STANDARD
FOR THE EXCHANGE OF DIGITAL INFORMATION ON CD-ROM

CD-ROM Read-Only Data Exchange Standard
(CD-RDx)

Commissioned by

The Information Handling Committee
Director of Central Intelligence
Intelligence Community Staff
Washington, DC 20090-0828

Version 3.00
December 31, 1990

STANDARD FOR THE EXCHANGE OF DIGITAL INFORMATION ON CD-ROM

Table of Contents

STANDARD FOR THE EXCHANGE OF DIGITAL INFORMATION ON CD-ROM

1. PURPOSES

The purposes of this standard are to:

o Establish **CD-ROM Read-Only Data Exchange** (CD-RDx) protocols that will enable universal interoperability of CD-ROMs and foster interchange of information.

o Provide criteria for the acceptance and rejection of application software, CD-ROM retrieval engines, user interfaces and CD-ROM databases according to this standard.

2. OBJECTIVES

It is in the best interests of the U.S. Government to foster an environment in which access to data on CD-ROM discs would be **system independent**, i.e., functionally interoperable across systems, and **software independent**, i.e., functionally interoperable with any search and retrieval program.

The **objectives** of this standard are (1) to define the criteria to enable interoperability of CD-ROMs within any and among all operating systems, CD-ROM drives, access/retrieval programs, and user interfaces; (2) to promote long-term storage and exchange of information on CD-ROM between, among and within the agencies of the U.S. Government; and (3) to provide a standard that would promote adoption of CD-ROM as an accepted archiving medium.

This standard provides a sufficient level of detail to guide implementation and minimize ambiguity in the production of open systems, interoperable CD-ROMs. This standard is designed to be fully compliant, at a minimum, with *ISO/IEC Standard 10149* which defines the logical blocks of data on a compact disc, the *International Standard for the CD-ROM Volume and File Format* (**ISO 9660**) which defines the logical volume and file format for CD-ROM. Requirements for adherence to this standard should be included in all requests for proposals for the production of digital data on CD-ROM, CD-

ROM retrieval engines, user interfaces (including those accompanying software application programs) and commercially available CD-ROM applications. Questions concerning this standard should be directed to Chairman, DCI Intelligence Information Handling Committee, Intelligence Community Staff, P.O. BOX 90828, Washington, D.C., 20090-0828 who will issue any interpretations or succeeding amendments or modifications, thereto, as may be required.

3. SCOPE

This standard is applicable to all CD-ROM retrieval engines, all indexes to data residing on CD-ROM discs or read-only updatable media, and all computers supporting CD-ROM drives and all commercially available CD-ROM applications.

4. OVERVIEW

CD-ROM discs are simple, inexpensive media containing large amounts of permanently stored digital data, an index to that data and software to search and retrieve the data. CD-ROM discs are used in CD-ROM drives, which serve only as read-only peripheral devices to computers. Because CD-ROMs are read-only media, once produced, they are **incapable** of having additional data written to them, unlike other removable media such as floppy diskettes or Write Once Read Many (WORM) disks. Physically, CD-ROM discs are small (12 cm or 4.72" diameter) and durable; yet they store the equivalent amount of textual data contained in a typical 4-5 drawer filing cabinet.

More importantly, CD-ROMs contain permanently stored, machine-readable (binary) data. This data can be quickly accessed, retrieved, revised and incorporated in reports, yet remain intact and available for subsequent access. Since CD-ROM data is in machine-readable form, easy transfer to any subsequent archiving media is assured. For these reasons, CD-ROM is considered an appropriate medium for the dissemination and archiving of digital data, replacing and/or supplementing paper, microfiche, 9-track tapes and other digital mass storage devices. It is, therefore, imperative to develop, implement and adopt a CD-ROM standard that enables interoperability ("plug and play") between and among all operating systems, all retrieval programs and all user interfaces.

No such standard for CD-ROM interoperability exists. Computer-controlled CD-ROM drives and CD-ROM search and retrieval software are subject to unique commands of each computer's operating system, which is most likely proprietary. Although ISO 9660 defines the volume and file format for the placement of data on CD-ROMs, no standard exists for accessing, searching and retrieving the data. These functions have been left to software developers to execute and to CD-ROM producers to implement. The result is a ever-expanding array of different installation methods per application, different user interfaces, different query functions and different system requirements. The burden of all these differences falls primarily on the user, the individual who must install each disc per unique instruction and learn how to use each

new user interface for each new CD-ROM.

The CD-RDx standard was developed to overcome the limitations of multiple, unique operating systems and the multiple, unique user interfaces accompanying the growing number of CD-ROM software programs applications.

4.1 Sequence of Events of a CD-RDx Query

With the CD-RDx standard, a CD-ROM would contain the user data, one or more data indexes, a search engine and a Server. The search engine and the Server may be separate or combined, may reside on the CD-ROM only or be copied onto the computer's hard disk drive. Data access and retrieval occurs in the following manner (see Figure #1):

(1) USER:

- Enters a query according to the user interface program of the user's choice.

(2) USER INTERFACE:

- Passes query to Server, using standard commands and protocols defined in the CD-RDx standard.

(3) Server:

- Passes query to search engine program.

(4) SEARCH ENGINE:

- Receives query from Server, using standard commands and protocols defined in the CD-RDx standard,
- Executes query against data index,
- Determines data that meets query logic,
- Identifies data locations on CD-ROM,
- Retrieves data from CD-ROM,
- Passes data to Server.

(5) Server:

- Passes data to Client program, using standard commands and protocols defined in the CD-RDx standard.

(6) Client:

- Presents data to user.

(7) USER:

- Views data per user interface features.

Although a user interface developed specifically for the database/retrieval engine may also accompany the CD-ROM application, the CD-RDx standard stipulates that any user interface conforming to the CD-RDx standard may access data on a CD-ROM conforming to the CD-RDx standard, as described above. In this way, the user may select the user interface of his or her choice and access any CD-ROM application conforming to the CD-RDx standard. The CD-ROM producer that publishes CD-RDx compatible applications is assured that the CD-ROM is operable within all operating systems and that the database is accessible to all users.

4.2 Precedence for CD-RDx

The **CD-ROM Read-Only Data Exchange** standard is required to assure producers and users of CD-ROM databases that the discs they publish and access are interoperable, regardless of the computer operating system, the retrieval engine or the user interface. Five problems or preferences served as the basis for defining CD-RDx:

(1) Historically, all data, indexes to the data and software to access the data on CD-ROM have been bundled. This has fostered the development of more than one hundred software programs and at least 1,000 commercially available CD-ROM applications. Each software program and/or database on CD-ROM is unique, and therefore the installation programs and user interfaces are also unique. This situation oftentimes forces the CD-ROM user to acquire unique hardware per application, to modify the computer's environment upon installation of each application, and to learn a new user interface per application.

(2) The CD-ROM user should interact only with the user interface software when seeking read-only data. All functions beyond the user interface, are, therefore transparent to the user, who may not, or need not be concerned with **how** the data is accessed, searched or retrieved.

(3) The growing sophistication of relational database management systems provides for a Client/Server architecture, which separates the functions of the user interface from the functions of data access and retrieval. The CD-RDx standard follows this same architecture and stipulates the commands and protocols for interaction between the Client and the Server.

(4) The Client or user interface may be developed separately from any given retrieval engine. Each Client may initiate basic data access and retrieval commands or may initiate more sophisticated commands required for displaying graphics and video or for combining the display of text simultaneously with audio or video segments. The Client may be a separate product or may come with a CD-ROM software application program. The Client may be upgraded and new features added without affecting access to data on a CD-RDx disc.

(5) The Server is developed directly for a specific retrieval engine by the software developer. In fact, for increased speed, the Server and the retrieval engine may

become one. The Server may be upgraded and new features added without affecting the access and display of data on a CD-RDx user interface.

The CD-RDx standard defines a set of protocols enabling the transfer of CD-ROM data between any CD-RDx Client and any CD-RDx Server. The standard consists of a set of commands, 2-dimensional tables and a simple procedure for their use. CD-RDx replaces the need for users to learn a new interface for each CD-ROM application. CD-RDx requires that software developers construct Servers for their proprietary retrieval engines. CD-RDx allows CD-ROM software developers, publishers or third-party developers to create new user interfaces or modify existing ones.

The CD-RDx standard assumes the following for each and every CD-ROM produced and used:

- Digital data resides on CD-ROMs in proprietary format.

- Each CD-ROM Server is available in memory.

- Selected user interfaces can be enhanced with a library of routines to access and manipulate certain kinds of digital data residing on a CD-ROM.

- Every user interface that implements CD-RDx can access any CD-ROM database that implements CD-RDx via a Server.

- CD-RDx Servers accept messages and respond appropriately without additional user or user interface involvement.

- A database structure (index) can be improved and additions made without affecting the user interface programs. Conversely, a user interface program can be improved and additions made without affecting the database structure.

- The CD-RDx Protocol/CD-ROM database combination is a **black box** approach used extensively throughout the computer industry and is known and accepted by software developers.

- Communication between a Server and user interface occurs using English-like messages in ASCII text format.

- The data accessed and retrieved from a CD-ROM complying with the CD-RDx standard may be alphanumeric, audio, still or animated graphics, raster or vector images, video, tagged or non-tagged data, compressed or decompressed images. The ability to access and display each or all of these formats is dependent on the capabilities of the Client. The ability to retrieve each or all of these formats is dependent on the capabilities of the Server.

5. FORMAT FOR CD-RDx PROTOCOLS

5.1 Server Structures

As with most database management systems, the basic unit of reference is the **record**. Records are composed of one or more **fields** of related information. Fields may contain fielded data, structured text, full text data, graphics, audio, video, tagged or non-tagged data, compressed or decompressed data. Some retrieval engines require that all fields appear in every record, that the fields appear in a particular order, or that fields occur only once. Some retrieval programs do not have such limitations.

Fielded data is normally organized with a uniform record model. In this type of environment, data is organized such that every record contains the same fields, in the same order.

Structured text (such as laws, regulations, technical documentation and specifications) often implies a method that represents the content as a series of fields. The developer typically starts with the lowest level of the text, and works upward. When completed, the records contain a collection of fields that describe the structure (e.g., Book, Chapter and Section), and a single field that contains the text of that element. The level of detail of the structure is determined by the developer or the requirements for the developer.

Full text data, unlike structured text, rarely implies a fielding strategy through structure alone. With full text data, the developer must rely on a logical content model, rather than on a predetermined structure.

All written communication has structure in the form of words, sentences, and paragraphs. Based on experimentation with the database, the developer can determine a logical principle that will aid in the rapid recall of information. For example, almost all documents have a meaningful paragraph structure. When this is the case, the developer might determine that every paragraph should be in a **separate record**. In another instance, the developer may decide that the entire document should be indexed as a **single record**. These decisions may then be implemented as Document Type Definitions (DTDs) and the data tagged according to SGML (Standardized General Markup Language) or a proprietary tagging scheme.

Typically, a CD-ROM database is composed of a series of records. Records contain a series of fields. A field within a record can be as large as the available CD-ROM disc space or can be limited in size. In order to process a record of indefinite size, each record is divided into a series of **sub-records**. Each field, present in the record, has one or more sub-records. Each sub-record contains a certain number of characters.

5.2 Objects and Tables

As a low-level tool, the protocols can be interpreted under both a procedural and an object metaphor. The CD-RDx Protocols are a system of objects (or tables) that behave in a similar way.

In the context of a procedural programming methodology, messages to the Server can be thought of as **commands** to a highly integrated parser. The object model allows the programmer to treat all requests sent to the parser in a similar way. This allows the developer to create a small family of functions that format command strings for execution and return results to the calling module. For example, many applications require the ability to browse lists of available indexes or fields while selecting specific members of the list. Since all of these objects respond to the same messages, the same module can be used to format command strings, and interpret results. Other aspects of the system behave in similar ways.

For OOP users, the CD-RDx Protocols provide the elements of a Server Class. The developer can produce a class using these elements that models the Server Object required for the specific application. The Server can be tailored to the Class concept that suits the developer's ideology.

Whatever the model, all the objects in the CD-RDx Protocols can be thought of as two-dimensional tables. A **table** is a list of rows and columns. Each **row** is a different member consisting of a set of columns called attributes. For example, the INDEXLIST object is a table in which each row contains an index description, and the columns give the attributes of that index. The same rationale is applied to all of the other object types described in this standard.

Each object contains zero or more elements (according to the specific instance of the object). Each object has different data available. For example, a data record object (RECORD) is a series of sub-records. Each element has a field name, and length associated with it as well as the actual data.

Each object instance must be opened (see OPEN message) before using that instance. Access is terminated with the CLOSE message. Some objects are associated with other objects. When these objects are opened, access to associated objects is implied. In particular, a database object has six associated objects, a DATASETLIST, a FIELDLIST, an INDEXLIST, an OPENRECORDLIST, an OPENKEYLIST, and an OPENRECORDSETLIST.

5.3 Currency

To ease the management of many instances of similar objects, the CD-RDx Protocols includes the concept of **currency**. The messages sent to the Server are received by the current instance of the named object. This allows the developer to have many instances of an object active without complicating the massaging system.

The Database keeps track of currency for each object. There are two types of currency. The first type is **member currency**. In each open object instance, the current member is kept for use with the GET message and the PREV, NEXT and CURR modifiers. For example, the message:

GET FIELDLIST CURR > NAME(30)

will ask the FIELDLIST object the name of the current member of the field list. The GET and SEEK messages automatically update member currency.

The second type of currency is **object currency**. When there is more than one instance of an object type, the Server keeps track of which object is current. The action of many of the messages is based on the current instance of an object. The user can change object currency by using the CURRENT message. Object currency is automatically updated with the OPEN and CREATE messages.

Whenever an object is opened or created, it becomes the current object of that type. When an object with associated objects is made current (or opened or created), its associated objects are also made current.

5.4 Object Summary

The following objects exist in the CD-RDx Protocols:

DATABASELIST	A table of all open databases. Each element is a description of an open database.
FIELDLIST	A table of all available fields for the current database.
INDEXLIST	A table of all available indexes for the current database.
DATASETLIST	A table of all available datasets for the current database.
DATASETFIELDLIST	A table of all available fields given the current dataset configuration.
FIELDDATASETLIST	A table of all available datasets given the current field configuration.
RECORD	A list of all sub-records in an open RECORD conforming to the current field configuration.
OPENRECORDLIST	A table of all open record descriptions for the current database.
RECORDSET	A table of all record numbers in an open RECORDSET.
OPENRECORDSETLIST	A table of all record set descriptions for the current database.
KEYLIST	An alphabetical list of all keys from an open index.
OPENKEYLIST	A table of all open KEYLIST descriptions for the current database.

5.5 Message Types

The following messages exist in the CD-RDx Protocols:

OPEN Prepares the named object for access. This message applies only to DATABASE, RECORD, and KEYLIST objects types. The OPEN message also establishes currency within the object.

CLOSE Terminates access to an object and removes the object from memory.

SELECT Adds the named FIELDS or DATASETS to the list of currently selected objects of the same type. The FIELDS selection modifies the list of FIELDS that will be returned on subsequent GET RECORD requests. The DATASET selection modifies the available set of records subsequent FIND messages will operate on.

REMOVE Removes the named FIELDS or DATASETS from the list of currently selected objects of that type. See SELECT.

GET Retrieves data from an object as specified in the info list description of each object.

SEEK Positions the currency pointer within an object according to the associated key.

CREATE Creates a new RECORDSET object, sets currency, and readies for subsequent FIND messages.

DELETE Deletes a RECORDSET object from memory.

CLEAR Resets a RECORDSET object to an empty state.

FIND Locates the specified key and updates the current record set with the new occurrence information.

RECORDSET Performs record set management.

CURRENT Establishes currency for the specified object type. This applies to databases, records, keylists, and record sets.

5.6 Modifiers

The following modifiers exist in the CD-RDx Protocols:

CURR Returns the currently selected instance of the specified object

FIRST Sets the currency pointer for the named object to the first element before executing the requested operation.

LAST Sets the currency pointer for the named object to the last element before executing the requested operation.

NEXT Sets the currency pointer for the named object to the next element before executing the requested operation.

If the end of the list is reached, a warning will be returned and currency will be wrapped around to the beginning of the list. The next call will return the first element in the list.

PREV Sets the currency pointer for the named object to the previous element before executing the requested operation.

If the beginning of the list is reached, a warning will be returned and currency will be wrapped around to the end of the list. The next call will return the last item in the list.

COUNT Returns the number of instances or items in the named object list.

5.7 Message Summary

The following is a summary of the messages that exist in the CD-RDx Protocols:

Message Type	**Object Types**	**Modifiers**
OPEN (O)	DATABASELIST (DBL)	FIRST
CLOSE	INDEXLIST	LAST
SELECT (SEL)	FIELDLIST	NEXT
REMOVE (REM)	DATASETLIST	PREV
GET (G)	OPENRECORDSETLIST	CURR
SEEK (SK)	OPENRECORDLIST	THROUGH
CREATE (CR)	OPENKEYLIST	COUNT
DELETE	RECORD (REC)	OR
FIND (F)	KEYLIST (KL)	AND
RECORDSET (REC)	RECORDSET (RECSET)	NOT
CURRENT (CUR)	DATASET (DS)	XOR
CLEAR (CLR)	FIELD	PROX(##)
	DATABASE (DB)	ADJ
	DATASETFIELDLIST	LIKE
	FIELDDATASETLIST	[name]
		'[id]

The parentheses indicate abbreviations for the message or object type. Most of the messages have the following format:

MESSAGETYPE OBJECTTYPE MODIFIER

The modifier refers to additional information which is MESSAGETYPE specific. Modifiers are parsed according to a few rules. There are several special characters. These are comma, space, ;, *, ?, ', `, \, and >.

The comma and spaces are delimiters. The semicolon is a terminator. The asterisk and question mark are wild cards. The single quote and back slash are used to place multiple words or special characters into text strings.

To put these characters in a text string, they must be in single quotes or preceded by a back slash (\). To use the asterisks (*) or question mark (?) or single quote ('), the back slash is used first.

The (`) character is used as an abbreviation to speed up some operations. For instance, when a field name (or index name or dataset name) is required in the syntax of a message, the field ID (or index id or dataset id) can be used in its place (e.g. SELECT FIELD `1).

The > symbol specifies the beginning of an information list. The individual

messages are described below.

5.8 Information Lists

The GET, SEEK and CURRENT messages can return information to the data area provided by the programmer (the results parameter). The user supplies a list of data names which are appropriate for the object type and the message. The information list follows the special '>' character in the message. The information list has the following format:

DATANAME1(size[,format]),DATANAME2(size[,format]), ... ;

The DATANAMEn is a selection from the list of available information. There are two parameters for each data name. The first is the field size and is required. The second describes the string formatting for the returned data. Valid data formats include the following: T for text (default), C for C type strings (left justified, null terminated), and P for PASCAL strings (left justified, with the first byte being the length byte). For example:

> NAME(30),ID(4);

is an info list specifying that the NAME field will be 30 characters in length. If the actual data is shorter than 30 characters, then it will be padded with spaces. If it is longer than 30 characters, then only the first 30 characters will be transferred. The same is true for the four character ID field.

The second parameter is the data format and is optional. This allows the user some flexibility since the source language for the Client program may require special handling. For example:

> NAME(30,C),ID(4,C);

is an information list which again asks for NAME and ID fields but requests that the data be returned in C string format (NULL terminated strings). It also means that only 29 bytes, and 3 bytes respectively will at most contain data. There will always be a NULL byte at the end of the string.

Another format is PASCAL strings. These strings contain a length byte at the beginning of every string. The same rule applies to PASCAL strings as to C strings. One extra byte will be needed. The length parameter always indicates the exact size of the buffer. The P format is used to request a PASCAL string.

6. MESSAGE DESCRIPTIONS

All messages yield a return code. This is a short (16 bit) integer value. A return code of zero indicates that the function was performed correctly. A positive return code is a warning message. This means that the message worked properly but there is no data returned. This is usually an end of table message. A negative number indicates that some sort of error occurred. See Appendix B for the list of possible error codes.

The GET, SEEK and CURRENT messages also return database information. The information received depends on the object type and the information list requested. The information list is optional for these messages. If there is no information list, then no additional information is returned.

The detailed format for each message of the CD-RDx Protocols follows.

6.1 OPEN

The OPEN message will open and initialize selected objects for use by a Client application. The objects are DATABASE, KEYLIST and RECORD. A name is always associated with each 'OPEN' object so that it may be referenced by the 'CURRENT' message. All open messages will return the name string upon completion without error.

- OPEN DATABASE dbname filepath [encryption key];

Opens the database indicated by the 'filepath' parameter and assigns the internal name 'dbname' to it. If the character '\' is included in the file path it must be duplicated in the string.

Example:

"OPEN DATABASE stamp l:\\stamps\\stampdb"

This message will open the database stampdb and assign the local name 'stamp' to the object.

- OPEN KEYLIST keyname indexname;

This message opens a keylist (index) for processing. This message also establishes currency for the keylist.

Example:

"OPEN KEYLIST k1 'word list'"

- OPEN RECORD recname rec#;

This message opens a record identified by rec# and assigns the name 'recname' to it. Record currency is established.

Example:

"OPEN RECORD rec1 `1"

This message will open the first record and set currency to the first sub-record for subsequent "GET RECORD" messages.

6.2 CLOSE

The CLOSE message will remove the named object from memory. It is only used with those objects that can be 'OPEN'ed.

- CLOSE DATABASE [dbname];

This message will close the named DATABASE. If no name is given, then the current database is closed.

- CLOSE KEYLIST [keyname];

This message will close a previously opened KEYLIST identified by the keyname. If no keyname is given, then the current keylist is closed.

- CLOSE RECORD [recname];

This message will close a previously opened RECORD identified by the 'recname'. If no recname is given, then it will close the current record.

6.3 SELECT

The SELECT message marks the specified items as selected and available for use by the Client application. Initially, all fields and datasets are considered selected. These selected items are used for various functions inside the Server. The selected datasets are used to limit a search to only a selected group of datasets even if search criteria spans datasets. The selected fields are used to return only those selected fields when reading a record. The selected lists are **NOT** to be confused with currency which is an entirely different concept and function.

- SELECT DATASETS ds1,ds2, ..., dsn;
- SELECT DATASETS *;
- SELECT FIELDS fld1,fld2, ..., fldn;
- SELECT FIELDS *;

Modify the selected list to contain the items requested. The asterisks (*) indicates all available items are selected.

6.4 REMOVE

The REMOVE message removes items from a selected list in the same manner that the SELECT message adds them;

- REMOVE DATASETS ds1,ds2, ..., dsn;
- REMOVE DATASETS *;
- REMOVE FIELDS fld1,fld2, ..., fldn;
- REMOVE FIELDS *;

Modify the selected list to remove the items requested. The asterisks (*) indicates all available items are removed.

6.5 GET

The GET message retrieves information from the object requested. Since the current object of the particular type is the one accessed, currency must be established for that type before issuing the GET message. Currency is established by using the CURRENT message. The OPEN or CREATE messages also establish currency when invoked.

The format of the message is as follows:

GET OBJECTTYPE MODIFIER [> info list];

The modifier specifies which action to take on the object type. The data returned is a series of strings which identifies attributes about the selected object member, except the COUNT modifier which returns a number of members (rows) of the object.

- GET DATABASELIST modifier [> info list];

  Returns an entry or count from the list of open databases. The data returned contains the name of the database, the file name, the number of records, and the maximum record size.

- GET INDEXLIST modifier [> info list];
- GET FIELDLIST modifier [> info list];
- GET DATASETLIST modifier [> info list];

  Returns an entry or count from the list of available indexes, fields or datasets. The data returned includes the name, id, and type.

- GET OPENRECORDSETLIST modifier [> info list];

  Returns an entry or count from the list of open record set names.

- GET OPENRECORDLIST modifier [> info list];

  Returns an entry or count from the list of open records.

- GET OPENKEYLIST modifier [> info list];

  Returns an entry or count from the list of open key lists.

- GET RECORD modifier [> info list];

  Returns a sub-record or count from the current record.

- GET KEYLIST modifier [> info list];

  Returns a key item or count from the current key list.

- GET RECORDSET modifier [> info list];

  Returns a record number or count from the current record set.

- GET DATASETFIELDLIST modifier [> info list];

  Returns a field or count from the list of fields for the current dataset.

- GET FIELDDATASETLIST modifier [> info list];

  Returns a dataset or count from the list of datasets for the current field.

6.6 SEEK

The SEEK message positions the current pointer in the object to the item specified in data. The format of the message is:

SEEK OBJECTTYPE data [> info list];

- SEEK DATABASELIST dbname;
- SEEK INDEXLIST indexname;
- SEEK FIELDLIST fieldname;
- SEEK DATASETLIST datasetname;
- SEEK OPENRECORDSETLIST recordsetname;
- SEEK OPENRECORDLIST recordname;
- SEEK OPENKEYLIST indexname;
- SEEK RECORD FIELD fieldname;
- SEEK RECORD SUBREC sub-recordnumber;
- SEEK RECORDSET sequence-number;

There is a special case of the SEEK message as it relates to KEYLISTs. The KEYLIST may be SEEKed on a portion of the keyvalue. If the key is not found, the Server will return the keyvalue that is one greater than the requested result, or end-of-table if the request places the currency pointer outside of the list. The info list for the KEYLIST includes an indicator for an exact match.

- SEEK KEYLIST keyvalue;

6.7 CURRENT

The CURRENT message establishes the object type instance which will be current for that object. It also can be used to indicate the current search mode. The format is:

CURRENT objtype name [> info list];

- CURRENT DATASET datasetname;
- CURRENT FIELD fieldname;
- CURRENT RECORD recordname;
- CURRENT RECORDSET recordsetname;
- CURRENT DATABASE dbname;
- CURRENT KEYLIST keyname;
- CURRENT INDEX indexname;

6.8 CREATE

The CREATE message creates an instance of an object type. Most objects are already present in this database system and do not have to be created. The one type

which must be created is the RECORDSET.

CREATE RECORDSET recordsetname;

This message creates a recordset for the current database. It is assigned the name 'recordsetname', added to the OPENRECORDSETLIST, and made the current RECORDSET.

6.9 DELETE

The DELETE message deletes an instance of an object. Currently, the only object which can be deleted is a RECORDSET.

DELETE RECORDSET [recordsetname];

This message deletes the current record set or the record set specified by 'recordsetname'.

6.10 CLEAR

The CLEAR message resets an object to empty. The only object currently clearable is a record set.

CLEAR RECORDSET [recordsetname];

This message empties the current record set or the record set specified by 'recordsetname'.

6.11 FIND

The FIND message performs a search for the key within the current KEYLIST. The search always updates the current record set with the result of the search.

FIND [boolean op] keyvalue * and ? are wildcards

Searches for a key value for the current index or set of indexes and uses the boolean operator to update the current record set. If no boolean operator is specified, then the OR operator is assumed. An asterisk (*) is a wildcard character meaning 0 or more characters, a question mark (?) is a wildcard character meaning a single character replacement.

FIND [boolean op] keyvalue1 THROUGH keyvalue2
->

Searches for a range of values.

FIND [boolean op] =

>

>= keyvalue

<

<=

Provides relational searching. The '=' is the same as format 1 above. The others provide a range searching capability from the keyvalue to the beginning or end of the keylist.

6.12 RECORDSET

The RECORDSET message is used to perform operations between the current RECORDSET and another open RECORDSET set or use the NOT operator to remove the current RECORDSET.

- RECORDSET [boolean op] recordsetname;

This message will use the boolean operator (the OR is used if not specified) to merge the indicated record set 'recordsetname' with the current record set.

- RECORDSET NOT

This message will reverse the current record set.

- RECORDSET SELECT recordnumber [THROUGH recordnumber2]
- RECORDSET REMOVE recordnumber [THROUGH recordnumber2]

This message will add records to (SELECT) or take away records from (REMOVE) the current record set.

6.13 LOGIN

The LOGIN command is analogous to the way many mainframe systems handle security. The syntax is as follows:

LOGIN userid/password/key

All of the functionality for security is handled by the Server. This includes selective access to various databases and/or parts of databases. It also includes encryption. The user should never be aware that any particular database, to which access is not available, even exists. INDEXLIST, DATASETLIST, FIELDLIST, etc. should reflect only the ones available and should provide no clues that others do or do not exist. How the access for individual users is specified is left up to the implementation. An optional LOGOUT command can also be specified to remove the

current access profile.

6.14 SERVERLIST

A table called the SERVERLIST is a list of available Servers. For most systems, only one Server will be available. The purpose in providing this table is to allow the user and the Client to obtain pertinent information about the current Server, as indicated below. By default, the SERVERLIST should contain the Server name, maximum open databases, maximum open records, etc. It should allow the Client to obtain information about the current environment. The SERVERLIST could be used to determine, which of many Servers is available. The CURRENT command (such as CURRENT SERVER servername) is used to establish the Server to be used. This would allow a Client program to access databases from many different vendors at the same time.

Specifications for each CD-ROM application should appear automatically on screen, in printed matter that accompanies the CD-ROM application, and would include the following information at a minimum(All data is for example only):

Data Contents: Text, graphics, maps, audio, video, etc.
Data Index: Inverted, Keyword, Boolean, etc.
Proximity/Adjacency: Yes or No
Stop Words: Yes or No
Scope of Index: All words, stop list, graphics, etc.
Server:
- Copyright Company: Dataware, Fulcrum, SilverPlatter
- Server Name: CDAnswer, ROMWARE, Folio, etc.
- Max Open Databases: 500, 5, 27, etc
- Max Open Records: 500, 5, 27, etc
- Memory Requirements: 215 Kbytes
- Other Information:

Standards compliancy:
- Operating Environment: MS-DOS, OS/2, UNIX, Macintosh
- CD-ROM Reader: Yellow book
- CD-ROM Disc Vol/File: ISO 9660, HSG, Proprietary
- CD-ROM Access Protocol: CD-RDx
- Data Structure: 8211, SGML, 3-D Bit Mapped
- Server Protocol: CALS, etc.
- Data Item: None, DoD xxxxxxx, etc.
- Data Element: None, DoD xxxxxxx, etc.
- Other Standards: MIL-STD-28001, etc.
- Other Information: xxxxxxx (FREE FORM to EOF).

6.15 Table of Contents Processing

The object of the Server is to provide access to the data in a database in a consistent manner. The commands should be simple and as independent from each

other as possible. In a n-way tree structure, each block has n rows which potentially point to n blocks at the next lower level. Each of these blocks also contains an indefinite number of rows, each of which may point to a lower level block, and so on. The following commands will be available:

OPEN TOC name		This command opens to the Table of Contents block at the next level from the current block and current row.
CLOSE TOC name		Close the named TOC block.
CURRENT TOC name		Make the named TOC block the current block.
GET TOC	NEXT PREV CURR FIRST LAST	Get data within TOC block.
SEEK TOC row#		Position to a row number.

The first level of the table of contents can either be opened automatically when the database is opened and given a reserved name, or it can require the Client to open the first level and assign its own name.

Another table, the OPENTOCLIST table, is also provided which contains a list of open TOCs.

This system is analogous to the rest of the Server. This OPENTOCLIST is exactly equivalent to the OPENRECORDLIST. Opening TOC blocks is equivalent to opening records. Only the exact manner of opening is different and this is because of the hierarchical nature of the table of contents. A Client can access the table of contents data in many different ways without overburdening the Server with lots of overhead to keep track of.

7. Server OUTPUT

The data items available for a given request depend on the object type and the message. There are eleven different information list types. The following describes each information list type and which messages return it.

7.1 Field Description Information

Returned by GET/SEEK FIELDLIST or CURRENT FIELD.

Data Name	Data Size	Values
NAME	50	Defined name of the field
ID	4	Field id 1-255
SEL	1	Y/N (field selected)
DTYPE	8	Data type (See Appendix F for more information.)
MVALUE	1	Y/N (Multi-valued)
FORM	4	Date form or number size
SIZE	4	Field size
DECIMAL	4	Number of decimal pos

7.2 Dataset Description Information

Returned by GET/SEEK DATASETLIST or CURRENT DATASET.

Data Name	Data Size	Values
NAME	16	Defined dataset name
ID	4	Dataset id 1-255
SEL	1	Y/N (dataset selected)
STREC	10	Starting Record number
ENDREC	10	Ending Record number

7.3 Index Description Information

Returned by GET/SEEK INDEXLIST or CURRENT INDEX.

Data Name	Data Size	Values
NAME	32	Defined index name
ID	4	Index id 1-255
ITYPE	1	Index type (W or I)
STEM	1	Y/N (stemming)
DTYPE	1	Index data type (C,N,D)
COUNT	10	Number of keys

7.4 Database Description Information

Returned by GET/SEEK DATABASELIST or CURRENT DATABASE.

Data Name	Data Size	Values
NAME	16	DB name (OPEN DB message)
PATH	70	Actual file & path name
NRECS	10	Number of records in DB
MAXREC	10	Maximum record size in DB
DATAID	80	Defined Data ID field
OWNERID	80	Defined Owner ID field
PUBID	80	Defined publishers id

7.5 Open Record Description Information

Returned by GET/SEEK OPENRECORDLIST or CURRENT RECORD.

Data Name	Data Size	Values
NAME	16	Record name (OPEN REC)
DSID	4	Dataset ID of the record
RECNUM	10	Record number
RECSIZE	10	Record size

7.6 Open Keylist Description Information

Returned by GET/SEEK OPENKEYLIST or CURRENT KEYLIST.

Data Name	Data Size	Values
NAME	16	Keylist name (OPEN KL)
IDXID	4	Index id
ITYPE	1	Index type (W or I)
DTYPE	1	Index data type (C,N,D)

7.7 Open Recordset Description Information

Returned by GET/SEEK OPENRECORDSETLIST or CURRENT RECORDSET.

Data Name	Data Size	Values
NAME	16	Recordset name (CREATE)
COUNT	10	# of records in RECSET

7.8 Record Set Information

Returned by GET/SEEK RECORDSET.

Data Name	Data Size	Values
NUMBER	10	Record number
SEQ	10	Record sequence

7.9 Record Information

Returned by GET/SEEK RECORD.

Data Name	Data Size	Values
SUBREC	255	Sub-record data
SRTYPE	1	Sub-record type (S-start field, C-cont field)
FLDID	4	Field id of sub-record
SRLEN	4	Sub-record length
SRNUM	4	Sub-record number

7.10 Key Information

Returned by GET/SEEK KEYLIST.

Data Name	Data Size	Values
KEY	52	Key value
COUNT	10	Number of records.
FREQ	10	Frequency
EXACT	1	Y/N (exact match)

EXACT is always 'Y' with the GET message. With the SEEK message, EXACT can be 'Y' indicating an exact match or 'N' indicating a partial match.

7.11 Count Information

Returned by GET objtype COUNT.

Data Name	Data Size	Values
COUNT	10	Number of members

Appendix A: Error Messages

All messages yield a return code. This is a short (16 bit) integer value plus 30 bytes for message text from server (i.e. Volume Identifier for -402). A return code of zero indicates that the function was performed correctly. A positive return code is a warning message. This means that the command worked properly but there is no data returned. This is usually an end of table message. A negative number indicates that some sort of error occurred.

Name	Code
ERR_no_error Text: No Error	0

General Errors

ERR_par_syntax	-1
ERR_par_cmd	-2
ERR_par_obj	-3
ERR_not_init	-4
ERR_no_current_db	-5
ERR_syntax_err	-6
ERR_mem_fail	-7
ERR_already_init	-8
ERR_par_mod	-9
ERR_db_not_found	-10
ERR_invalid_obj	-11
ERR_dataset_not_found	-12
ERR_index_not_found	-13
ERR_field_not_found	-14
ERR_record_not_found	-15
ERR_keylist_not_found	-16
ERR_recordset_not_found	-17
ERR_no_keyfile	-18

OS Errors

ERR_os_open_err	-100
ERR_os_read_err	-101
ERR_os_seek_err	-102

Dictionary Errors

ERR_dict_fail	-200

Open/Close File Errors

ERR_max_dbs_exc	-300
ERR_max_oprecs_exc	-301
ERR_no_current_rec	-302
ERR_invalid_recnum	-303
ERR_already_opened	-304
ERR_max_open_keylist_exc	-305
ERR_max_rs_exc	-306

Access Errors

ERR_db_access_denied	-400
ERR_invalid_access_file	-401
ERR_mount_fault	-402

Get Errors/Messages

MSG_end_of_table	500
MSG_beg_of_table	501
ERR_empty_table	-500
ERR_no_current	-501
ERR_no_keys	-502
ERR_key_read_error	-503

Seek Errors/Messages

ERR_not_found	-600
ERR_field_not_selected	-601
ERR_seek_error	-602
ERR_invalid_sequence	-603

Find Errors

ERR_no_current_recordset	-700
ERR_no_current_keylist	-701
ERR_invalid_edi	-702
ERR_key_not_found	-703
ERR_different_dbs	-704

VMM Errors

ERR_reload_err	-800
ERR_unload_err	-801

Server Error

ERR_server_msg	-900

User Interrupt

ERR_abort	-999

More On Error Code

The Server returns an error code in addition to whatever information is requested. The following command syntax illustrates the error code return.

The return code (rc) variable contains the return code.

There are three types of return codes.

0 - Means there is no error.
<0 - Means that the command failed (ERROR).
>0 - Means that the command worked but further information is provided (WARNING).

WARNING CODES

There are, presently, two warning messages and both involve the GET command.

500 - End_of_table. This means that a GET table NEXT command was issued and there are no more entries. No Data is returned.

501 - Beg_of_table. This means that a GET table PREV command was issued and the current pointer is at the beginning of the table. No Data is returned.

ROMWARE Server Error Returns

ERROR CODES

There are many error return codes. They are divided into groups as follows:

-1 to -50 General Command Errors which may occur in a number of commands.

-100 to -150 DOS I/O errors. These errors may also occur in a number of commands. These are due to DOS reporting an error in response to an input request. This can be the result of physical media errors, bad hardware, corrupted copy of DOS, or database integrity corruption.

-200 to -250 Dictionary read errors. This is usually caused by the data dictionary file being corrupted.

-300 to -350 Open/Close command errors. This usually occurs when the object to be opened or closed is not present, or the maximum number of open objects has been exceeded.

-400 to -425 Access control errors. These occur when you attempt to open a database, to which you do not have access.

-500 to -550 GET errors. These codes are returned for a variety of reasons.

-600 to -625 SEEK errors. These codes are returned when a seek command fails. The most common reason is the not found message.

-650 to -675 TOC (Table of Contents) errors. These occur when access to the Table of contents fails.

-700 to -750 FIND errors. These are issued when the FIND command fails.

-800 to -825 Virtual memory errors. These are issued when the unloading or reloading of data fails.

-999 User Abort. The user requests an abort by pressing ctrl-break.

Specific Error Messages

GENERAL ERRORS:

-1 (ERR_par_syntax)

Parsing syntax error. This usually occurs, when the command line is empty or part of the command is missing. This usually refers to the first two parameters of the command, the command verb and the object. Errors in the rest of the command are indicated to the (-6) syntax error (see below).
ACTION: Correct command syntax.

-2 (ERR_par_cmd)

Parsing error. Invalid command.
ACTION: Use valid command.

-3 (ERR_par_obj)

Parsing error. Invalid Object or Table type.
ACTION: Use valid table type.

-4 (ERR_not_init)

The Server has not been initialized or the initialization failed. This should not happen to client programs. SYSTEM ERROR.

-5 (ERR_no_current_db)

There is no current database. This results when no database has been opened (or the open attempt has failed) and another command is issued. It can also occur when a database is closed and a new current database has not been established with the CURRENT command. Most commands require that a database has been opened and is current.
ACTION: Be sure that a database is opened and that currency has been established.

-6 (ERR_syntax_err)

Syntax error. This indicates an error in the data portion of the command. This means that the command and object are correct but that the data following is syntactically incorrect.
ACTION: Correct command syntax.

-7 (ERR_mem_fail)

This message occurs anytime memory is required by the Server and is unavailable. The command failed, usually, somewhere in the middle and the Server objects are not consistent. It is best to shut down the current operation by deleting the current record sets.
ACTION: The Server needs to be started with more static RAM,

more Expanded/Extended memory and/or a virtual disk buffer.

-8 (ERR_already_init)
This error occurs when the system initialization is performed while it is currently running. This should not appear to Client programs. SYSTEM ERROR.

-9 (ERR_par_mod)
Invalid command modifier. A command modifier is fixed text in the syntax of some of the commands, such as NEXT, PREV, etc.
ACTION: Correct syntax.

-10 (ERR_db_not_found)
DB name not found. This message occurs when the command is expecting an internal database name (assigned when the database is opened, by the OPEN DATABASE command), and the name is not present. This error could occur from a misspelling because a the database was closed.
ACTION: Use the correct name.

-11 (ERR_invalid_obj)
This error occurs when a command verb attempts an invalid action on an object. For example, attempting to OPEN a DATABASELIST.
ACTION: Check the documentation to be sure you understand the meaning of the command and/or table.

-12 (ERR_dataset_not_found)
Dataset name is not found. This message occurs, when the command is expecting a valid dataset name or dataset id number, and it is not present. It could be from a misspelling or when the wrong database is current.
ACTION: Use the correct name or id number.

-13 (ERR_index_not_found)
Index name is not found. This message occurs when the command is expecting a valid index name or index id number, and it is not present. The error could result from a misspelling or when the wrong database is current.
ACTION: Use the correct name or id number.

-14 (ERR_field_not_found)
Field name is not found. This message occurs, when the command is expecting a valid field name or field id number, and it is not present. This error could result from a misspelling, or because the wrong database is current.

ACTION: Use the correct name or id number.

-15 (ERR_record_not_found)
Record name is not found. This message occurs when the command is expecting an internal record name (assigned by the user with the OPEN RECORD command) and it is not present. It could result from a misspelling, because the wrong database is current, or when the record was closed.
ACTION: Use the correct name.

-16 (ERR_keylist_not_found)
Keylist name is not found. This message occurs when the command is expecting an internal keylist name (assigned by the user with the OPEN KEYLIST command), and it is not present. It could result from a misspelling, because the wrong database is current, or when the keylist was closed.
ACTION: Use the correct name.

-17 (ERR_recordset_not_found)
Recordset name is not found. This message occurs when the command is expecting an internal recordset name (assigned by the user with the OPEN RECORDSET command), and it is not present. It could result from a misspelling, the wrong database being current, or the recordset was closed.
ACTION: Use the correct name.

-18 (ERR_no_keyfile)
This message occurs, when there is no index files present (the database has not been indexed), and an operation requiring the index files is requested, such as FIND or OPEN KEYLIST.

-19 (ERR_inflist_err)
There is a syntax error in the information list provided with the command. Undefined data items do NOT cause a syntax error. If a data item is undefined, blanks are returned in the field. Syntax errors usually occur in the size and type portion of the line.

OS ERRORS:

-100 (ERR_os_open_err)
OS has returned an error while attempting to open a file.
ACTION: Make sure you have a valid database and media.

-101 (ERR_os_read_err)
OS has returned an error while attempting to read data from a data

file. This is an internal or media error.
ACTION: Make sure you have a valid database and media.

-102 (ERR_os_seek_err)
OS has returned an error while attempting to position the file pointer. This is an internal or media error.
ACTION: Make sure you have a valid database and media.

DICTIONARY ERRORS:

-200 (ERR_dict_fail)
This error occurs, when there is a problem loading or reloading the data dictionary into memory.
ACTION: Make sure you have a valid database and media.

OPEN/CLOSE ERRORS:

-300 (ERR_max_dbs_exc)
An OPEN DATABASE command is issued, when the maximum number of open databases are already opened.

-301 (ERR_max_oprecs_exc)
An OPEN RECORD command is issued, when the maximum number of open records are already opened.
ACTION: Close one of the open records.

-302 (ERR_no_current_rec)
An operation requiring a current record is issued and there is no current record.
ACTION: Open a record if none are opened, or establish currency (CURRENT command) on a previously existing record.

-303 (ERR_invalid_recnum)
The record number in an OPEN RECORD command is invalid.
ACTION: Use a valid record number.

-304 (ERR_already_opened)
The internal name supplied with the OPEN table command is currently in use.
ACTION: Use a different name or close the object with the name you wish.

-305 (ERR_max_open_keylist_exc)
An OPEN KEYLIST command is issued when the maximum number of open keylists are already opened. The current maximum

is 10.
ACTION: Close one of the open keylists.

-306 (ERR_max_open_rs_exc)
An OPEN RECORDSET command is issued when the maximum number of open recordsets are already opened. The current maximum is 30.
ACTION: Close one of the open recordsets.

ACCESS ERRORS:

-400 (ERR_db_access_denied)
An OPEN DATABASE command was issued for a database for which access is denied.
ACTION: Use another database or get access permission.

-401 (ERR_invalid_access_file)
This is a system error. The access file is corrupted.
ACTION: Get new access file.

GET ERRORS:

-500 (ERR_empty_table)
The table to which the GET command was directed has no entries.
ACTION: This depends on the reason the table is empty. If it is an OPENxxxLIST, such as OPENRECORDLIST, then open some objects must first be opened.

-501 (ERR_no_current)
A GET table CURR command is issued and the current entry in a table is not available. This could occur because the position was not established or it was lost due the GETting passed the end of table of beginning of table.
ACTION: Correct program logic. Make sure that an entry in the table is current.

-502 (ERR_no_keys)
A keylist is empty or the current dataset selection makes it empty.
ACTION: Use more datasets. Make sure the database was indexed properly.

-503 (ERR_key_read_error)
The index files in the database are in error.

SEEK ERRORS:

-600 (ERR_not_found)

The entry that was the target of the SEEK command is not present in the table. Check the spelling of an object name or for a number out of range.
ACTION: Correct the name or number.

-601 (ERR_field_not_selected)

This message is returned, when a SEEK RECORD FIELD 'fieldname' command is issued and the field in question has been de-selected (REMOVE command).
ACTION: Select the field or change fields.

-602 (ERR_seek_error)

This is an internal error, seeking within a record.

-603 (ERR_invalid_sequence)

This occurs when a sequence number for a recordset is out of range.
ACTION: Use a correct sequence number.

TOC ERRORS:

-650 (ERR_no_toc)

A command for the table of contents is issued but there is no table of contents for this database.
ACTION: Do not access the TOC if it does not exist. This message also allows the Client program to check if the TOC exists.

-651 (ERR_no_more_tocs)

An OPEN TOC command is issued and the current line of the current TOC block has no branches.
ACTION: Check the MORE data field for the current line before issuing an OPEN TOC command for that line.

-652 (ERR_no_current_toc_entry)

An OPEN TOC command is issued and there is no current entry in the current TOC block.
ACTION: Make an entry current (GET or SEEK command).

-653 (ERR_no_current_toc)

An OPEN TOC command is issued and there is no current TOC block.
ACTION: Make a TOC block current (CURRENT command).

-654 (ERR_max_optocs_exc)

An OPEN TOC command is issued when the maximum number of open TOC blocks are already opened. The current maximum is 20.
ACTION: Close one of the open TOC blocks.

-655 (ERR_toc_not_found)
TOC block name is not found. This message occurs when the command is expecting an internal TOC block name (assigned by the user with the OPEN TOC command) and it is not present. This error could occur from a misspelling, is because the wrong database is current, or when the TOC block was closed.
ACTION: Use the correct name.

FIND ERRORS:

-700 (ERR_no_current_recordset)
There must be a current recordset when a FIND command is issued. The results of the FIND command is reflected in the current recordset.
ACTION: Create a recordset or make one current.

-701 (ERR_no_current_keylist)
There must be a current keylist when a FIND command is issued. The FIND command searches the current keylist for the data.
ACTION: Open a keylist or make one current.

-702 (ERR_invalid_edi)
This is an internal error.

-703 (ERR_key_not_found)
The key requested on the FIND command line is not in the current keylist.
ACTION: Use another key.

VIRTUAL MEMORY ERRORS:

-800 (ERR_reload_err)
An error occurs while attempting to reload a data segment from virtual memory. This is usually an internal error.
ACTION: Verify that the media is correct, then call systems support.

-801 (ERR_unload_err)
An error occurs while attempting to unload a data segment to virtual memory. This is usually an internal error.
ACTION: Verify that the media is correct, then call systems support.

USER ABORT:

-999 (ERR_abort)

The user has pressed the Ctrl-C or Ctrl-break keys while the server was processing a request.
ACTION: Take whatever action, you wish to have happen, when the user aborts your Client program.

Appendix B: Client & Servers Explained

A Client is an operating system (OS) specific computer program that creates and send queries, via an OS-specific method to a Server and displays the data returned by that Server.

A Server is a memory available, OS-specific, program initiated before a Client that receives queries via an OS specific method. The Server then processes the request and/or returns an error code via an OS specific method.

The functions of the Client and Server are separated as follows:

Client Functions:

-- Screen Control: Any writing to the screen is the domain of the Client program. This is necessary to avoid screen collisions from two programs in the same screen. The exception to this is at the time the Server is started or installed.

-- Input Device Monitoring: Data from the keyboard, mouse, touch screen, light pen, bar code reader or other input device is the sole domain of the Client. In order to make changes to the Server the Client must be terminated and then the Server can be modified, terminated or replaced.

-- Control of Data Reflowing: When data is returned to the Client it may not fit in the data window. The client is responsible for reflowing the data to fit the window. Examples of this occurs when the Client changes fonts, changes window size, zooms, clips or scrolls returned data.

-- Handle Interrupts: The Client has the duty to specify when a search is interrupted. The limits can be on wait time, return data size, and/or number of hits (Word or Document hits is up to the Server).

-- Provide Help Functions: Search method assistance, screen sensitive help, and/or tutorial type help are the domain of the Client. This is not to be confused with error messages.

-- Expand Error Messages: The Client expands

error messages based on the pre-defined error message set. The method**, wording and language of the error message is up to the Client designer. The Client may choose to pass the Server message as a default.

** Note: This includes Error recovery functions.

-- Ask For Security Information: The Security Information request is generated by the Client for the user to complete and passed to the Server. See Server functions for more information.

Server Functions:

-- CD-ROM Reading Of Data Via OS Extensions: Only the Server may access data from the storage media. It may receive data by any available method.

-- Perform Searches And Boolean Operations: The Server performs all necessary functions needed to return requested data.

-- Format Return Data Records: The Server packages and formats the data. The Server also adds data type tags and inserts formatting information.

-- Uncompressing Data: If data is stored on the CD-ROM in a compressed form it is the function of the Server to decompress (via hardware or software) the data before returning it to the Client.

-- Activate Audio Play: If CD-DA data is requested, the Server will spin the disc and activate play of the audio.

-- Enforce Security: It is the function of the Server to verify and enforce security if it is available. Security functions are not required of CD-RDx but may be a requirement of a customer.

If messages are incorrect or the user does not have the appropriate clearance low level, security information is given to the Server then the Server may return "nothing available" instead of an error message. If no security information is passed to a Server that requires a login then an error message will be returned. If no security is available from the Server and security information is passed to the Server then the Server

must generate proper error messages.

Mandatory minimums for Clients and Servers:

Client:

-- The Client must be able to request and display the Basic Data Types. See the Appendix for more information on the Basic Data Types.

Server:

-- The Server must be able to communicate with the OS extensions to CD-ROM.

-- The Server must be able to return the Basic Data Types. The basic data types must be supported if the Server has that data available in any of the databases it is responsible for. See the Appendix for more information on the Basic Data Types.

Appendix C: Data Types

The data types in the data dictionary include the following general categories:

-- Text, Fielded Terms, Specific Program Formats, Executable.

-- Numerical Values, Numerical Array.

-- Graphics, 2-D Bit Map, 3-D Bit Map, 2-D Vector, 3-D Vector, Animated Bit Map, Animated Vector.

-- Digital Audio Music, Speech, Noise(Sound Effects), and digitized Sounds at various quality levels.

The Basic Data Types are required to be available to the Server and displayable by the Client. The Basic Data Types are suggested to include:

-- Screen formatted text.

-- TIFF raster graphics.

-- CGM vector graphics.

-- Program Names and Paths

-- Digital Audio

Appendix D: Data Dictionary

Text Data:

TEXT0001	Screen formatted text. Hard return every 72 characters. Word wrapping up to Server.
TEXT0002	Re-flowable text. Hard return at end of paragraph and blank line. No font info. No Highlight info.
TEXT0003	Re-flowable text. No font info. Highlight info around hits.

MULT0001	Multi valued field with separator as first character.

DATE0001	MM/DD/YY
DATE0002	DD/MM/YY
DATE0003	MM/YY
DATE0004	Month Day, Year
DATE0005	MM/DD/YYYY
DATE0006	DD/MM/YYYY
DATE0007	JULIAN DATE

SGML0001	SGML tagged data with DTD ISO 9660 file name and path.

DB3F0001	DBase III format.
DB4F0001	DBase IV format.

PROG0001	ISO 9660 Program name and path for specific OS.

Numerical Data:

NUMR0001	TBD

Image/Graphic Data:

TIFF0001	TIFF format graphic.

CGM_0001	CGM format graphic.

Digital Audio:

CDDA0001	CD Digital Audio recorded per Philips/Sony Red book. Track, Start Time, Run Time.

Appendix E: OS Specific Information

Information: The MS-DOS Version

Every CD-ROM software application complying to this standard will have a Server linking the CD-RDx Protocols to the user interface. An example is given for MS-DOS.

Beginning with DOS, we add the MicroSoft CD-ROM Extensions. On top of this rests the Server. On the platform of these three components the application programmer begins the task of creating a user interface that suits both the data and the operating environment.

The technique used for accessing DOS functions involves a combination of the interrupt vectors stored in the first 1K of memory, and the contents of the hardware registers built in to the 80x86 architecture. To activate a function within the operating system, the programmer loads the registers with relevant parameters, then 'calls,' or jumps, to the memory location stored in the appropriate vector. The operating system will return the results of the operation, or a message code, in the registers.

The Server extends the operation of the MS-DOS interrupts and the MicroSoft Extensions interrupt through the addition of interrupt 63H. In this way, the MS-DOS version of the Server behaves like any other low level component present within the system. This aspect of the design is critical in integration with variable video adapters, a variety of CD player configurations, asynchronous communication lines (such as modems and fax cards), and various other devices attached in the course of successfully completing an application on the DOS platform.

The Server, accessed via an interrupt, is a **passive memory resident** module which is loaded when DOS is called and is permanently put into memory, is available to any application program, and available to any program language. The TSR responds to ASCII message strings. Because of the run-time independence on any system and the near universal acceptance of the ASCII format, CD-ROM application programs that are cross platform in nature are possible. A standard protocol can be injected between the DOS machine and any other machine (eg. via a network). The DOS machine can be accessed through this protocol using ASCII strings embedded in standard communications packets.

Server Example for DOS

The Server is a memory resident program. The Server is started up before any application program. It occupies approximately xxxK of main memory in addition to the buffers specified when the Server is first invoked. The Server may be removed from memory when desired.

The delivery platform requires a CD-ROM drive and enough main memory to

support the developer's application, DOS, and the Microsoft CD-ROM Extensions (MSCDEX). This can typically be accomplished with as little as 512K main memory on the host system. The Server has no hard drive requirements but, the developer's application may require hard disk space to operate. The Server will use this device for temporary storage if the -v parameter is specified at start up. The Server has no requirement for video adapters since it does not write directly to the screen. However, the Server should not conflict with any standard video adapter that the developer chooses to use or require in the application.

The Server supports both extended and expanded memory with appropriate drivers loaded. Other optional equipment may also be used. The Server is passive and self contained within both memory allocation and code space. The executable is installed in a directory on the hard disk drive, and the PATH is set to include that directory. This will allow the executable to be used in many different areas of the hard disk without making multiple copies of the executable.

Server Programs:

DBSTSR.EXE - The TSR Server
DBSTERM.EXE - The TSR Server terminator
_DBSUTIL.EXE - A demonstration program

DBSTSR - The Server

Format: DBSTSR <options>

This program loads The Server into memory and keeps it there and sets up an interrupt interface for the Client applications.

Options:

-h - displays available options
-ixxx - modifies the default interrupt
-e - use expanded memory (if available)
-x - use extended memory (if available)
-s - use speed over size optimization
-mXXXXX - Number of bytes of memory allocated
-dDIR - Directory for Virtual overflow file
-rXXXX - Default record buffer size

DBSTERM - Remove The Server from memory.

Format: DBSTERM

DBSUTIL - Run Demonstration of The Server.

Format: DBSUTIL <options>

Options:

-h - Display available options
-q - Quiet mode

Important: The above is only a sample for DOS. CD-RDx

Information: Macintosh

This example is given for Apple Macintosh.

Beginning with HFS, Apple provides two INIT files (Foreign File Access and CD Audio Access. The addition of these two files to the system folder, allows access to ISO 9660 formatted
CD-ROMs. Running in this environment add the Server, also into the system folder. On the platform of these components the application programmer begins the task of creating a user interface that suits both the data and the operating environment.

The technique used for accessing INITs is outlined in the guidelines established by Apple. Apple developer services has several publications and products to aide the developer in the creation of INITs and the communication to INITs.

The Server, accessed via an standard memory location, is a **memory resident** program which is loaded when the Macintosh is started. It is available to any application program, and available to any program language.

Appendix F: Glossary

ASCII TEXT FORMAT This is a standard format used to represent characters of textual information. Each character has its own unique number between 0 to 127. The IBM Extended Character Set (numbered 128 to 255) is used for inner-operability between multiple operating platforms and general purpose programming languages.

CLASS A term from Object Oriented Programming that describes a collection of objects with common traits.

DATABASE A collection of one or more related datasets.

The DATABASE object knows its name and the path that it is currently using.

DATABASE DICTIONARY A collection of descriptions of the structures contained within a database. It may describe each dataset, index, and field for the database.

DATABASELIST The object that contains the names, location and other pertinent information about all of the databases currently open in the system. Its currency points to the database to which the current INDEXLIST, FIELDLIST, DATASETLIST, OPENRECORDSETLIST, OPENRECORDLIST, and OPENKEYLIST objects are associated.

DATASET A collection of records with a common set of fields. The DATASET object knows the range of records across which it is active, and the name defined with the build system.

DATASETFIELDLIST The object that contains the names of all of the fields for the current dataset.

DATASETLIST The object that contains the names, the subtended ranges of records, and other pertinent information relevant to datasets within the current database.

ELEMENT The CD-RDx Protocols consist of two dimensional tables. The rows of these tables are referred to as elements, and the components of these elements are referred to as columns or attributes.

FIELD A field is a collection of text, imagery, sound, etc. of related information. For example, the letters of a person's name may compose the contents of a field entitled "Name."

FIELD objects know the type of their contents and other valuable information.

FIELDDATASETLIST The object that contains the names of all datasets associated with the current field.

FIELDLIST The object that contains the names, types and formatting attributes of all of the fields associated with the current database.

FIELD SET The set that contains every instance of every field in a database.

INDEX A set of keys contained within the field set. See also "Word Index" and "Item Index."

INDEXLIST The object that contains the names, sizes and other pertinent information about all of the indexes within the current database.

INSTANCE A term from Object Oriented Programming that refers to a particular copy of an object. For example, "5" is an instance of an integer object in the number class.

ITEM INDEX An item index indexes every value or phrase in the field set across the entire database.

KEY An index key is a word or phrase that has a list of records associated with it.

KEYLIST The object that contains the entire contents of the field set against which it has been applied. Each instance of a keylist knows a count and frequency of the key within the entire database, or selected dataset range, or across records for every key.

LINE See subrecord.

OBJECT A term from Object Oriented Programming that describes an entity that contains internally managed data and the methods that manipulate the data.

The objects that form the CD-RDx Protocols can be thought of as tables. A table is a list of rows and columns. Each row is a different member of the set.

The objects within the CD-RDx Protocols always subscribe to this metaphor. Of course, not all object-oriented systems contain this reference; however, methods can be built into object-oriented systems directly that model the behavior of these tables.

OPENKEYLIST The object that contains the names of the labels defined by the developer for all KEYLIST objects they have opened.

OPENRECORDLIST The object that contains the record numbers of all records that have been opened by the developer.

OPENRECORDSETLIST The object that contains the developer-defined labels and the gross size of each of the family of record sets currently opened by the developer.

RECORD A record is composed of one or more fields of related information.

The RECORD object knows the contents of all subrecords and the field identifier associated with each of them. In addition, the RECORD object keeps other pertinent information for each subrecord. Subranges can be defined within the RECORD object through use of the SELECT FIELDS message.

RECORDSET The object that contains the record numbers of every record that has satisfied FIND statements issued by the developer.

The RECORDSET message is used to combine record sets, or to explicitly add and remove specific records or ranges of records to the current RECORDSET object.

SUBRECORD A record, or any field within that record, can be as large as the available disc space. In order to process a field of indefinite size, each field is divided into a series of subrecords. Each field in a record has one or more subrecords.

WORD INDEX An index that contains every word from every field across the database.

Appendix G: Questions & Answers

Q. What does CD-RDx involve?
A. Like many other computer operations, the CD-RDx standard incorporates a **Client/Server** approach, meaning that the **Client** (what the user interacts with, such as the menus and/or commands for a particular software application, command structure, graphical user interface or pull down menu) is separate from the **Server** (the unique retrieval engine and data index requirements). Therefore, CD-RDx describes in detail the **Black Box** through which both sides are to communicate with one another.

Q. What's in the Black Box?
A. The Black Box is not a thing; rather it is a set of rules that provide a level of abstraction so that both sides -- the **Client** and the **Server** -- can communicate with one another. These rules consist of commands and 2-dimensional tables as described in detail in the CD-RDx standard.

Q. What will an organization need to do to adhere to this standard?
A. If an organization is purchasing commercially available CD-ROM applications or developing a solicitation for the delivery of digital data on CD-ROM discs, then an explicit statement requiring that the discs contain the necessary index/retrieval engine drivers (**Server**) conforming to this standard is needed.

If, on the other hand, an organization is receiving CD-ROM discs with the **Server**, then the organization must have the necessary user interface drivers (**Client**) conforming to the standard in order to use the discs.

Q. Does this mean that an organization has to create a new **Client** for each **Server**?
A. Absolutely NOT. In fact, quite the opposite. Once a **Client** is created for a top-level user interface, such as Windows 3.0, Motif, Hypercard, or a software application program, such as SAS, SPSS, Lotus 1-2-3, Excel, Word, WordPerfect and so on, then these types of user interfaces will work with ANY CD-ROM disc that have the requisite **Server**.

Q. What will happen to existing CD-ROM discs that do not contain a **Server**?
A. These discs continue to be useful just as they are now; however, unlike those discs containing a **Server**, the existing discs will have to be installed with each use and the user interfaces will be specific to that disc.

Q. Can a user interface now used for a specific CD-ROM retrieval engine be used for ALL discs containing a **Server**?
A. Yes, if the software vendor has designed the retrieval engine and user interface as separate functions and if the user interface has the necessary **Client** drivers.

Q. What are the major implications of the CD-RDx standard?
A. Presently, most CD-ROM publishers purchase CD-ROM software based on the user interface and the features that the user interface provide. With this standard,

potential publishers will not have to concern themselves about a user interface -- since these decisions will reside with users and the user interfaces/**Client** the users select. Instead, publishers will be dealing with performance of the software, the structure of the index(es) and data preparation requirements of the software program. Although this is exactly what potential CD-ROM publishers should be concerned with, most software vendors would like potential licensees to consider less important factors, such as sticky notes, printout capabilities, Boolean operators and licensing fees.

> For CD-ROM publishers, this standard could also move the perspective of potential buyers from, again, features as observed in the user interface to quality, accuracy and completeness of the data. Like a book, CD-ROM should convey appropriate, complete and useful information. A potential purchaser is now wise enough to detect a beautifully bound book with irrelevant information. This "next-level-of-intelligence" will occur for CD-ROM purchasers and users as well, as a result of this standard.

Q. What are the risks? What will users or purchasers of CD-ROMs have to give up?
A. Nothing. The standard provides only rewards for users and purchasers. Instead of having to learn each software program with each new disc, users will select the user interface of their choice and access data on any one or a combination of discs with this single user interface. Users may develop a single search and then apply it to one or more databases on separate discs published by different vendors.

Q. What will CD-ROM producers have to do to implement this standard?
A. Assuming that the specific CD-ROM indexing and retrieval software selected comes with a **Server**, then the answer is nothing different than what is required of the program now. The software developer/vendor, however, will have to create its own **Server** driver **ONCE** for each OS.

Q. What will CD-ROM users have to do to implement this standard?
A. Obtain or develop a **Client** driver for each top-level or application-specific user interface as desired. For example, the user may want to access full-text information from Windows 3.0, from a word processing program, and also from "XYZ's" menu. **Client** drivers will have to be developed for each of these three programs.

Q. What will the CD-ROM industry have to do to implement this standard?
A. Developers of CD-ROM indexing retrieval software will each have to develop their own **Server** drivers to work with their proprietary software engines and indexing schemes. Software and/or third-party developers of top-level interfaces and of software applications will have to develop **Client** drivers to work with a number of programs.

Q. Does this mean that a user interface does not come automatically with a CD-ROM application?
A. Not necessarily. A CD-ROM producer/publisher may decide to deliver a user interface for the database as well as the **Server** on the same disc. The standard does not limit publishers nor users from providing or using a user interface that comes with

the software as we are used to at the present time.

Q. What is the proposed course of action with respect to implementing the CD-RDx standard as an international standard?
A. The DCI Intelligence Information Handling Committee (IHC) agrees on the need for a CD-RDx standard for use throughout the Intelligence Community. Simultaneously, IHC is briefing civilian agencies, the Department of Defense, NISO/NIST and other groups in the public and private sectors on the details of the CD-RDx standard. It is also began funding the development of **Client** drivers for some "most-used" user interfaces in the DOS, UNIX, OS/2 and Macintosh operating system environments so that CD-ROM discs containing the requisite **Server** driver can be demonstrated to work in different operating systems within different user interfaces.

Q. Does this CD-RDx standard conflict with any of the standardization activities within the Computer-Aided Acquisition and Logistics Support (CALS) system?
A. The answer is NO.

Q. How does the CD-RDx standard address the accessing, retrieval and display of graphics?
A. Images are passed as data to the **Client**. It is up to the **Client** to support a specific type of graphics. In short, CD-RDx in no way limits the use of graphics, nor the type of graphics.

Q. How does CD-RDx deal with MARK RECORD, SORT RESULTS and similar functions?
A. MARK and SORT are functions of the **Server**, not a part of a CD-ROM access and retrieval standard. It is up to the Client to understand the marked data. The Client can resort data.

Q. How does CD-RDx deal with ADJACENCY and PROXIMITY?
A. A modification to the FIND command with ADJ and PROX have been implemented.

Q. Are the specifications and the internal components of the specifications extensible?
A. Yes. Extensibility is in the data display. The data return types include extensible/proprietary data types. NOTE: Proprietary data types can only be used by Clients that understand that type of data. (i.e. The Server company's Client.)

Q. How will multiple simultaneous protocols be defined? Where will the protocol handlers be located? Does CD-RDx take into account multi-relational, multi-file databases?
A. Implementation of the **Server** is not limited to the CD-RDx standard. If a particular CD-ROM retrieval engine supports multi-anything, then that particular software developer should write these capabilities into the **Server**.

Q. Doesn't a memory resident program, such as a Terminate and Stay Resident (TSR), limit acceptance of the CD-RDx standard?
A. The primary purpose of CD-RDx is to transcend the need for writing a **Server** for each computer or interface environment. Without a memory resident **Server**, it would not be a standard acceptable in all environments.

Q. Is there any reason why the **Server** must be a TSR? There are any number of alternative ways to implement such a **Server**, some of which have significant advantages over a TSR based implementation.
A. A Terminate and Stay Resident (TSR) program may be used, but it is only needed in the DOS part of the CD-RDx standard. The **Server** just needs to be an independent program in memory.

Q. Where in this CD-RDx standard are security issues addressed?
A. If security and audits are an issue, then they may be added to the **Server**. Such functional capabilities are part of the **Server**.
Q. The limits for PATH and FileName in the POSiX specification is more liberal than those for MS-DOS. Can the length limitations be increased?
A. **Clients** operating in different environments can specify any length for path and file names.

Q. How will the user process records/fields that are huge, such as 1 MB or larger?
A. CD-RDx imposes no limitations at all. The **Client** sets buffer length. The **Server** merely writes to the buffer. The buffer may be written to the full available size of any storage device available. This allows the Server to use both RAM and mass storage.

Q. What techniques are used in CD-RDx for dealing with Database Passwords, Field Passwords and "restricted" records?
A. See LOGIN command and Appendix B (Server Functions)

Q. How does CD-RDx work with networks.
A. The ability to access databases that reside on a networked drive is a function of the Server. Establishing the link to the network is not in the realm of how data is passed between the Client and Server. When the database retrieval power is on a network server, it is the responsibility of the Server running on a local computer to request and process the remote information. The possible exception to this is in the UNIX world where the remote Server has established an ID on the CD-RDx Socket.

Q. ISO 9660 has three levels of implementation. Does CD-RDx have a similar approach to implementation so that developers can get "up and running" faster?
A. CD-RDx supports everything and nothing can be left out. However, developers of **Clients** and **Servers** can certainly begin with the basics and provide increasing levels of sophistication over time. The basic data types must be supported if the Server has that data available in any of the databases it is responsible for.