user. If additional users compete for use of the memory cache, more large objects are written to the staging-area blobSpace. To properly assess the use of the memory cache, look at how many applications are processing simple large objects and set the OPCACHEMAX to accommodate the largest large object.

The system configuration parameter OPCACHEMAX uses a default value of 128 kilobytes; however, you can set OPCACHEMAX to any value. If OPCACHEMAX is set to 0, the memory cache is not used and OnLine/Optical writes all simple large objects to the staging-area blobSpace.

### ***The Staging Area***

The structure of the staging-area blobSpace is the same as all other OnLine/Optical blobSpaces. When it is created, the staging area consists of only one chunk, but more can be added as desired. BlobSpace free-map pages manage the space. For information on how to create a blobSpace, see the [\*INFORMIX-Universal Server Administrator's Guide\*](#).

The optimal size for the staging-area blobSpace depends on the following application factors:

- The frequency of simple-large-object storage
- The frequency of simple-large-object retrieval
- The average size of the simple large objects

To calculate the size of the staging-area blobSpace, you must estimate the number of large objects that you expect to reside there simultaneously and multiply that number by the average size. In a single transaction, all simple large objects that move to the staging area are held there until the transaction completes. The number of large objects that reside in the staging area simultaneously thus depends on the number of rows being inserted at one time. As a minimum, the staging-area blobSpace must be at least as large as the largest simple large object that will reside there. If OnLine/Optical uses the staging-area blobSpace, it writes an entire large object to the staging area before it outmigrates the object to the optical subsystem.

If a hardware failure occurs in the staging-area blobSpace, OnLine/Optical rolls the transaction back. OnLine/Optical does not commit the transaction until it has written the large object to the optical platter.

The staging-area blobSpace cannot be mirrored.

**The STAGEBLOB Parameter**

You can either use ON-Monitor or a text editor to edit the ONCONFIG file and set the STAGEBLOB parameter with the name of the staging-area blobSpace. The following example shows the format of the STAGEBLOB parameter:

```
STAGEBLOB blobSpace_name
```

The *blobSpace\_name* variable. For more information on setting the STAGEBLOB parameter, refer to the [INFORMIX-Universal Server Administrator's Guide](#).

The presence of the STAGEBLOB parameter in ONCONFIG file signals OnLine/Optical that an optical subsystem is present. If OnLine/Optical does not find the STAGEBLOB parameter in the ONCONFIG file, it behaves as if there were no optical storage subsystem. If the STAGEBLOB parameter is present but you have not created the staging-area blobSpace, OnLine/Optical displays the following message:

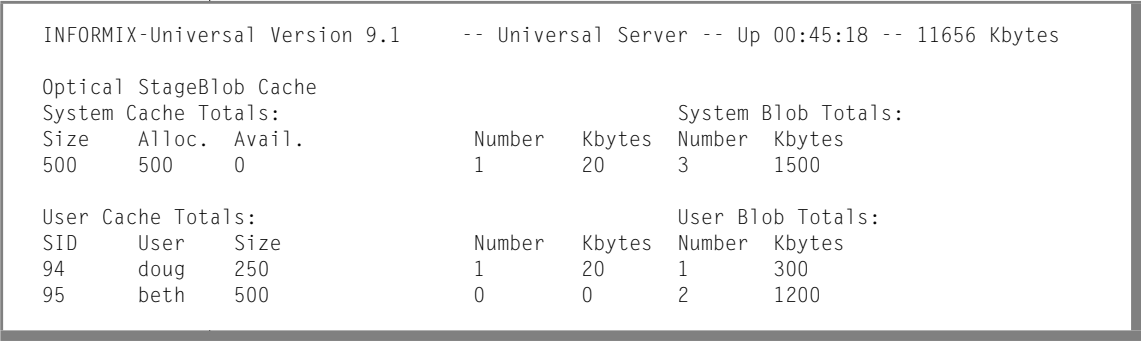
```
Invalid or missing name for Subsystem Staging BlobSpace
```

You must create the staging-area blobSpace and reinitialize OnLine/Optical before it can migrate simple large objects to the subsystem. For the procedure to create the staging-area blobSpace, refer to [Chapter 2, "Installation and Initial Configuration."](#)

**The onstat -O option**

You can use the -O option of the **onstat** utility to monitor available and allocated resources for the memory cache and the staging-area blobSpace. Figure 1-4 shows a sample of **onstat -O** output.

**Figure 1-4**  
Example of onstat -O Option



The example shows that Doug has written one 20-kilobyte large object to the memory cache. He also tried to write a 300-kilobyte large object, but it did not fit because he is limited to 250 kilobytes of the memory cache. Beth can use the entire 500-kilobyte memory cache, but the two simple large objects that she tried to write did not fit into the memory cache (1,200 kilobytes) and were written to the staging-area blob space. No optical writes are in progress in [Figure 1-4 on page 1-16](#) (Avail = 0). No optical work was completed except for these two programs because the totals are equal to the sum of the two user entries (1,500 kilobytes). You can tell that the two programs are still connected because the entries are visible.

## The API Internal Layer

The internal layer of the API manages the following operations:

- An *outmigration* operation that stores simple large objects in the subsystem
- An *inmigration* operation that retrieves simple large objects from the subsystem

*Outmigration* is the operation through which OnLine/Optical directs the subsystem to store a simple large object on an optical platter. Outmigration activity occurs in the following sequence:

1. OnLine/Optical stores the large object in the memory cache in 1-kilobyte pieces.
2. OnLine/Optical stores large-object data that exceeds the memory cache in the staging-area blob space.
3. OnLine/Optical makes the large-object size and cluster information available to the subsystem.
4. The subsystem finds and reserves space to store the simple large object.
5. OnLine/Optical transfers the simple large object to the optical subsystem.

6. The optical subsystem signals the end of the simple-large-object migration. Once the outmigration process ends, the staging-area blobpages that held the migrating large object are marked *free*. In addition, the memory cache is flushed and available for use.
7. The family number, volume number, and byte-offset used by the subsystem to describe the storage location are passed back to OnLine/Optical, where the information is stored in the simple-large-object descriptor in the data row.

Because data cannot be rewritten to the same location on a WORM optical platter, when OnLine/Optical updates a simple large object, it outmigrates the large object to a new, clean location on the platter and updates the descriptor to reflect the new location. The subsystem does not have the capability to reclaim space on the WORM platter after a large object is modified or deleted.

*Immigration* is the operation through which OnLine/Optical requests and gains access to a large object stored in the subsystem. Immigration activity occurs in the following sequence:

1. OnLine/Optical requests access to a large object by supplying the optical subsystem with the blob family name, volume number, and byte offset.
2. The subsystem locates the large object and retrieves it in pieces. The subsystem deposits each portion of the large object into memory buffers that OnLine/Optical can access.
3. The subsystem signals the end of optical data migration.



## The API External Layer

The external layer of the API consists of a set of low-level functions that implement the interaction between the API and the subsystem. These functions are compiled into a library called **libop.a** that the subsystem vendor provides. The installation process links the **libop.a** library with OnLine/Optical.

## The OnLine/Optical Message File

Under certain exceptional conditions, the OnLine/Optical API writes messages to the message file. In general, the messages indicate that OnLine/Optical made a request of the optical subsystem, and the subsystem was unable to accomplish it. See your subsystem vendor documentation for the content of these messages and an appropriate action or response, if one is required. The MSGPATH variable in the ONCONFIG file specifies the UNIX pathname of the system message log file. The default value for MSGPATH is **/usr/informix/universal.log**. For more information about the message log file, refer to the [\*INFORMIX-Universal Server Administrator's Guide\*](#).

The following messages concerning the optical subsystem are written to the message log file by OnLine/Optical:

```
oninit: invalid or missing name for Subsystem Staging  
BlobSpaceOptical Subsystem is running
```

**Cause:** STAGEBLOB is specified in the configuration file, but the named dbspace does not exist. This is a normal message the first time **oninit** is started with OnLine/Optical.

**Action:** Use onspaces with the **-d** option to create the dbspace.

**Cause:** You have set the value of the STAGEBLOB parameter in the configuration file, and OnLine/Optical is communicating properly with the optical storage subsystem.

**Action:** None required.

```
Optical Subsystem is not running
```

**Cause:** You have set the value of the STAGEBLOB parameter in the configuration file, but OnLine/Optical cannot detect the existence of the optical storage subsystem.

**Action:** Check that the optical subsystem is on-line.

```
Optical Subsystem STARTUP Error
```

**Cause:** OnLine/Optical detects that the optical storage subsystem is running, but cannot communicate with it properly.

**Action:** Check your optical subsystem for errors.

```
Unable to initiate communication with the Optical Subsystem
```

**Cause:** The optical driver supplied by the optical-drive vendor has indicated that the drive is not accessible.

**Action:** Check the driver installation and the cabling between the computer and the drive.

---

# Installation and Initial Configuration

Prerequisites . . . . .	2-4
Installing OnLine/Optical . . . . .	2-5
Creating the Staging Area . . . . .	2-8
Naming the Staging Area on the STAGEBLOB Parameter . . . . .	2-8
Initializing INFORMIX-OnLine/Optical . . . . .	2-8
Creating the Staging-Area Blobspace . . . . .	2-9
Verifying the Presence of the Optical Subsystem . . . . .	2-10
Creating Optical Families . . . . .	2-10
Testing the Connection . . . . .	2-10
Test 1 . . . . .	2-10
Test 2 . . . . .	2-11



**T**his chapter contains instructions for installing and configuring INFORMIX-OnLine/Optical. It contains the following sections:

- [“Prerequisites”](#) lists the procedures that your optical subsystem vendor must complete before you can install OnLine/Optical.
- [“Installing OnLine/Optical”](#) provides detailed instructions on installing OnLine/Optical.
- [“Creating the Staging Area”](#) tells you how to set the STAGEBLOB parameter in the OnLine/Optical configuration file and how to create the staging-area blob space.
- [“Creating Optical Families”](#) directs you to your optical subsystem vendor documentation for instructions on creating optical families and assigning volumes to them.
- [“Testing the Connection”](#) directs you to perform two tests that indicate whether the connection between OnLine/Optical and the optical subsystem is operating correctly.

---

## Prerequisites

You must obtain an optical storage subsystem from an authorized OnLine/Optical vendor before you can use OnLine/Optical. Contact your Informix sales representative to obtain a list of authorized optical subsystem vendors.

Your optical subsystem vendor provides you with the optical subsystem support library that is linked with OnLine/Optical during the installation process. Your subsystem vendor also provides you with subsystem utilities that perform the following functions:

- Monitor the operation of the subsystem
- Introduce and initialize new optical platters into the optical subsystem
- Perform administrative and management tasks related to the optical subsystem

Your optical subsystem vendor should also perform the following tasks before you install OnLine/Optical:

- Install the subsystem hardware
- Configure the host operating system to support the optical storage subsystem
- Run diagnostics to ensure proper operation of the optical subsystem on the host computer

## Installing OnLine/Optical

This section contains instructions for loading and installing OnLine/Optical. Before starting the procedures, read “Preparing to Install Informix Products” in the *INFORMIX-Universal Server Installation Guide for UNIX*, Version 9.1x. These sections provide important information on preinstallation considerations and setting UNIX and Informix environment variables, respectively.

Your Informix product materials include a serial number keycard and electronic media that contain all product files. Both are necessary for installation. The media can be in any standard form ( $\frac{1}{2}$ -inch 9-track reel tape,  $\frac{1}{4}$ -inch streaming cartridge tape,  $5\frac{1}{4}$ -inch disk,  $3\frac{1}{2}$ -inch disk, and so forth), but it should match the media device of your computer (tape drive, floppy disk drive, and so forth). If you do not have the serial number keycard or the proper media, contact your Informix supplier.

Your serial number keycard provides the correct command for copying the Informix files onto the hard disk of your computer. In most cases, it lists a version of the **tar** or **cpio** command similar to one of the following forms:

```
tar xv[fb] devicename [20]
```

or

```
cpio -icvdBum <devicename
```

The *devicename* refers to the full pathname to that device. Devices are commonly in **/dev**, so the name is usually **/dev/*devicename***.

### To upgrade to Version 9.1x media

1. Log in as **root**.
2. If you are upgrading from a version of INFORMIX-OnLine Dynamic Server, create a complete backup or level-0 archive of your current OnLine system before you load the Version 9.1x OnLine/Optical software. If you use ON-Bar for your backup and restore tool, refer to the *INFORMIX-Universal Server Backup and Restore Guide* for information on performing a complete backup. If you use ON-Archive for your backup and restore tool, refer to the *INFORMIX-Universal Server Archive and Backup Guide* for more information on performing a level-0 archive.



3. Install INFORMIX-Universal Server by following the instructions in the [INFORMIX-Universal Server Installation Guide for UNIX](#).
4. From the `$INFORMIXDIR` directory, enter the following commands:

```
mkdir optical
cd optical
```

Transfer the OnLine/Optical software from the media to the current directory by entering the appropriate **tar**, **cpio**, or other loading command listed on the serial number keycard.

***Tip:** If floppy disks are supplied, you might need to repeat the system command for each disk, or you might be prompted to insert each new disk and press RETURN.*

5. Unload and install the optical subsystem support software from your third-party subsystem vendor according to the vendor's documentation. Before proceeding with the following installation procedures, make sure that the **libop.a** library, supplied by the third-party vendor, is in the `$INFORMIXDIR/optical/oplib` directory. The **libop.a** library must be in the `$INFORMIXDIR/optical/oplib` directory to be linked with OnLine/Optical.
6. Enter the following command to link OnLine/Optical:

```
./opticallink
```

This script relinks the executables for INFORMIX-OnLine/Optical with the object files supplied by the optical subsystem vendor and copies them to the appropriate location in the `$INFORMIXDIR` directory. The INFORMIX-Universal Server tape must already be loaded in the `$INFORMIXDIR` directory, and the **installius** script must be run after optical relinking completes successfully.

At this point the installation script begins. The script first tests to see that you have **root** permissions. It then displays the following message:

```
Press RETURN to continue,
or the interrupt key (usually CTRL-C or DEL) to abort.
```

Press RETURN to continue the installation.



7. If you did not log in as **root**, the script terminates with the following message:  

```
Please rerun this relink procedure as super-user.
```

If you see this message, log in as **root** and restart at step 6.
8. The **opticalink** script displays the following message:  

```
'Linking executables for INFORMIX-OnLine/Optical.'
```

The script relinks **oninit** and uses the object files in the **oplib**, **rsam**, and **turbo** directories and then version stamps the executable files. You see messages that are related to the linking process.
9. The **opticalink** script displays the following message:  

```
Copying linked executables to INFORMIXDIR:
($INFORMIXDIR)
Press RETURN to continue,
or the interrupt key (usually CTRL-C or DEL) to abort.
```

The value of the **\$INFORMIXDIR** variable replaces (**\$INFORMIXDIR**) in the message that appears.

Press RETURN to continue the installation.
10. The **opticalink** script then copies the executable files to **\$INFORMIXDIR/bin**. You see the following messages when the copy operation is complete:  

```
Copying Done!!
Run installius to install INFORMIX-OnLine/Optical

Deleting directories with object files
Press RETURN to continue,
or the interrupt key (usually CTRL-C or DEL) to abort.
```

Press RETURN to continue the installation.

The script removes all the directories and files that were unloaded.
11. Rerun the **installius** script as described in the [\*INFORMIX-Universal Server Installation Guide for UNIX\*](#), Version 9.1x. It is not necessary to reload files from the media before rerunning the script.
12. If you are installing OnLine/Optical Version 9.1x for the first time, configure and initialize OnLine/Optical following instructions in the [\*INFORMIX-Universal Server Administrator's Guide\*](#).
13. Continue with “[Creating the Staging Area](#)” before you access the optical subsystem.

---

## Creating the Staging Area

When simple large objects move from the OnLine/Optical environment to the optical storage subsystem, they are first directed to a memory cache or a blob space on a magnetic disk, which is a *staging area*. Before using the optical storage subsystem, the OnLine/Optical administrator must perform the following tasks:

- Name the staging-area blob space (STAGEBLOB) through ON-Monitor or in the ONCONFIG file (\$INFORMIXDIR/etc/\$ONCONFIG)
- Initialize OnLine/Optical shared memory
- Create the staging-area blob space

## Naming the Staging Area on the STAGEBLOB Parameter

You must provide the name of the staging-area blob space on the STAGEBLOB parameter. The format of the STAGEBLOB parameter is as follows:

```
STAGEBLOB blob space_name
```

If you use ON-Monitor to bring up OnLine/Optical, you are prompted to enter a name for the staging-area blob space. If you choose, you can use a UNIX system editor to edit the STAGEBLOB parameter in your configuration file. For more information on setting ONCONFIG parameters, refer to the [INFORMIX-Universal Server Administrator's Guide](#).

## Initializing INFORMIX-OnLine/Optical

Initialize OnLine/Optical shared memory. For specific information, see the [INFORMIX-Universal Server Administrator's Guide](#). The following error message appears on the screen:

```
Invalid or missing name for Subsystem Staging Blob space
```

OnLine/Optical still comes up even though the staging-area blobSpace has not yet been created. The staging-area blobSpace name supplied as the STAGEBLOB parameter informs OnLine/Optical that an optical storage subsystem is present. You must create the staging-area blobSpace, however, before you can outmigrate a simple large object to the optical storage subsystem.

## Creating the Staging-Area BlobSpace

You create the staging-area blobSpace by using either ON-Monitor or the OnLine/Optical utility **onspaces**. The staging-area blobSpace is the same as any other blobSpace. For instructions on creating a blobSpace, see the [\*INFORMIX-Universal Server Administrator's Guide\*](#). The name of the staging-area blobSpace must be the same as the name you provided on the STAGEBLOB parameter.

After you create the staging-area blobSpace, you must reinitialize OnLine/Optical.

The following example shows the commands you use to initialize OnLine/Optical and create the staging-area blobSpace:

```
# Name the staging-area blobSpace either through ON-Monitor
# or by editing the $ONCONFIG file
# Log in as informix
# NOTE: Use oninit -iy ONLY on new systems. The -i option of
# oninit reinitializes OnLine/Optical and deletes all
# existing databases.
oninit -iy
# Wait for Universal Server to have a status of "online"
# before using onspaces. You can use onstat to check the
# status.
# Add staging-area blobSpace
onspaces -c -b stageblob -g 4 -p /stageblob -o 0 -s 10000
# Start new log
onmode -l
# Backup logs
ontape -a
# Do a level 0 archive
ontape -s -L 0
# Bring down oninit
onmode -yuk
# Bring up oninit
oninit
```

## **Verifying the Presence of the Optical Subsystem**

After configuration, check the OnLine/Optical message log file to make sure that your database server recognizes the optical subsystem. You should see the following message:

```
Optical subsystem is running.
```

If this message is not present, check to make sure that the setup of the optical subsystem is correct and that the OnLine/Optical configuration is correct.

---

## **Creating Optical Families**

A simple large object that is outmigrated to the optical storage subsystem is stored on a volume in an optical family. A simple large object is assigned to an optical family through the CREATE TABLE statement. Before you can assign a simple large object to an optical family, however, you must use the utilities of the subsystem vendor to create an optical family and assign volumes to it. For the procedure to create an optical family, see your subsystem vendor documentation.

---

## **Testing the Connection**

The following tests ensure that you followed the relink and install procedures correctly and that the connection between OnLine/Optical and the optical subsystem is operational.

### **Test 1**

Run the optical utilities of the subsystem vendor that monitor the operation of the optical subsystem and provide the options for platter management.

The optical subsystem log file, located in \$INFORMIXDIR or a user-specified path, should indicate no abnormalities.

## Test 2

Start **oninit** while the subsystem is up. Both the OnLine/Optical log file and the optical subsystem log file should indicate normal operations. The OnLine/Optical log file is located in **\$INFORMIXDIR** or the directory specified by **MSGPATH** in the **ONCONFIG** file, **\$INFORMIXDIR/etc/\$ONCONFIG**.

The message `Optical Subsystem is running` appears in the OnLine/Optical log file if the subsystem is operating normally. For the name of the subsystem log file and the messages that should appear there, see the subsystem vendor documentation.



# Using OnLine/Optical

Assigning Simple-Large-Object Columns to an Optical Platter . . . .	3-4
Reading Simple-Large-Object Columns from an Optical Platter . . .	3-5
Clustering Simple Large Objects . . . . .	3-5
Choosing the Cluster Key . . . . .	3-6
Choosing the Cluster Size . . . . .	3-7
Altering the Cluster Size . . . . .	3-8
Dropping an Optical Cluster . . . . .	3-8
Managing Volumes on the Optical Drives. . . . .	3-9
Reserving an Optical Volume . . . . .	3-9
Releasing a Reserved Optical Volume. . . . .	3-10
Setting the Mounting Time-Out . . . . .	3-11
Using Simple-Large-Object Descriptors . . . . .	3-12
Locating Columns Stored in Optical Families . . . . .	3-14
Locating the Optical Volume Where a Simple Large Object Is Stored . . . . .	3-15
Migrating a Database with Simple Large Objects on an Optical Platter . . . . .	3-16
The onunload and onload Utilities . . . . .	3-16
The dbexport and dbimport Utilities . . . . .	3-17





# T

his chapter illustrates how the SQL extensions provided by INFORMIX-OnLine/Optical support access to WORM optical media. It provides examples that show you how to perform the following functions:

- Assign a simple-large-object column to an optical platter (CREATE TABLE).
- Create an optical cluster (CREATE OPTICAL CLUSTER).
- Reserve an optical volume (RESERVE).
- Release an optical volume (RELEASE).
- Set the mounting time-out (SET MOUNTING TIMEOUT).
- Locate the optical volume where a TEXT or BYTE column is stored (FAMILY() and VOLUME() functions).
- Use simple-large-object descriptors to enable multiple tables to share the same simple large object (DESCR() function).
- Migrate simple large objects in an optical family from one OnLine/Optical system to another (**onunload** and **onload**, **dbexport** and **dbimport**).

The examples use the **cat\_descr** and **cat\_picture** simple-large-object columns from the **catalog** table of the **stores7** database. You can use the optical SQL statements from DB-Access or from a program developed with one of the Informix SQL APIs, such as INFORMIX-ESQL/C.

---

## Assigning Simple-Large-Object Columns to an Optical Platter

To tell OnLine/Optical that a TEXT or BYTE column is to be stored on an optical platter, specify an optical *family name*, rather than a dbspace or blobspace name, in the column-definition portion of the CREATE TABLE statement. The optical storage subsystem administrator must create the optical family name in the subsystem before you can execute a CREATE TABLE statement that refers to it. For the procedure to create the optical family name, see the subsystem vendor documentation.

In the following example, the TEXT column **cat\_descr** of the **catalog** table is stored with the table in a dbspace and the BYTE column **cat\_picture** is stored in the optical family **catalog\_stores7**:

```
CREATE TABLE catalog
(
  catalog_num      SERIAL (10001),
  stock_num        SMALLINT NOT NULL,
  manu_code        CHAR(3) NOT NULL,
  cat_descr        TEXT IN TABLE,
  cat_picture      BYTE IN catalog_stores7,
  cat_advert       VARCHAR (255, 65)
)
IN dbspace10
```

When the statement is executed, OnLine/Optical checks to determine whether **catalog\_stores7** is a dbspace, a blobspace, or an optical family name. Either of the following conditions returns an error:

- The name refers to a blobspace or dbspace *and* an optical family name.
- The name does not refer to a blobspace, dbspace, *or* optical family name.

Once the table is created, you can insert TEXT and BYTE data into an optical family in the following ways:

- With the LOAD statement (DB-Access) or the **dbload** utility
- From locator variables (INFORMIX-ESQL/C)

---

## **Reading Simple-Large-Object Columns from an Optical Platter**

Use the SELECT statement to read a simple-large-object column from an optical platter in the same way that you read a simple-large-object column from a magnetic disk. The syntax of the SELECT statement does not change to support access to an optical storage subsystem. However, OnLine/Optical supports extensions to SQL that were developed exclusively for use with optical storage. These additional SQL statements accomplish the following tasks that affect how efficiently a simple-large-object column is read from the optical platter:

- Clustering simple large objects
- Managing optical volumes

### **Clustering Simple Large Objects**

You can use the CREATE OPTICAL CLUSTER statement to store logically related simple large objects on the same volume of an optical platter. Storing related simple large objects together minimizes platter exchanges when the simple large objects are retrieved. The CREATE OPTICAL CLUSTER statement requires you to perform the following steps:

- Assign a name to the optical cluster.
- Specify the simple-large-object columns in a table that are to be stored together in the optical cluster.
- Specify the cluster key. The cluster key consists of one or more columns in the table that define the logical grouping of the simple-large-object columns. The columns that make up the cluster key must not be simple-large-object columns.
- Specify the size of the cluster in kilobytes.

The following CREATE OPTICAL CLUSTER statement allocates a cluster of 18,000 kilobytes on an optical platter for the **cat\_picture** BYTE column in the **catalog** table. It assigns the name **catalog\_catpics** to the cluster.

```
CREATE OPTICAL CLUSTER catalog_catpics
  FOR catalog (cat_picture)
  ON (manu_code)
  CLUSTER SIZE 18000
```

The cluster key for **catalog\_catpics** is **manu\_code**. This means that **cat\_picture** simple large objects that have the same value in the **manu\_code** column are stored together on the same volume or volumes, if more than one is required.

### *Choosing the Cluster Key*

In the preceding example, the **manu\_code** column was chosen for the cluster key because, of all the columns in the **catalog** table, it best meets the following criteria for choosing a cluster key:

- It provides a logical order for retrieving the simple large objects.
- It contains duplicate values.

The first criterion for the cluster key, which can consist of more than one column, is that it provides a logical order for retrieving the simple large objects. The purpose of the cluster key is to optimize retrieval of the simple large objects by grouping related simple large objects on the same volume. Grouping simple large objects by a cluster key is only advantageous, however, if the simple large objects are normally retrieved in the order that they are grouped. If the **cat\_picture** simple large objects are grouped by **stock\_num** and then retrieved by **manu\_code**, as shown in the following example, the subsystem might still need to perform multiple platter exchanges to retrieve the simple large objects:

```
SELECT cat_picture FROM catalog WHERE manu_code = 'HRO'
```

The second criterion for a cluster key is that it must contain duplicate values. Of all the columns in the **catalog** table, only the first three, **catalog\_num**, **stock\_num**, and **manu\_code**, provide logical sequences for accessing the table. Of these columns, **catalog\_num** is not useful because it has a SERIAL data type, which produces a unique value for each row in the table. Clustering on this column produces a cluster for every row in the table where **cat\_picture** is not null.

Assume, for example, that 26 rows in the **catalog** table have pictures in the **cat\_picture** column, and that the **manu\_code** column has seven unique values with occurrences as shown in the following example:

### Output

manu_code	occurrences
ANZ	7
HRO	6
KAR	1
NRG	1
PRC	4
SHM	5
SMT	2

For these characteristics, **manu\_code** produces seven unique clusters with the same number of simple large objects in each cluster as there are occurrences of each key. By contrast, assume that in the same 26 rows, there are 21 unique **stock\_num** values, only three of which occur multiple times (104 x 2, 110 x 4, 205 x 3). In this case, choosing **stock\_num** as the cluster key produces 21 clusters, 17 of which contain only one simple large object.

### Choosing the Cluster Size

The cluster size is the size of the cluster expressed in kilobytes. It is based on the number of simple large objects that you expect to store in each cluster and the average size of the simple large object. It can be calculated as follows:

$$\text{cluster size} = \text{estimated number of simple large objects} * \text{average simple-large-object size}$$

If the average size of the **cat\_picture** images is 60 kilobytes, and you wish to store up to 300 simple large objects per cluster, the cluster size is 18,000 kilobytes.

### ***Altering the Cluster Size***

You can use the ALTER OPTICAL CLUSTER statement to change the size of an optical cluster for all clusters that are created after the statement is executed. In the following example, the size of the **catalog\_catpics** cluster is reduced to 6,000 kilobytes, a cluster size that stores up to one hundred 60-kilobyte large objects per cluster:

```
ALTER OPTICAL CLUSTER catalog_catpics CLUSTERSIZE 6000
```

The ALTER OPTICAL CLUSTER statement changes only the cluster size for large objects created after the statement is executed. Clusters that were created previously remain their original size.

### ***Dropping an Optical Cluster***

You can use the DROP OPTICAL CLUSTER statement to terminate optical clustering. The following example terminates clustering for **catalog\_catpics**:

```
DROP OPTICAL CLUSTER catalog_catpics
```

The DROP OPTICAL CLUSTER statement does not affect the simple large objects that were already stored in a cluster. It terminates clustering only for the affected columns in the future.

You can also use the DROP OPTICAL CLUSTER statement with the CREATE OPTICAL CLUSTER statement to change either the simple-large-object columns or the cluster-key columns for an optical cluster. For example, you can change the cluster key for **catalog\_catpics** from **manu\_code** to **stock\_num** with the following statements:

```
DROP OPTICAL CLUSTER catalog_catpics;  
CREATE OPTICAL CLUSTER catalog_catpics  
  FOR catalog (cat_picture)  
  ON (stock_num)  
  CLUSTERSIZE 3000
```

These statements do not affect simple large objects that were previously clustered on **manu\_code**, because you cannot alter data written to a WORM optical platter. These statements change the clustering only for future inserts or updates. If you wish to recluster the simple large objects that were clustered by **manu\_code**, you must update them; the simple large objects are then rewritten to the new cluster. The simple-large-object descriptors are updated with the new location of the simple large objects. The space that these simple large objects originally occupied is lost because you cannot reuse space on WORM optical media.

## Managing Volumes on the Optical Drives

Your application can perform the following operations to manage the volumes on the optical drives and, consequently, affect the performance of the application:

- Reserve an optical volume to keep it mounted during a set of operations
- Release a reserve request to free an optical drive
- Set the time-out period for mounting an optical volume

### *Reserving an Optical Volume*

You can use the RESERVE statement to keep a required optical volume mounted on an optical drive while you periodically access it during a set of operations. The RESERVE statement implies a request to mount the specified volume if it is not currently mounted.

When multiple users access the optical platters simultaneously, you may need to reserve a volume if your application accesses it consecutively. Otherwise, individual accesses are interleaved with requests from other applications so that each access by your application could require the volume to be remounted on the optical drive. This results in an unsatisfactory response time from the optical storage subsystem.

For each RESERVE request, the optical storage subsystem adds the value one (1) to a reserve counter for that volume. If the value of the reserve counter is greater than zero, you cannot remove the volume. You can issue multiple RESERVE requests for a volume, but you must release each one before you can remove the volume from the drive. When you are finished accessing the volume, use the RELEASE statement to release it. For details, “Releasing a Reserved Optical Volume” in the following section.

If all the optical drives are reserved when a RESERVE statement is issued, the subsystem waits for the number of seconds specified by the SET MOUNTING TIMEOUT statement or the default time-out period.

You can use a SELECT statement to obtain the family name and volume number of the volume that you want to reserve. The FAMILY() and VOLUME() function expressions return the family name and volume number, respectively. In the following example, the SELECT statement uses the FAMILY() and VOLUME() functions to identify the volume that contains pictures of stock for manufacturer HRO. The RESERVE statement then reserves the volume, using the family name and volume number that the SELECT statement returned.

```
SELECT FAMILY(cat_picture), VOLUME(cat_picture)
      FROM catalog
      WHERE manu_code = 'HRO'
:
RESERVE 'catalog_1991' 013
```

### ***Releasing a Reserved Optical Volume***

You must release each RESERVE request for an optical volume with the RELEASE statement. The RELEASE statement cancels the latest RESERVE request for an optical volume by subtracting the value one (1) from the reserve counter for that volume. When the reserve counter is reduced to zero, you can remove the optical volume from the drive. The following statement releases volume 013 of the optical family:

```
RELEASE 'catalog_1991' 013
```

An error is returned if you issue a RELEASE statement for a volume that is not reserved.



The optical storage subsystem vendor supplies a utility that you can also use to release a volume. The subsystem administrator must release a volume with this utility when a program terminates prior to releasing a reserved volume.

### ***Setting the Mounting Time-Out***

The SET MOUNTING TIMEOUT statement lets you control the amount of time that your application waits for a volume to be mounted before it abandons the request to continue processing. When you request access to an optical volume, the subsystem cannot mount it immediately if all the optical drives are busy processing requests from other users. In addition, one or more drives might be reserved at the time of your request. If a SET MOUNTING TIMEOUT statement is not executed, the subsystem times out after five minutes (300 seconds) if it cannot mount the requested volume.

The SET MOUNTING TIMEOUT statement provides the following options for setting the time-out period:

- Wait indefinitely until a drive is available for the requested volume
- Abort the request for a volume immediately if it is not currently mounted on an optical drive
- Specify a number of seconds to wait for a volume to be mounted
- Wait for a volume to be mounted only if a drive is available

If it is mandatory that a particular optical volume be mounted before an application continues, it might be appropriate for the application to wait an indefinite amount of time for that volume to mount. If so, the following statement has that effect:

```
SET MOUNTING TIMEOUT TO WAIT
```

If multiple users and multiple applications are competing for the optical drives, there might be times when you do not want your application to wait for a volume that is not currently mounted. If this is the case, you might set the mounting time-out to NOT WAIT. For example, if your application retrieves data about an item of merchandise and the simple-large-object data on an optical platter is desirable but not required, the application might not display the simple-large-object data if it is not immediately available.

The following example causes a request for a volume to abort immediately if the volume is not currently mounted:

```
SET MOUNTING TIMEOUT TO NOT WAIT
```

Normally, you should set the time-out period to the number of seconds that is appropriate for the application to wait for a volume. The following statement sets the mounting time-out period to 30 seconds. If the volume is not mounted within 30 seconds, the process times out and control is returned to the application.

```
SET MOUNTING TIMEOUT TO 30
```

Set the mounting time-out period to zero seconds to cause the process to wait for a volume to be mounted only if a drive is currently available, as follows:

```
SET MOUNTING TIMEOUT TO 0
```

When a time-out period of zero is specified, the subsystem waits for a volume to mount if a drive is currently available for it. When the NOT WAIT option is specified, the subsystem does not wait for the requested volume to mount if it is not currently on a drive.

---

## Using Simple-Large-Object Descriptors

Data rows that include simple-large-object data do not include the simple-large-object data in the row itself. Instead, the data row contains a 56-byte simple-large-object descriptor that includes a forward pointer (rowid) to the location where the first segment of data is stored. The descriptor can point to a dbspace blobpage, a blobspace blobpage, or an optical storage location.

You can use the DESCR() function expression to retrieve the simple-large-object descriptor for a large object that is stored on WORM optical media. The DESCR() function returns the descriptor in an encoded format that is 112 bytes long. Using the DESCR() function as a column value in INSERT and UPDATE statements enables you to create additional pointers to existing simple large objects on the optical media. This saves space on the optical platters by allowing multiple tables to refer to the same physical simple large object.

Before the simple-large-object descriptor is inserted into a data row, it is decoded and validated to verify consistency and legitimacy. The optical subsystem performs all encoding, decoding, and validations through OnLine/Optical calls to API functions. If the validation process fails, the insert operation fails.

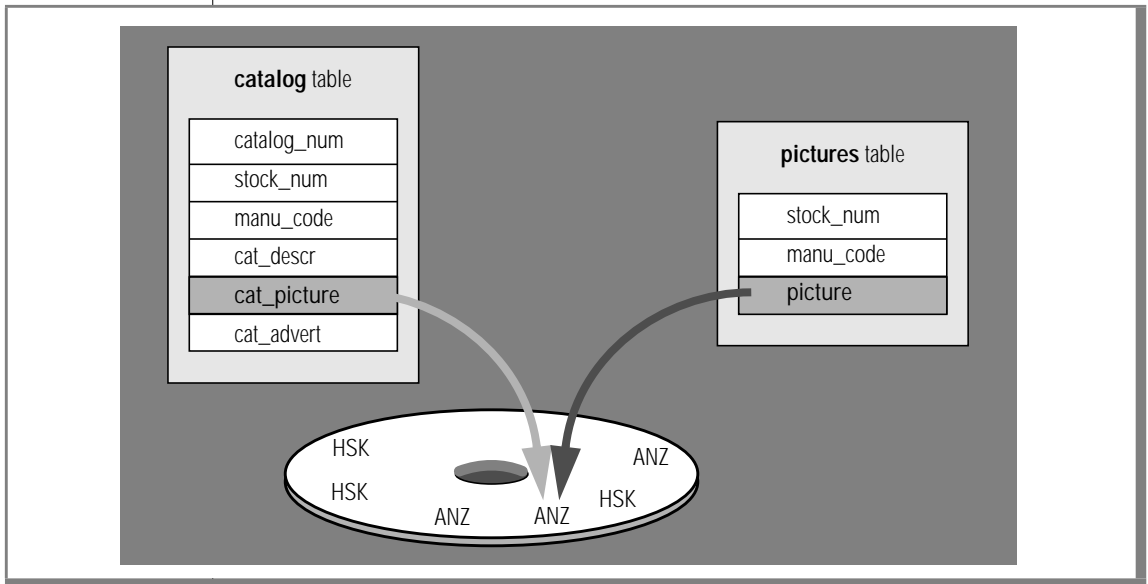
The following example uses DESCR() in an INSERT statement to insert simple-large-object descriptors, or pointers to existing simple large objects, into the following hypothetical **pictures** table. Assume that the **picture** column is a simple-large-object column assigned to an optical platter.

```
INSERT INTO pictures (stock_num, manu_code, picture)
  SELECT stock_num, manu_code, DESCR(cat_picture)
    FROM catalog
   WHERE manu_code = 'ANZ'
```

Figure 3-1 illustrates the result of the preceding INSERT statement for a row in the **pictures** table.

**Figure 3-1**

*Two Descriptors That Point to the Same Large Object on a WORM Platter*



The following example uses DESCR() to update the **pictures** table with all pictures from the **cat\_picture** column of the **catalog** table:

```
UPDATE pictures
  SET (picture) =
      (SELECT DESCR(cat_picture)
       FROM catalog
       WHERE pictures.stock_num = catalog.stock_num
       AND pictures.manu_code = catalog.manu_code
       AND cat_picture IS NOT NULL)
```

You can use the DESCR() function expression for simple-large-object columns that are stored on an optical platter. An error is returned if you use DESCR() on a nonoptical large-object column.

---

## Locating Columns Stored in Optical Families

You can determine which columns in a database are stored on an optical platter by querying the **sysblobs** table. The **sysblobs** table contains four columns that provide the following information:

- The space name (blob space, db space, or family name) where the column is stored
- The media type (M = magnetic; O = optical)
- The table identification number
- The column number within the table

The following SELECT statement shows the locations of all simple-large-object columns in a given database:

```
SELECT * FROM sysblobs
```

The output that the preceding SELECT statement produces for the **stores7** database might indicate that column number 5 in table 109 is stored in the optical family **family1**.

### Output

spacename	type	tabid	colno
rootdbs	M	109	4
family1	0	109	5

With this information, you can query the **syscolumns** table to obtain the name of the column, as follows:

```
SELECT colname FROM syscolumns WHERE tabid = 109 AND colno = 5
```

That particular query produces the following output:

### Output

colname
cat_picture

The **oncheck** utility with the **-pD** option also tells you whether a simple-large-object column is stored on an optical platter. However, the information is displayed on a row-by-row basis. The following **oncheck** command displays information for rows in the **catalog** table of the **stores7** database:

```
oncheck -pD stores7:catalog
```

For more information about **oncheck**, refer to the [INFORMIX-Universal Server Administrator's Guide](#).

---

## Locating the Optical Volume Where a Simple Large Object Is Stored

You can use the **FAMILY()** and **VOLUME()** function expressions in a **SELECT** statement to obtain the family name and volume number where a simple large object is stored. The **SELECT** statement in the following example produces a list of the optical volumes that holds pictures of bicycle helmets (**stock\_num** = 110):

```
SELECT FAMILY(cat_picture), VOLUME(cat_picture)
FROM catalog
WHERE stock_num = 110
```

---

## Migrating a Database with Simple Large Objects on an Optical Platter

When a database contains simple large objects that are stored in an optical family, you can migrate the database from one OnLine/Optical system to another by using the following utilities:

- **onload** and **onunload**
- **dbexport** and **dbimport**

This section documents the operation of these programs with respect to migrating a database that contains simple large objects stored on an optical platter. For a complete description of the **onload** and **onunload** utilities, refer to the [INFORMIX-Universal Server Administrator's Guide](#). For a description of **dbexport** and **dbimport**, see the [Informix Guide to SQL: Reference](#).

If you are migrating a database that contains simple large objects from one OnLine/Optical subsystem to another, check your subsystem vendor documentation for the compatibility requirements among subsystems. For example, the destination subsystem might need to include the same simple-large-object family name as the one that the source subsystem uses.

### The onunload and onload Utilities

The **onunload** utility unloads data from a database or table to tape. The **onload** utility reads a tape created by **onunload** and creates the database or table on disk. The **onunload** and **onload** utilities represent the most efficient way to unload and load a database because they read and write data in binary disk-page units. For a table that stores simple large objects in an optical family, **onunload** and **onload**, respectively, unload and load the simple-large-object descriptors and the large-object data. When using **onload**, the destination computer must have the same page size as the computer from which the database was unloaded.

**Warning:** When you are using WORM optical media, loading simple-large-object data back to the optical storage subsystem where it originated causes the data to be duplicated in the optical family.



## The dbexport and dbimport Utilities

The **dbexport** utility unloads a database into ASCII files; the **dbimport** utility creates a database from ASCII files. You can use the **-d** option of the **dbexport** utility to specify that only the simple-large-object descriptor is written for a simple-large-object column that is stored on an optical platter. Migrating the descriptor permits a simple large object that is stored on an optical platter to be shared in multiple databases, eliminating the need to store duplicate copies of it on an optical platter. The simple-large-object descriptor contains information about the location and size of the data. It holds all the information that is necessary to retrieve a simple large object from an optical platter. If **-d** is not specified, **dbexport** exports both the descriptor and the simple large object.

If you want a different cluster arrangement in the destination database and you can afford the space on an optical platter, you can choose to duplicate the simple large object in a different cluster.

The following **dbexport** command exports the **stores7** database to the **./exstores7** directory, exporting only descriptors for tables that contain simple-large-object columns. (The **-c** instructs the program to ignore all errors except fatal errors, **-o** specifies the output directory, and **-q** suppresses the echo of SQL statements.)

```
dbexport -c -d -o ./exstores7 -q stores7
```

Simple-large-object descriptors remain valid from one OnLine/Optical system to another because each volume on an optical platter contains an internal tracking label. In addition to simple-large-object data, each WORM volume contains a label with the following information:

- Family name
- Family number (also referred to as the subsystem serial number)
- Volume number
- Optical subsystem identifier

Each optical subsystem device, whether a stand-alone drive or jukebox, has a unique optical subsystem identifier. The subsystem uses the identifier and the family name to uniquely identify the family within the subsystem. This becomes an important issue in a distributed database environment where more than one optical storage subsystem might be available.





---

# SQL for OnLine/Optical

ALTER OPTICAL CLUSTER . . . . .	4-4
CREATE OPTICAL CLUSTER . . . . .	4-6
DROP OPTICAL CLUSTER . . . . .	4-10
RELEASE. . . . .	4-12
RESERVE. . . . .	4-14
SET MOUNTING TIMEOUT . . . . .	4-16
Function Expressions. . . . .	4-17
DESCR(). . . . .	4-19
FAMILY(). . . . .	4-19
VOLUME(). . . . .	4-20



# T

his chapter describes the syntax and usage of the individual SQL statements that INFORMIX-OnLine/Optical provides to support optical storage subsystems. The syntax of each statement is illustrated in a diagram that shows the sequence of the required and optional elements. For a complete description of the conventions used in these diagrams, see the “Introduction.” The SQL statements presented in this chapter are used exclusively with optical storage subsystems. For additional information on the context in which each statement is used, see [Chapter 3, “Using OnLine/Optical.”](#)

This chapter describes the following SQL statements:

- ALTER OPTICAL CLUSTER
- CREATE OPTICAL CLUSTER
- DROP OPTICAL CLUSTER
- RELEASE
- RESERVE
- SET MOUNTING TIMEOUT

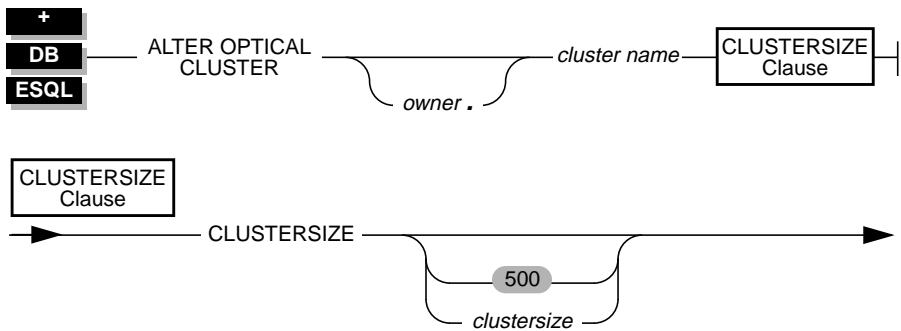
This chapter also describes the following function expressions:

- DESCR()
- FAMILY()
- VOLUME()

# ALTER OPTICAL CLUSTER

Use the ALTER OPTICAL CLUSTER statement to alter the size of an optical cluster.

## Syntax



Element	Purpose	Restrictions	Syntax
<i>cluster name</i>	The name of the cluster to alter	The cluster name must already exist	Identifier segment, see <a href="#">Informix Guide to SQL: Syntax</a>
<i>clustersize</i>	Size of the cluster, in kilobytes, to be reserved for each unique cluster-key value	Must be less than the volume size	Literal Number segment, see <a href="#">Informix Guide to SQL: Syntax</a>
<i>owner</i>	The user name of the owner of the cluster	The specified name must be a valid user name	Identifier segment, see <a href="#">Informix Guide to SQL: Syntax</a>

## Usage

To alter an optical cluster, you must be the owner of the cluster, have the Index privilege on the table, or have the DBA privilege.

In the following example, the CREATE OPTICAL CLUSTER statement creates **cat\_clstr**, an optical cluster of 6,000 kilobytes for the **cat\_picture** column in the **catalog** table. For each unique value of **stock\_num** in the table, **cat\_clstr** stores the associated data for this column. The following example changes the amount of space allocated for each **cat\_clstr** from 6,000 kilobytes to 8,000 kilobytes. This change affects all clusters created after the ALTER OPTICAL CLUSTER statement is executed. Clusters created prior to an ALTER OPTICAL CLUSTER statement maintain their original size.

```
CREATE OPTICAL CLUSTER cat_clstr
  FOR catalog (cat_picture)
  ON (stock_num)
  CLUSTER SIZE 6000
ALTER OPTICAL CLUSTER cat_clstr CLUSTER SIZE 8000
```

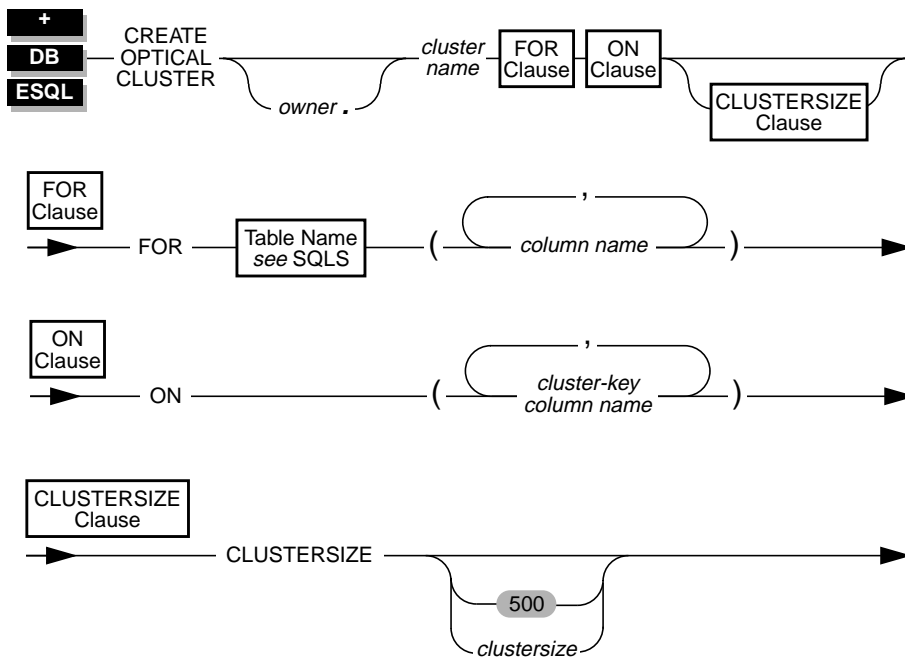
## References

See the CREATE OPTICAL CLUSTER statement on [page 4-6](#) and DROP OPTICAL CLUSTER statement on [page 4-10](#).

## CREATE OPTICAL CLUSTER

Use the CREATE OPTICAL CLUSTER statement to create a cluster on optical platter for one or more logically related simple-large-object columns.

### Syntax



Element	Purpose	Restrictions	Syntax
<i>column name</i>	The name of the simple-large-object column to be clustered	You can assign up to 16 distinct columns to a cluster.	Identifier segment, see <a href="#">Informix Guide to SQL: Syntax</a>
<i>cluster-key column name</i>	The name of the column that is the cluster key	You can specify up to 16 columns to create a composite cluster key. You cannot specify a simple-large-object column as part of a cluster key.	Identifier segment, see <a href="#">Informix Guide to SQL: Syntax</a>
<i>cluster name</i>	The name of the optical cluster	The name must be unique.	Identifier segment, see <a href="#">Informix Guide to SQL: Syntax</a>
<i>clustersize</i>	The amount of optical platter space to allocate for the cluster, specified in kilobytes. If no clustersize is specified, the size of the cluster defaults to 500 kilobytes.	The clustersize must be less than the volume size.	Expression segment, see <a href="#">Informix Guide to SQL: Syntax</a>
<i>owner</i>	The valid user name of the cluster	The specified name must be a valid user name.	Identifier segment, see <a href="#">Informix Guide to SQL: Syntax</a>

## Usage

An optical cluster enables logically related simple large objects to be stored on the same volume, which minimizes time-consuming platter exchanges during retrieval.

You can create (and, by default, become owner of) an optical cluster if you are the owner of the table, have the Resource privilege on the database and the Index privilege on the table, or have the DBA privilege on the database.

Only a person with DBA privileges can create the optical cluster and specify another user as the owner of the cluster.

### ***Simple-Large-Object Columns***

The following restrictions are placed on the simple-large-object columns that are being clustered on the optical platter:

- All columns within a single clustering strategy must reside on the same optical family.
- A single column cannot be clustered more than once.

### ***Cluster-Key Columns***

A cluster key can be a single column (for example, **stock\_num**) or a composite of up to 16 columns (for example, **stock\_num**, **manu\_code**). You cannot specify a simple-large-object column as a cluster key. The maximum length of a cluster key is 256 bytes. An error message is returned if you submit a cluster key that is longer than 256 bytes.

You can gain better performance if you organize the columns in a composite cluster key so that the column with the fewest number of duplicate values is placed first.

### ***Clustersize***

The *clustersize* is the size, in kilobytes, of the space reserved on an optical volume for each cluster with a unique cluster-key value. The size of the cluster can be enlarged beyond this size, if necessary, but it cannot exceed the size of a volume. If you do not specify *clustersize*, the default size of the cluster is 500 kilobytes.

You can create more than one optical cluster for the same list of cluster-key columns. This option differs from the restrictions placed on indexes, each of which you must create from a unique set of index-key columns.



### Example

Assume that the CREATE TABLE statement for the **catalog** table in the **stores7** database specifies that the columns **cat\_descr** and **cat\_picture** are to be stored in the same optical family. In the following example, the CREATE OPTICAL CLUSTER statement creates **cat\_clstr**, an optical cluster of 6,000 kilobytes. A new **cat\_clstr** is allocated to store the data for **cat\_descr** and **cat\_picture** each time a row is inserted with a unique value for **manu\_code**.

```
CREATE OPTICAL CLUSTER cat_clstr
  FOR catalog (cat_picture, cat_descr)
  ON (manu_code)
  CLUSTER SIZE 6000
```

If the **stores7** database contains six unique values for **manu\_code**, the statement lays the groundwork for the optical storage subsystem to create six optical clusters, one for each value. If the table contains ten rows for one **manu\_code**, then the **cat\_clstr** for that value contains the **cat\_descr** and **cat\_picture** data for all ten rows.

To create the cluster, the subsystem reserves 6,000 kilobytes on the current volume. If the current volume is too full to accept the space reservation, the cluster is created on the next available volume in the family. The location and size of each cluster on the **manu\_code** cluster key is tracked in an internal optical-cluster table. The name of the optical cluster, the number of its simple-large-object columns, and the cluster-key column numbers are stored in the **sysopclstr** table.

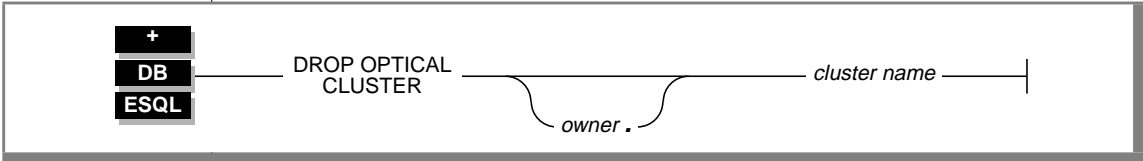
### References

See the ALTER OPTICAL CLUSTER statement on [page 4-4](#) and the DROP OPTICAL CLUSTER statement on [page 4-10](#). In the *Informix Guide to SQL: Syntax*, see the CREATE TABLE statement.

# DROP OPTICAL CLUSTER

Use the DROP OPTICAL CLUSTER statement to terminate optical clustering for the simple-large-object columns that are associated with the cluster name.

## Syntax



Element	Purpose	Restrictions	Syntax
<i>cluster name</i>	The name of the optical cluster	The cluster name must already exist.	Identifier segment, see <a href="#">Informix Guide to SQL: Syntax</a>
<i>owner</i>	The user name of the owner of the cluster	The specified name must be a valid user name.	Identifier segment, see <a href="#">Informix Guide to SQL: Syntax</a>

## Usage

The DROP OPTICAL CLUSTER statement terminates clustering on the optical platter for the simple-large-object columns associated with *cluster name*, and drops the internal optical cluster table. After the DROP OPTICAL CLUSTER statement is executed, the data items that are associated with the cluster name are stored on the current volume; they are stored in the sequence that they are output to the optical storage subsystem.

You must be the owner of the optical cluster or have the DBA privilege to drop the optical cluster.

You can change the cluster-key columns of an optical cluster by dropping the old optical cluster and creating a new optical cluster with the new cluster-key columns. You can change your optical clustering strategy so that the **cat\_picture** and **cat\_descr** columns of the **catalog** table are clustered according to **stock\_num** instead of **manu\_code**, as follows:

```
DROP OPTICAL CLUSTER cat_clstr
CREATE OPTICAL CLUSTER cat_stock_clstr
    FOR catalog (cat_picture, cat_descr)
    ON (stock_num)
    CLUSTER SIZE 6000
```

After the new optical cluster, **cat\_stock\_clstr**, is created on **stock\_num**, all data that is inserted to the database for the **cat\_picture** and **cat\_descr** columns is optically clustered according to the associated value of **stock\_num**. However, the new cluster does not alter the clustering of simple large objects already stored on the WORM optical platters. A new cluster affects only subsequent outmigrations. Therefore, in this example, the simple large objects that were previously clustered by **cat\_clstr** retain their original locations. In subsequent outmigrations, the data for the **cat\_descr** and **cat\_picture** columns is clustered by **cat\_stock\_clstr**, based on the value of **stock\_num**. Changing the cluster key impacts how the data is stored on the optical platter but not on the integrity of the data.

Changing the clustering arrangement often can have an adverse affect on how efficiently data is retrieved. The simple large objects that remain clustered by **manu\_code** are effectively unclustered when they are accessed by **stock\_num**. Accessing them by **stock\_num** can result in frequent platter exchanges. You can alleviate this problem by updating the simple large objects that were previously stored by **manu\_code** so that they are rewritten in the new **stock\_num** clusters. Because the optical media is WORM, however, the space where the simple large objects were stored originally cannot be reused.

Administrators and programmers should be aware that the optical clustering strategy significantly affects the use of space on the optical platters.

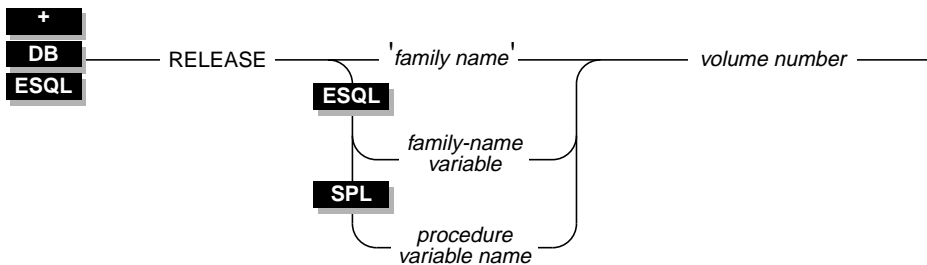
## References

See the CREATE OPTICAL CLUSTER statement on [page 4-6](#).

## RELEASE

Use the RELEASE statement to cancel a reserve request for an optical volume.

### Syntax



Element	Purpose	Restrictions	Syntax
<i>family name</i>	A quoted-string constant that specifies a family name in the optical subsystem	Must be an existing family name	Quoted String segment, see the <a href="#">Informix Guide to SQL: Syntax</a>
<i>family-name variable</i>	A CHARACTER or VARCHAR host variable that contains a family name	Must be an existing variable name	Variable name must conform to language-specific rules for variable names.
<i>procedure variable name</i>	The name of a variable defined in a stored procedure	Must be defined in the procedure and valid in the statement block	Identifier segment, see the <a href="#">Informix Guide to SQL: Syntax</a>
<i>volume number</i>	The volume being released; specified as an integer	Must be reserved and mounted correctly	VOLUME() function, p. 4-20 RESERVE statement, p. 4-14 Expression segment, see the <a href="#">Informix Guide to SQL: Syntax</a>

## Usage

Each time a RELEASE statement is executed, the value one (1) is subtracted from the reserve counter for the volume. When the value of the reserve counter is zero, you can unmount the volume.

The volume specified in the RELEASE statement must be reserved and the platter containing the volume must be mounted currently. An error is returned if the platter is not mounted or if the value of the reserve counter for the volume is zero.

You can issue multiple reserve requests, but you must release each one before you can remove the volume from the optical drive.

The following example releases a reservation for volume 013 in the **catalog\_1991** optical family:

```
RELEASE 'catalog_1991' 013
```

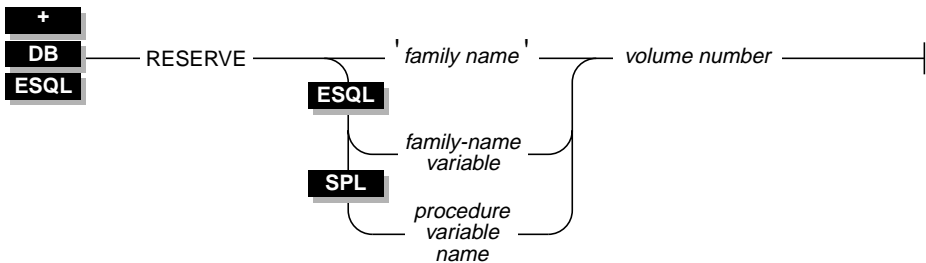
## References

See the RESERVE statement on [page 4-14](#).

# RESERVE

Use the RESERVE statement to keep a particular optical volume mounted on the optical drive for the duration of a set of operations. The RESERVE statement implies a request to mount the specified volume if it is not currently mounted.

## Syntax



Element	Purpose	Restrictions	Syntax
<i>family name</i>	A quoted-string constant that specifies a family name in the optical subsystem	Must be a unique name	Quoted String segment, see the <a href="#">Informix Guide to SQL: Syntax</a>
<i>family-name variable</i>	A CHARACTER or VARCHAR host variable that contains a family name	Must be a unique variable name	Variable name must conform to language-specific rules for variable names.
<i>procedure variable name</i>	The name of a variable defined in a stored procedure	Must be defined in the procedure and valid in the statement block	Identifier segment, see the <a href="#">Informix Guide to SQL: Syntax</a>
<i>volume number</i>	The number of the volume being reserved; specified as an integer	None	VOLUME() function, p. 4-20 Expression segment, see the <a href="#">Informix Guide to SQL: Syntax</a>

## Usage

If the platter that contains the specified volume is not currently mounted on a drive, the subsystem waits for a drive to become available and mounts the platter before it continues. The default waiting period for a mount attempt is five minutes (300 seconds). You can change this default period with the SET MOUNTING TIMEOUT statement.

You can issue multiple reserve requests for the same volume. Each time a RESERVE statement is executed for a volume, the value one (1) is added to the reserve counter for that volume. If no reserve requests have been issued for an optical volume, the value of the reserve counter is zero, indicating that you can remove the volume from the optical drive.

When you finish accessing the reserved volume, you can release it with the RELEASE statement. Each time a RELEASE statement is executed, the value one (1) is subtracted from the reserve counter for that volume. Thus, you can issue multiple reserve requests, but you must release each one before you can remove the volume from the optical drive.

You can use a SELECT statement with the FAMILY() and VOLUME() functions to obtain the family name and volume number of the volume that you wish to reserve. Then, you can specify that optical volume in the RESERVE statement, as follows:

```
SELECT FAMILY(cat_picture), VOLUME(cat_picture)
      WHERE manu_code = HRO AND stock_num = 301
:
:
RESERVE 'catalog_1991' 013
```

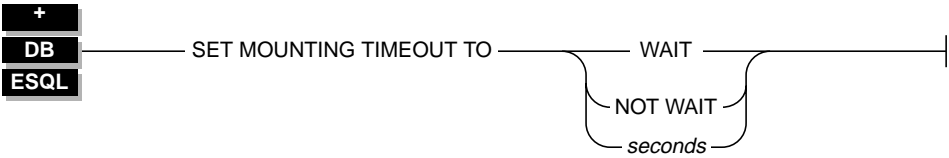
## References

See the RELEASE statement on [page 4-12](#).

# SET MOUNTING TIMEOUT

Use the SET MOUNTING TIMEOUT statement to set the time-out period for mounting a volume on the optical drive. If the SET MOUNTING TIMEOUT statement is not executed, the default time-out for mounting a volume is five minutes (300 seconds).

## Syntax



Element	Purpose	Restrictions	Syntax
<i>seconds</i>	The maximum number of seconds the process should wait for access to a volume. There is no default value for time-out within the statement.	Must be specified as a positive integer or zero.	Literal Number segment, see <a href="#">Informix Guide to SQL: Syntax</a>

## Usage

If you set the time-out value to WAIT, all requests for a volume wait until the volume is mounted on an optical drive. The process that made the request waits indefinitely.

If you set the time-out value to NOT WAIT, all requests for a volume are aborted if that volume is not currently mounted on an optical drive.

If you set the time-out value to a positive integer, the integer specifies the number of seconds that the process waits for a volume to be mounted before the request is aborted.



If you set the time-out value to zero, processes are directed to wait for a volume to be mounted only if an optical drive is currently available. If a drive contains either a reserved volume or a platter that is being accessed, the request is aborted. If the drive is available, the process waits for the platter containing the requested volume to be mounted. In general, mounting an optical platter from its slot to a drive in the jukebox takes about 15 seconds.

The following example sets the mounting time-out period to 45 seconds. When the application subsequently requests that a volume be mounted, it waits for up to 45 seconds for the platter with that volume to be mounted. If the platter with that volume cannot be mounted within the 45-second period, the process times out.

```
SET MOUNTING TIMEOUT TO 45
```

---

## Function Expressions

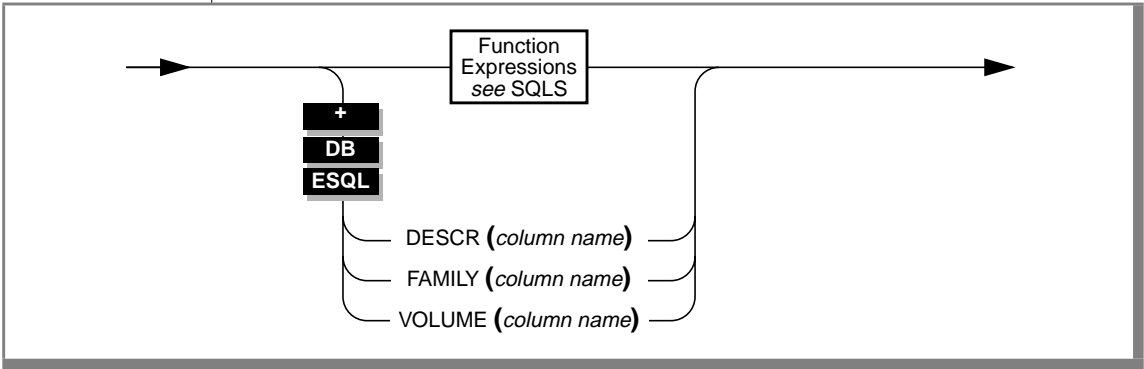
The following function expressions support the optical storage subsystem. Use them with the SELECT statement.

- DESCR()
- FAMILY()
- VOLUME()

You can use a function expression to perform various operations on column data or to obtain information from the database server about the contents of one or more columns. A function expression requires an argument or parameter. For all three of these function expressions, the argument is the name of a column that is stored on an optical platter. For more information on function expressions and a complete list of other function expressions, see the [Informix Guide to SQL: Syntax](#).

Use the DESCR(), FAMILY(), and VOLUME() function expressions to obtain information about simple-large-object columns that are stored on optical platters.

Syntax



Element	Purpose	Restrictions	Syntax
<i>column name</i>	The name of the simple-large-object column on which the function operates	The column name must already exist	Identifier segment, see <a href="#">Informix Guide to SQL: Syntax</a>

Usage

The DESCR() function returns a TEXT large object that is the encoded descriptor for the simple-large-object column. The FAMILY() function returns the name of the optical family in which a column is stored. The VOLUME() function returns the number of the volume on which a column is stored. All three functions are valid *only* for data stored on an optical platter. They are valid whether the simple large objects are stored sequentially or in clusters. An error message is returned if the functions are used on a column that is not stored on an optical platter.

## **DESCR()**

The DESCR() function returns a TEXT large object, which is the encoded descriptor for the simple-large-object column. The encoded descriptor is 112 bytes. It is used as a column value in INSERT and UPDATE statements as a way to create additional pointers to existing simple large objects on the optical platter. Before the descriptor is inserted into a data row, it is decoded and validated to verify its consistency and legitimacy. The optical subsystem performs all encoding, decoding, and validations through calls by OnLine/Optical to API functions. If the validation fails, the insert operation fails.

The DESCR() function is designed for WORM optical media only.

In the following example, the DESCR() function is used in an INSERT statement to fill the **picture** column of the **pictures** table with pointers to existing simple large objects from the **catalog** table:

```
INSERT INTO pictures (stock_num, picture)
  SELECT stock_num, DESCR(cat_picture) FROM catalog
  WHERE manu_code = 'HRO'
```

## **FAMILY()**

The FAMILY() function returns the name of the optical family in which a simple-large-object column is stored.

In the following example, the FAMILY() function obtains the *family name* of the optical platters that contain the data for the **cat\_picture** column:

```
SELECT FAMILY(cat_picture) FROM catalog WHERE manu_code = 'HRO'
```

One row of output (*family name*) is generated for each row where **manu\_code** is equal to HRO.

## **VOLUME()**

The **VOLUME()** function returns the number of the volume where a simple-large-object column is stored.

Volumes are filled in numeric order. Once a volume is filled, or no room is available for a cluster, all subsequent writes occur on volumes with a higher number. When simple large objects are stored sequentially, the highest volume number contains the most recently stored objects. However, this generalization is not true for clustered large objects.

In the following example, the **VOLUME()** function generates a list of the volumes that contain data for the **cat\_picture** column:

```
SELECT VOLUME(cat_picture) FROM catalog
```

## **References**

For more information on using **FAMILY()**, **VOLUME()**, and **DESCR()**, see [Chapter 3](#).

For the following information, see the [Informix Guide to SQL: Syntax](#):

- A complete list of other function expressions
- The syntax of the column-definition portion of the **CREATE TABLE** statement, which is used to assign a simple-large-object column to an optical family
- The **SELECT** statement

# Index

## A

ALTER OPTICAL CLUSTER  
     statement 3-8, 4-4  
 ANSI compliance  
     -ansi flag Intro-10  
     level Intro-16  
 Application programming interface (API)  
     external layer 1-11, 1-19  
     internal layer 1-11, 1-17  
 Archive, restrictions on 1-7  
 Assigning a column to optical disk 1-7  
 Assigning a simple-large-object column to optical disk 1-7, 3-4  
 Autochanger 1-5

## B

BLOB data type 1-3  
 Blobspace, creating the staging-area 2-9  
 BYTE column  
     in catalog table 1-9, 3-4  
     storing on optical disk 3-4

## C

CLOB data type 1-3  
 Cluster key  
     choosing the 3-6  
     composed of 3-5  
 CREATE OPTICAL CLUSTER  
     statement 1-9  
     defining 1-9

Cluster size  
     altering 3-8  
     calculating 3-7  
     choosing the 3-7  
     specifying 3-5  
 Clustered storage 1-7  
 Clustering simple large objects  
     CREATE OPTICAL CLUSTER  
         statement 3-5  
         discussion of 3-5  
 Clustering simple-large-object columns  
     how it occurs 1-10  
 Column-definition portion of  
     CREATE TABLE statement 1-7, 3-4  
 Column, simple-large-object  
     assigning to optical family 1-7  
     changing 3-8  
     clustering 3-5  
     CREATE OPTICAL CLUSTER  
         statement 1-9  
     defining 1-9  
 Comment icons Intro-7  
 Compliance  
     with industry standards Intro-17  
 Compliance, with industry standards Intro-16  
 Configuration parameter,  
     OPCACHMAX 1-13  
 CREATE OPTICAL CLUSTER  
     statement  
         cluster key 4-8  
         clustering simple large objects 3-5  
         clustering simple-large-object  
             columns 1-9  
         clustersize 4-8

example 3-6, 4-9  
logically ordering simple-large-object columns 1-9  
requirements 3-5  
restrictions on simple-large-object columns 4-8  
syntax and use 4-6  
**CREATE TABLE** statement  
  assigning a column to optical platter 1-7  
  assigning a simple-large-object column to optical disk 1-7, 3-4  
  creating optical family name 3-4  
  defining simple-large-object column for catalog table 1-9  
  example 3-4  
  **IN** clause 3-4

---

## D

Data types  
  BLOB 1-3  
  CLOB 1-3  
dbaccessdemo7 script Intro-5  
dbexport 3-17  
dbimport 3-17  
Default locale Intro-4  
Demonstration database Intro-5  
Descriptor, simple large object  
  DESCR() function expression 3-12  
  using 3-12  
DESCR() function expression  
  in an INSERT statement 3-13  
  syntax and use 4-18  
Documentation conventions  
  icon Intro-7  
  sample-code Intro-14  
  syntax Intro-8  
  typographical Intro-6  
Documentation notes Intro-16  
Documentation, types of  
  documentation notes Intro-16  
  error message files Intro-15  
  machine notes Intro-16  
  on-line manuals Intro-15  
  printed manuals Intro-15  
  release notes Intro-16

**DROP OPTICAL CLUSTER**  
  statement 4-10  
  described 3-8

---

## E

Environment variable,  
  **INFORMIXOPCACHE** 1-13  
en\_us.8859-1 locale Intro-4  
Error message files Intro-15  
Extension, to SQL  
  symbol for Intro-10  
External layer (API), libop.a 1-19

---

## F

**FAMILY()** function expression 4-19  
Family, description of 1-6  
Features, product Intro-5  
Function expressions  
  DESCR() 4-19  
  **FAMILY()** 4-19  
  **VOLUME()** 4-20

---

## G

Global Language Support  
  (GLS) Intro-4

---

## I

Icons  
  comment Intro-7  
  syntax diagram Intro-9  
Industry standards, compliance  
  with Intro-16  
**INFORMIXDIR**/bin  
  directory Intro-5  
**INFORMIX-ESQL/C** and  
  **INFORMIX-OnLine/Optical** 3-3  
**INFORMIX-OnLine/Optical**  
  description of Intro-3, 1-3  
  installing 2-5  
  prerequisites for installation 2-4

**INFORMIXOPCACHE**  
  environment variable 1-13  
Immigration  
  activity 1-18  
  definition of 1-17  
Inserting simple-large-object  
  descriptors 3-13  
Installation  
  creating optical families 2-10  
  creating the staging-area  
    blob space 2-8  
  **INFORMIX-Universal Server** 2-6  
  installing the optical subsystem  
    support software 2-6  
  installius script 2-7  
  materials included 2-5  
  testing the connection 2-10  
  verifying that database server  
    recognizes optical  
    subsystem 2-10  
Installing **STAGEBLOB**  
  parameter 2-8  
installius script 2-7  
ISO 8859-1 code set Intro-4

---

## J

Jukebox 1-5

---

## L

libop.a, description of 1-19  
Locale Intro-4

---

## M

Machine notes Intro-16  
Major features Intro-5  
Memory cache  
  how space is allocated 1-14  
  how to assess size 1-14  
  monitoring available and  
    allocated resources 1-16  
  purpose 1-13  
  relationship to staging-area  
    blob space 1-13

setting for client with  
INFORMIXOPCACHE 1-13  
setting for system with  
OPCACHEMAX 1-13  
Migrating a database, with simple  
large objects on optical  
platter 3-16

---

## O

On-line manuals Intro-15  
onload 3-16  
ON-Monitor 1-14  
  naming the staging-area  
  blob space 1-16  
Onstat utility, using to monitor  
  resources 1-16  
onunload 3-16  
OPCACHEMAX configuration  
  parameter 1-13  
Optical drive, performance 3-9  
Optical family 3-4  
Optical media  
  accessing 1-5  
  advantages 1-4  
  platter 1-5  
  storing simple large objects 1-4  
Optical platter  
  component of subsystem 1-5  
  exchanges 3-5  
Optical storage subsystem  
  assigning a simple-large-object  
  column to an optical family 1-7  
  components 1-5  
  interaction between INFORMIX-  
  OnLine/Optical and API 1-11  
  interaction between the API and  
  the subsystem 1-11  
  relationship with INFORMIX-  
  OnLine/Optical and the  
  API 1-12  
  stand-alone 1-5  
  subsystem management  
  software 1-5  
  vendor library 1-11  
Outmigration  
  activity 1-17  
  definition of 1-17

---

## P

Performance, application 3-9  
Printed manuals Intro-15

---

## R

Release notes Intro-16  
RELEASE statement  
  example 3-10  
  reserve counter 3-10  
  syntax and use 4-12  
RESERVE statement  
  FAMILY() function  
  expression 3-10  
  reserve counter 3-10  
  syntax and use 4-14  
VOLUME() function  
  expression 3-10

---

## S

Sample-code conventions Intro-14  
SELECT statement  
  obtaining family name 3-10  
  obtaining volume number 3-10  
  reading a simple-large-object  
  column 3-5  
  with FAMILY() function  
  expression 3-15  
  with VOLUME() function  
  expression 3-15  
Sequential storage, simple large  
  object 1-7  
SET MOUNTING TIMEOUT  
  statement  
  number of seconds option 3-12  
  syntax and use 4-16  
  using 3-11  
Shelf 1-5  
Simple large object  
  clustered storage 1-7  
  description of 1-4  
  sequential storage 1-7  
  storage location 1-18

---

Software dependencies Intro-4  
SQL extensions, purpose 3-5  
SQL statements, using with  
  INFORMIX/OnLine Optical 3-3  
STAGEBLOB parameter  
  blob space\_name 1-16  
  naming the staging-area  
  blob space 2-8  
Staging-area blob space  
  creating 2-9  
  creating with ON-Monitor or  
  onspaces 1-14  
  how to assess size 1-15  
  using onstat utility to monitor  
  resources 1-16  
stores7 database Intro-5  
Subsystem management  
  software 1-5  
Syntax conventions  
  elements of Intro-9  
  example diagram Intro-12  
  how to read Intro-12  
  icons used in Intro-9  
sysblobs system catalog table 3-14  
sysopclstr table 4-9  
System catalog table, sysblobs 3-14

---

## T

TEXT column  
  in catalog table 3-4  
  storing on optical disk 3-4

---

## U

Utility program  
  dbexport 3-17  
  dbimport 3-17  
  oncheck 3-15  
  onload 3-16  
  ON-Monitor 1-14  
  onspaces 1-14  
  onunload 3-16

---

## **V**

### **Volume**

- managing on the optical drive 3-9
- SET MOUNTING TIMEOUT
  - statement 3-11

### **VOLUME() function expression**

- identifying an optical volume 3-10
- syntax and use 4-20

### **Volume, description of 1-6**

---

## **W**

### **Write-Once-Read-Many (WORM) 1-4**

---

## **X**

### **X/Open compliance level Intro-16**