IBM DB2 Digital Library

Guide to Object-Oriented and Internet Application Programming

Version 2 Release 4

SC26-8868-02

IBM DB2 Digital Library

Guide to Object-Oriented and Internet Application Programming

Version 2 Release 4

SC26-8868-02

Note

Before using this information and the products it supports, be sure to read the general information under “Appendix. Notices” on page 221.

Third Edition (December 1998)

This edition applies to Version 2 Release 4 of IBM DB2 Digital Library (product number 5765-C86) and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of product.

Order publications through your IBM representative or the IBM branch office serving your locality.

Contents

About this guide ix
Who should read this guide ix
Where to find more information ix
How to send your comments x

Chapter 1. Introducing the Digital Library solution 1
Digitizing da Vinci 1
The need 1
The solution 1
The Digital Library solution 2
Triangle architecture 3
Image search server 3
What's new in Version 2 4
What's new in Version 2.4 5

Chapter 2. Digital Library application programming concepts 7
Understanding Digital Library data concepts 8
Items 8
Item attributes and index classes 8
Item parts 8
Documents and folders 8
Understanding the dynamic data object concepts 8
Dynamic data objects (DDO) 9
Extended data objects (XDO) 9
Representing multimedia content 9
Comparing DDO/XDOs with attribute values and item-parts 10
Persistent object identifiers (PID) 10
Understanding data access through datastores 10
Datastores in a Digital Library session 10
Datastores and DDOs 10
Comparing schema with index classes and attributes 11

Chapter 3. Using the C++ application programming interface 13
General handling and tracing information 13
Handling large objects 13
Exception handling 14
Tracing Information 14
Multi-Search facilities 14
Understanding a Digital Library datastore 15
Establishing a connection 15
Setting and getting datastore options 15
List servers 15
List schema and schema attributes 16
Programming tips 17
Understanding datastore text search (Text Search Engine) 17
Establishing a connection 17
Getting and setting datastore TS options 18
List servers 18
List schema. 18
Using DKAny 19
Type code 19
Memory management in DKAny 19
Using constructors 19

Getting the type_code . . . 19
Assignment to DKAny: DKAny = <TYPE>. . . . 20
Assignment from DKAny: TYPE = <DKAny>. . . . 20
Displaying DKAny . . . 20
Destroying DKAny . . . 21
Programming tips . . . 21
Collections and iterators . . . 21
Sequential collection . . . 21
Sequential iterator . . . 22
Memory management in collections . . . 23
Sorting the collection . . . 24
Programming tips . . . 24
Using DDO (Dynamic Data Object) . . . 24
Creating a DKDDO . . . 25
Pid . . . 25
Adding data-items and properties . . . 25
Adding properties to a DDO . . . 26
Setting and getting data_item values . . . 26
Getting the properties . . . 27
Displaying the whole DDO . . . 27
DDO representing Digital Library information . . . 28
DDO representing Text Search Engine information . . . 29
Creating and using a DKPARTS attribute . . . 30
Creating and using DKFOLDER attribute . . . 31
Deleting a DDO . . . 32
Using XDO . . . 32
XDO Pid . . . 32
XDO data members . . . 33
Indexing XDOs by search engines . . . 33
Programming tips . . . 33
XDO as a part of DDO instead of stand alone XDO . . . 34
Querying the datastore . . . 49
Differences between dkResultSetCursor and DKResults . . . 49
Parametric query . . . 50
Text query . . . 52
Result set cursor . . . 57
Open and close the result set cursor to re-execute the query . . . 57
Set position and get position in a result set cursor . . . 58
Creating a collection from a result set cursor . . . 58
Retrieving a document or folder . . . 59
Retrieving parts . . . 59
Retrieving a folder . . . 60
Creating, updating, and deleting documents or folders . . . 61
Creating a new document . . . 61
Updating a document or a folder . . . 63
Updating parts . . . 63
Updating folders . . . 64
Deleting a document or a folder . . . 65
Queryable collection . . . 66
Getting the result of a query . . . 66
Evaluating a new query . . . 66
Using queryable collection instead of combined query . . . 67
Programming tips . . . 67
Combined query . . . 67
Combined parametric and text query . . . 68
Using a scope . . . 68

Ranking . . . 69
Programming tips . . . 69
Text Search Engine . . . 69
Programming tips . . . 69
Boolean query . . . 70
Free text query . . . 70
Hybrid query . . . 70
Proximity query . . . 70
GTR query . . . 71
Loading data to be indexed by Text Search Engine . . . 71
Environment settings . . . 71
On AIX . . . 71
On Windows NT . . . 72
Building C++ programs on AIX . . . 72
Building C++ programs on Windows NT . . . 72
Setting console sub system for code page conversion on Windows NT . . . 73
Understanding the workflow and workbasket functions . . . 73
Understanding the workflow service . . . 73
Establishing a connection . . . 73
List workflows . . . 74
List workbaskets . . . 74
List items in a workflow . . . 75
Executing a workflow . . . 75
Creating a workflow . . . 76
Creating a workbasket . . . 76

Chapter 4. Using the Java application programming interface . . . 77
General Handling Information . . . 77
Handling large objects . . . 77
How to catch a DKException . . . 78
Tracing Information . . . 78
Multi-Search facilities . . . 78
Client/Server architecture . . . 79
Packaging for the Java environment . . . 79
Package hierarchy . . . 79
Differences from the C++ application programming interface . . . 80
Understanding Datastore DL . . . 80
Establishing a connection . . . 80
Setting and getting datastore options . . . 80
List servers . . . 81
List schema and schema attributes . . . 81
Understanding datastore text search (Text Search Engine) . . . 82
Establishing a connection . . . 82
Getting and setting datastore TS options . . . 83
List servers . . . 83
List schema . . . 83
Collections and iterators . . . 84
Sequential collection . . . 84
Sequential iterator . . . 84
Sorting the collection . . . 85
Using DDO (Dynamic Data Object) . . . 85
Creating a DKDDO . . . 86
Pid . . . 86
Adding data-items and properties . . . 86
Adding properties to a DDO . . . 87
Setting and getting data_item values . . . 87

Getting the properties . . . 88
Displaying the whole DDO . . . 88
DDO representing Digital Library information . . . 89
DDO representing Text Search Engine information . . . 90
Creating and using DKPARTS attribute. . . . 91
Creating and using DKFOLDER attribute . . . 92
Using XDO . . . 92
XDO Pid . . . 93
XDO data members. . . . 93
Indexing XDOs by search engine . . . 93
Programming tips . . . 93
XDO as a part of DDO instead of stand alone XDO . . . 94
Querying the datastore . . . 107
Differences between dkResultSetCursor and DKResults . . . 107
Parametric Query . . . 107
Text Query . . . 109
Result set cursor . . . 114
Open and close the result set cursor to re-execute the query . . . 114
Set position and get position in a result set cursor . . . 115
Creating a collection from a result set cursor . . . 115
Retrieving a document or folder . . . 116
Retrieving parts . . . 116
Retrieving a folder . . . 117
Creating, updating, and deleting documents or folders . . . 117
Creating a new document . . . 117
Updating a document or a folder . . . 119
Deleting a document or a folder . . . 121
Queryable collection . . . 121
Getting the result of a query. . . . 121
Evaluating a new query . . . 122
Using queryable collection instead of combined query . . . 122
Programming tips . . . 123
Combined query . . . 123
Combined parametric queries and text queries . . . 123
Using a scope . . . 124
Ranking . . . 124
Programming tips . . . 124
Text Search Engine . . . 125
Programming Tip . . . 125
Boolean query . . . 125
Free text query . . . 125
Hybrid query . . . 126
Proximity query . . . 126
GTR query . . . 126
Loading data to be indexed by Text Search Engine . . . 126
Environment settings and component names . . . 127
On AIX . . . 127
On Windows NT . . . 128
Client/Server in Java . . . 128
Digital Library datastore connect and disconnect in the client side . . . 128
Setting up the environment . . . 129
Programming tips . . . 131
Understanding the workflow and workbasket functions . . . 131
Understanding the workflow service . . . 131
Establishing a connection . . . 132
List workflows . . . 132

List workbaskets . . . 132
List items in a workflow . . . 133
Executing a workflow . . . 133
Creating a workflow . . . 134
Creating a workbasket . . . 134

Chapter 5. Using the Sample Java Applets and Servlet . . . 137
Connect applet . . . 137
Query applet . . . 138
View applet . . . 141
Retrieve servlet . . . 142
Java application on a Digital Library client . . . 142
Local applet access on a Digital Library client . . . 143
Remote applet access . . . 144
Display part information . . . 145
Index a part . . . 147
Set content viewer . . . 150
Load video streams . . . 151
Display video stream parts information . . . 153
Play video streams . . . 154
Working in conjunction with Dynamic Page Builder . . . 154

Chapter 6. Using the ActiveX application programming interface . . . 157
Multi-search facilities . . . 157
Installation . . . 158
Understanding Digital Library datastore DL . . . 158
Digital Library datastore connect and disconnect . . . 158
Set and get Digital Library datastore options . . . 159
Digital Library datastore list servers . . . 159
Digital Library datastore list schema and schema attributes . . . 159
Understanding datastore text search (Text Search Engine) . . . 160
Text Search Engine datastore connect and disconnect . . . 160
Set and get TS datastore options . . . 161
Text Search Engine datastore list servers . . . 161
Text Search Engine datastore list schema . . . 161
Collections and iterators . . . 162
Using DDO (Dynamic Data Object) . . . 163
Pid . . . 163
Adding data-items and properties . . . 163
Adding properties to a DDO . . . 164
Setting and getting data_item values . . . 164
Getting the properties . . . 164
Displaying the whole DDO . . . 165
DDO representing Digital Library information . . . 166
DDO representing Text Search Engine information . . . 167
Creating and using dx_DKPARTS attribute . . . 167
Creating and using dx_DKFOLDER attribute . . . 168
Deleting a DDO . . . 169
Using XDO . . . 169
XDO as a part of DDO instead of stand alone XDO . . . 169
XDO data members . . . 170
Indexing XDOs by search engine . . . 170
Programming tips . . . 170
Stand-alone XDO . . . 171
XDO in datastore . . . 175
Querying the datastore . . . 175

Parametric query . . . 175
Text query . . . 177
Result set cursor . . . 182
Retrieving a document or folder . . . 183
Retrieving a document . . . 183
Retrieving parts . . . 184
Retrieving a folder . . . 184
Creating, updating, and deleting documents or folders . . . 185
Creating a new document . . . 185
Updating a document or a folder . . . 186
Deleting a document or a folder . . . 188
Advanced topics . . . 189
Queryable collection . . . 189
Combined query . . . 190
Text Search Engine . . . 192
Handling large objects . . . 194
Client/Server mode . . . 194
Understanding the workflow and workbasket functions . . . 195
Understanding the workflow service . . . 195
Establishing a connection . . . 196
List workflows . . . 196
List workbaskets . . . 196
List items in a workflow . . . 197
Executing a workflow . . . 197
Creating a workflow . . . 198
Creating a workbasket . . . 198

Chapter 7. Using the Dynamic Page Builder . . . 201
Configuring the Dynamic Page Builder with Net.Data . . . 201
The ENVIRONMENT statement (DTW_DLDPB) . . . 202
The MACRO_PATH statement . . . 202
The INCLUDE_PATH statement . . . 202
The HTML_PATH statement. . . . 202
Dynamic Page Builder functions . . . 202
API functions . . . 202
Input parameters . . . 203
Inline data . . . 204
Variable definition . . . 205
Special output variable. . . . 205
Developing a Net.Data macro for the Dynamic Page Builder . . . 205
Sample macro I . . . 206
Sample macro II . . . 210
Improving performance . . . 216
Invoking the wizard . . . 216
Starting the Dynamic Page Builder sample . . . 217
Web server configuration . . . 217
Connection manager setup . . . 218

Appendix. Notices . . . 221
Programming interface information . . . 222
Trademarks . . . 222

Glossary . . . 223

Index . . . 227

About this guide

This guide describes how to use and modify the C++, Java, ActiveX, and Dynamic Page Builder portion of the IBM DB2 Digital Library Software Developer's Kit. The kit lets you create multi-search query applications that search for documents in Digital Library and have a World Wide Web interface.

This guide includes:

- Descriptions of how the various components of work.
- Tips for identifying application requirements as you create a query application.
- Details of how the sample code works and how to tailor it to your needs.

For additional information about programming with other Digital Library components in C language, see the Application Programming Guide for the platform you are using.

Who should read this guide

This guide is intended for application programmers with some or all of the following skills:

- Experience with either C++, Java, ActiveX, HTML
- Familiarity with relational database concepts
- Knowledge of the DDO/XDO protocol

Where to find more information

IBM Digital Library Version 2 Release 4 includes a complete set of information to help you plan, install, administer, and use your system. Each publication is provided in two different formats on the *IBM Digital Library* CD-ROM.

Acrobat PDF files

The Digital Library publications in the Adobe Acrobat Portable Document Format (PDF) are in the PDFBOOKS directory. The reference modules are in the PDFBOOKS\REF subdirectory.

You can view the PDF files online using the Adobe Acrobat Reader 3.0 for your environment: AIX, Windows NT, Windows 95, or Macintosh.

PostScript files

The Digital Library publications in PostScript (PS) format are in the PSBOOKS directory. The reference modules are in the PSBOOKS\REF subdirectory.

You can print the PostScript files on printers equipped with PostScript fonts.

Table 1 shows the Digital Library publications included on the CD:

Table 1. Digital Library publications

Order Number	Digital Library publication
GC26-8623-04	*Planning and Installation Guide*[1]
GC26-8626-03	*System Administration Guide*
SC26-8868-02	*Guide to Object-Oriented and Internet Application Programming*

Table 1. Digital Library publications (continued)

Order Number	Digital Library publication
SC26-8654-02	*Application Programming Guide for AIX*
SC26-8651-01	*Application Programming Guide for Windows*
SC26-9033-01	*Application Programming Guide for Macintosh*
SC26-8652-03	*Application Programming Reference*
SH12-6317-01	*Text Search Engine: Application Programming Reference*
SC26-9199-04	*Client for Windows User's Guide*
SC26-8625-02	*Messages and Codes*
SC26-9283-01	*Image Search User's Guide and Reference*[2]

Notes:

1. You receive a printed copy of the *Planning and Installation Guide* when you receive the product. You can also order a printed copy of the *Messages and Codes*.
2. Image search APIs are documented in the *Image Search User's Guide and Reference* and are not included in the *Application Programming Reference*

Copying the PDF files: To copy the PDF files from the CD-ROM to your hard drive:

1. Change directory to PDFBOOKS
2. Copy *.PDF files to your designated hard drive directory
3. Change directory to REF
4. Copy *.PDF files to your hard drive directory

Installing the PDF reader: The Adobe Acrobat Reader is available in the ACROBAT directory of the CD-ROM and from http://www.adobe.com. To install Acrobat Reader, decompress the program files for your platform and follow the instructions in the Acrobat installation program or the installation text file.

On AIX, untar the .tar file and read INSTGUID.TXT.

On Windows platforms, run the executable file.

On Macintosh, expand the self-extracting archive and run "Install Acrobat Reader."

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other **Digital Library** documentation. You can use either of the following methods to provide comments:

- Send your comments from the Web. Visit the Online RCF for IBM Data Management page at:

  http://www.software.ibm.com/data/rcf

  You can use the page to enter and send comments.
- Send your comments by e-mail to comments@vnet.ibm.com. Be sure to include the name of the product, the version number of the product, the name and part number of the book (if applicable). If you are commenting on specific text, please

include the location of the text (for example, a chapter and section title, a table number, a page number, or a help topic title).

Chapter 1. Introducing the Digital Library solution

The Digital Library solution provides leading-edge technology to bring the world's resources to your computer desktop. Digital Library can help you maximize the value of your information and multimedia assets while protecting digital intellectual property. The open, scalable nature of Digital Library allows a solution to be uniquely suited to your enterprise.

With Digital Library's object-oriented application programming interfaces, developers can write Internet applications, which interact with users through a Web browser. Digital Library's search servers provide search capabilities that enable users to quickly access all forms of multimedia data. Digital Library's rights management resources include both visible and invisible watermarking techniques that help you protect multimedia assets.

This chapter gives an overview of the Digital Library solution. A fictitious scenario about a university art museum with works by and about the great artist and inventor, Leonardo da Vinci, demonstrates the wide range of Digital Library's functions.

Digitizing da Vinci

A large university has an extensive collection of articles, videos, and audiotapes about Leonardo da Vinci. The university also has an impressive collection of da Vinci's art and writings. Curators of these works want to ensure that they are readily available to university patrons: visitors, students, and academics; but the curators must also protect these priceless items. The curators plan to convert all the information, artwork, and multimedia content into digital data that users can quickly and easily search and access.

The need

Many original da Vinci works cannot be displayed publicly because of physical deterioration. To preserve the public's ability to appreciate these fragile pieces, the university needs to convert the originals to digital formats. The curators can then store the originals in a vault, while the public has quick and easy access to high-resolution digital reproductions.

Many different users need access to the da Vinci library, from art students to researchers to commercial ventures wanting to use images of da Vinci's works. The curators want to restrict access to certain items, while providing unlimited access to others. They want to exploit the easy-to-use interface provided by Internet applications. They also need to ensure that items owned by the university are protected from unauthorized use.

Because the university has a variety of computer systems, which run AIX and Microsoft Windows NT, the solution needs to provide access from multiple operating systems.

The solution

The university selects Digital Library because its comprehensive technologies allow storage, management, and retrieval of digital data objects over networks. The

university staff works closely with IBM research, development, and service organizations to develop a customized solution that includes:

- A precision capture subsystem
- The database and storage subsystem
- An Internet gateway for end-user access

Capture subsystem

The capture subsystem consists of a high-resolution IBM scanner and an IBM workstation equipped with a precision capture application. The university uses the capture application to digitize images and documents.

As each item is scanned, information about the item is entered into a DB2 database. Samples (lower resolution than the display images) and thumbnail images (small-sized versions) are also produced to provide more efficient access via the Internet.

The capture application uses the Digital Library rights management APIs to add watermarks to any object or document copyrighted by the university.

Database and storage subsystem

The university chooses IBM DB2 Universal Database, because it provides powerful data management and ease of use. Digital Library uses the DB2 database on the library server to manage the information about stored objects and provide pointers to the objects in the da Vinci collection. DB2 databases also contain information about how to manage the storage of the objects on the object server.

ADSTAR Distributed Storage Manager (ADSM) works with Digital Library's object servers to store the objects to DASD, tape, and optical storage devices. ADSM also provides data integrity and manages backups.

Videos in the da Vinci collection are stored separately on IBM DB2 Digital Library VideoCharger Version 2.0, which works with an object server and Digital Library to deliver audio and video objects over the Internet.

Internet gateway

The Internet gateway consists of Internet applications, which enable users to search for and request objects. Using the APIs in the Digital Library software developer's kit, the university provides access to the da Vinci collection through any Web browser. From their desktops, users can browse the virtual museum or search for and display a particular image or document. Watermarking on copyrighted images and documents protects intellectual property, deterring anyone from using items in the collection without authorization.

The Digital Library solution

Digital Library is a comprehensive product; its components work together to provide solutions that are uniquely suited to your needs. Centered on a triangular client/server architecture, the solution includes a Java-based tool for managing the system and a text search component for full-text searches. VideoCharger can be used with Digital Library to access video and audio files.

Triangle architecture

The Digital Library architecture is based on a triangular client/server model comprised of one *library server*, one or more *object servers*, and one or more clients. A client sends a request to the library server. The library server forwards the request to an object server and responds to the client's request. The object server then responds to the library server's request, and delivers the requested digital data object to the client.

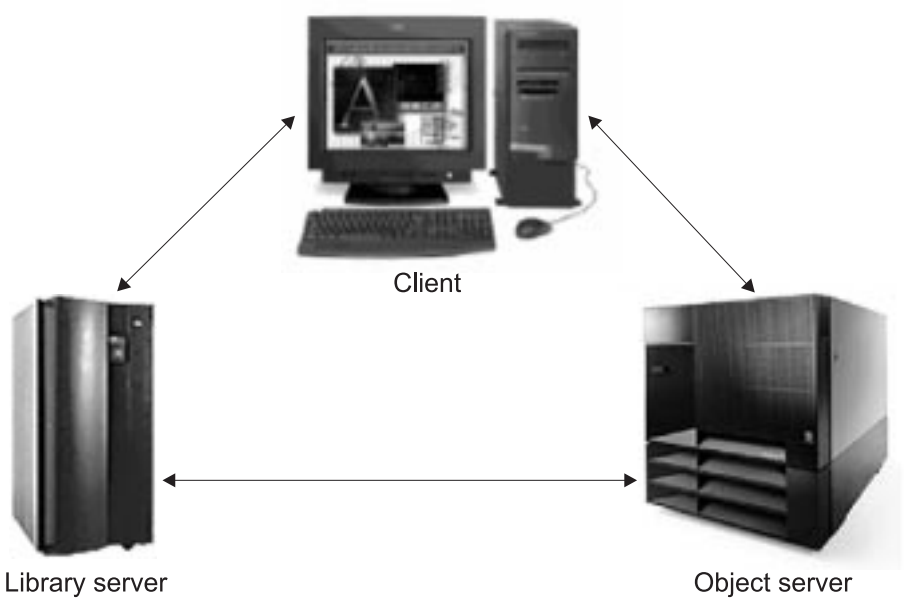


Figure 1. Client/server relationship in the Digital Library system

This architecture provides the following advantages:

- Support for multiple, distributed object servers allows you to place storage for digital objects close to the users who need to access these objects frequently. This support is especially important for delivery of large multimedia objects.
- Support for heterogeneous servers allows you to use the system for all kinds of data (including streamed data), while optimizing the processing of individual data types.
- Client communication through the library server ensures the integrity of data objects. A client can access objects only by requesting them through the library server. The library server contains a database with information such as authorized users of the system, indexes of all stored objects, and access control for each object.
- Separation of client applications, indexes, and data makes applications independent of the data's location on the servers.
- The scalable, open architecture allows integration of additional object servers on the same or different operating system and scaling from one operating environment to another.

Image search server

The Digital Library image search server uses IBM's QBIC (query by image content) technology to help you search on objects by certain visual properties, such as color and texture. The image search server analyzes images and stores the image

information in a database. Then users can run image queries, which use the visual properties of images, to match colors, textures, and their positions without describing them in words. You can combine content-based queries with text and keyword searches to get powerful retrieval methods for image and multimedia databases.

Each image search server has a data directory that contains one or more image search databases, which hold the image search catalogs. An image search catalog stores the data about the visual features of a collection of images. The actual image objects are stored on object servers in the Digital Library system. The image search server runs on AIX and Windows NT.

What's new in Version 2

Digital Library Version 2 includes many new features to help you get the most out of your system.

Multi-search capability
Using Java, C++, or ActiveX APIs, you can write and run combined queries. In a single query, you can perform searches that combine parametric, text, and image search. This multi-search capability is demonstrated in a sample application that comes with Digital Library.

Rights management
Protect intellectual property with both visible watermarking and invisible marking. A sample application demonstrates this technology. The sample application can also be used to produce your own marking.

Java-based system administration
The Java-based system administration program is a graphical interface that you use to configure and manage your Digital Library system. You use the system administration program for such tasks as authorizing user access to the system, managing work in progress, defining policies for storage management, and customizing the way objects are processed.

Media manager administration, text search administration, image search administration and the database utilities are now integrated with the Digital Library system administration program to unify administration tasks.

Internet toolkit
The Internet toolkit consists of the Internet application toolkit and the dynamic page builder. The Internet application toolkit is a set of Java applets or a Java application that enables users to query, view, create, update, and delete objects stored in Digital Library. The dynamic page builder lets you create applications that dynamically format the results of a query and display them on a Web page. An included sample application demonstrates the dynamic page builder APIs.

ActiveX APIs
These APIs allow you to create Digital Library applications in ActiveX enabled languages, such as Visual Basic and Lotus Script.

TaskGuide for AIX install
The AIX installation program uses IBM TaskGuide technology, which simplifies installation and setup tasks. The installation TaskGuide is a graphical user interface for the installation program and AIX installation utilities. Start the TaskGuide and fill in the necessary information to install Digital Library.

Media manager
The media manager is an AIX or Windows NT-based component of VideoCharger. Digital Library and VideoCharger provide capabilities for storing and playing audio and video files. Clients that work with the media manager can run on AIX, Windows NT, or Windows 95.

New information delivery media
The online help is delivered in HTML to easily view from an included Web browser. All the books are available in both PDF and PostScript formats on one CD-ROM. The Adobe Acrobat Reader is included on the CD-ROM.

Performance
Digital Library server performance and dynamic page builder performance have been improved since Version 1.

More client and server operating systems
You can now install servers on Windows NT operating systems. Clients can run in the Macintosh environment.

What's new in Version 2.4

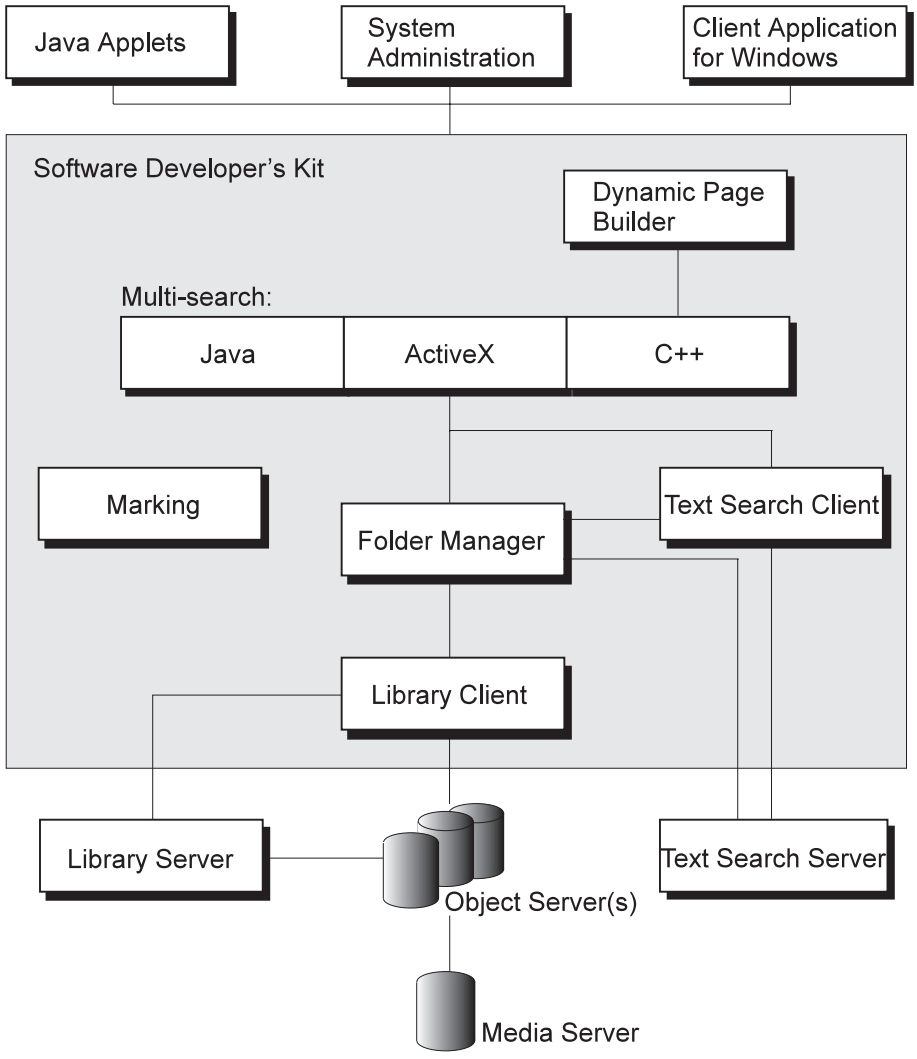
Digital Library Version 2.4 provides the following enhancements to Version 2 features:

- AIX multi-threaded toolkit allows an application to concurrently log on to the same or different library servers multiple times, performing the same or different functions.
- Performance improvements reduce the time it takes to store and retrieve large objects (for example, 200 MB) between Windows NT and AIX object servers and clients.
- Windows NT and AIX toolkits provide the ability to perform multiple retrieves and stores resulting in better performance for desktop and Web applications.
- Additional toolkit enhancements (such as new folder manager APIs) result in better performance and increased function of user applications. The Dynamic Page Builder APIs, which enable you to create Digital Library World Wide Web applications, now allow update of metadata, and creation, modification and deletion of objects. New implementations of the object-oriented interface to Digital Library enable the WorkFlow/WorkBasket, VideoCharger, and new Text Search Engine functions.
- Improvements to the Digital Library installation process make it easier to upgrade your current Digital Library system.

Chapter 2. Digital Library application programming concepts

Digital Library offers three object-oriented (OO) application programming interfaces you can use to create query applications (APIs) that access and display relational data, including multimedia data. This chapter provides a brief overview of how these APIs fit into the Digital Library architecture, and describes the object-oriented programming concepts that the APIs are based on.

The following diagram shows how the C++, Java, and ActiveX APIs access the Digital Library object and library servers, through the folder manager and library client, for parametric searches. Additionally, the APIs can access the Text Search Engine client and server text searches. The APIs let you create applications that can combine the two kinds of searches in one query.



Sample programs and applications are provided to demonstrate how the APIs work:

- The SAMPLES directory contains sample programs for the C++, Java, and ActiveX APIs. These samples are described throughout this book.
- See the *Planning and Installation Guide*, and "Starting the Dynamic Page Builder sample" on page 217, for more information about using the samples.

Understanding Digital Library data concepts

Digital Library provides a set of data objects you can use to organize and store your data. This section describes the concepts associated with items, index classes and attributes, item parts, documents, and folders.

Items

Items are the building blocks in Digital Library. Each item in Digital Library is uniquely identified by an item ID that is persistent and stored in Digital Library library server. Each item has an ID, which is the handle used to access its associated data, such as the value of an item attribute.

An item can represent a multimedia document or a folder.

Item attributes and index classes

An item is associated with a set of attributes (also called key fields) and an index class that is created using these attributes. An item can be said to be ″indexed by″ the index class). An index class is a group of items that share a set of attributes.

Each attribute of an item has a value whose data type can be varied. See the System Administration Guide for more information about the data types of attributes.

Item parts

An item can be associated with, or contain, one or more parts. Each item part corresponds to one multimedia data, such as in image, or text.

Documents and folders

A document is an item and has a unique item ID. A document in Digital Library can be any multimedia item that has digital content.

A document can be composed of parts. For example, a paper document consists of a set of closely related sub-documents, such as pages in a report.

In Digital Library, a document can contain textual information, images, video clips, or a combination of these. For example, a movie document, which can be considered a document that represents movie images, can also contain textual information, a set of still images, and an image of a movie poster.

A folder is a container that holds one or more items, such as documents or other folders. A folder is an item.

Understanding the dynamic data object concepts

In compliance with Object Management Groups's CORBA Persistent Object Service and Object Query Service Specification, Digital Library provides an implementation of the Dynamic Data Object (DDO) and its extension, the Extended Data Object (XDO), which are part of the CORBA Persistent Data Service (PDS) Protocols. The concepts of DDO and XDO are not specific to any one datastore, and can be used

to represent data objects in any database management system supported Digital Library. (A datastore is a general term for data storage facilities, such as IBM DB2 and Oracle.)

The Dynamic Data Object is an interface to move data in and out of a datastore. DDOs exist in the application space and vanish when an application terminates.

In addition to the DDO and XDO, other concepts, such the PID, datastores, collections, queryable collections, iterators, and the Query Manager are described in this section.

Dynamic data objects (DDO)

The DDO is a datastore-neutral representation of an object's persistent data. Its purpose is to contain all of the data for a single persistent object. It's also an interface to retrieve persistent data from, or load persistent data into, a datastore.

A DDO has a single persistent ID (PID), an object type, and a set of data items whose cardinality is called the ″data count″. Each data item can have a name, a value, an ID, one or more data properties, and data property count. Each data property can have an ID, a name, and a value.

For example, a DDO can represent a row of a database table whose columns are represented by DDO's data items and their properties.

A DDO can contain one or more Extended Data Objects (XDOs) that represent non-traditional data types.

Extended data objects (XDO)

An XDO is a representation of complex multimedia data. Examples are 'parts' in Digital Library or a new data-type introduced by an relational database's object-relational facilities, such as IBM DB2 Extenders.

XDOs complement DDOs by storing multimedia data of complex types and offering functions that implement the data type's behaviors in the application space. XDOs can be contained in or 'owned' by a DDO to represent a complex multimedia data object.

XDOs have a set of properties to represent such information as data types and IDs.

XDOs can also be stand-alone dynamic objects.

Representing multimedia content

DDOs and XDOs can represent any multimedia data objects.

For example, a movie can be represented by a DDO. This DDO contains multiple data items, which represent attributes of the movie such as `Director_Name` or `Movie_Title`, and multimedia XDOs, which represent the movie's multimedia data such as video clips or still images.

Comparing DDO/XDOs with attribute values and item-parts

A DDO corresponds to an item in Digital Library. The DDO's object type corresponds to the item's associated index class. The data items of a DDO correspond to an item's attributes. In Digital Library, an index class is created using a set of attributes, and an item is always indexed by an index class.

A DDO can hold one or more XDOs that correspond to 'item parts' in Digital Library.

Persistent object identifiers (PID)

The persistent object identifier (PID) uniquely identifies a persistent object in any datastore. A DDO's PID consists of a Digital Library item ID, a datastore name, and other related information. When a DDO is added to a datastore, a PID needs to be created for it. For example, a PID must be assigned to a DDO that is created by getting persistent data out of the datastore with a query result list.

Because a DDO is a dynamic interface to persistent data that is moved in or out of datastores, different DDOs can represent the same persistent data entities, and therefore the DDOs can have the same PID. For example, a DDO can be created to move a data entity into a datastore to store data persistently, and another DDO can be created to hold the same data entity that is checked out from the same datastore for modification. In this case, these two DDOs share the same PID value.

Understanding data access through datastores

A datastore is a data repository that is compatible with the DDO/XDO protocol. A Digital Library datastore supports user sessions, connections, transactions, cursors and query executions. Digital Library applications using the class library described in this book can add, retrieve, update and delete DDOs using the datastore's functions.

Datastores in a Digital Library session

After connecting successfully to a Digital Library library server, a Digital Library datastore maintains a user session with the library server.

Datastores and DDOs

DDOs are created and dynamically associated with a datastore. The association between a DDO and a datastore is established with the DDOs PID.

In general, a Digital Library application goes through the following steps to move data in and out of a data repository:

1. Create a datastore.
2. Establish a connection to data repository.
3. Create the DDOs to be operated on, and associate the datastore with the DDOs.
4. Add, retrieve, update, and delete the DDOs using appropriate methods.
5. Close the connection and destroy the datastore.

Comparing schema with index classes and attributes

Datastores provide a set of methods to list mapping schema information, such as listSchema() and listSchemaAttributes().

For a Digital Library datastore, listSchema() returns a list of index classes that are defined in Digital Library library server, and listSchemaAttributes() returns a list of attribute information for a given index class.

Chapter 3. Using the C++ application programming interface

The C++ application programming interface (API) is a set of classes that provides access and manipulation of local or remote data stored in the Digital Library storage facilities.

This document describes the C++ Application Programming Interface, the C++ implementation of Digital Library API multi-search facilities and Internet connectivity. It is derived from the C++ Class Library API, in turn derived from OMG/Object Query Services (OQS) and Dynamic Data Object (DDO) protocol, which is a part of OMG/Persistence Object Services.

The C++ API supports:

- Multi-search and update across a combination of heterogeneous datastores
- A common-object model for Digital Library data access
- A flexible mechanism to use a combination of different search-engines; for example, Text Search Engine text search and Query by Image Content (QBIC) image search

The C++ API and Java API chapters contain similar information placed within a consistent structure; this consistency between chapters unites the information contained in the C++ and Java API sets. The consistent structure also helps application developers as they coordinate the development of their applications, regardless of the programming language they choose.

General handling and tracing information

This section includes the information about handling large objects and exceptions; in the previous edition of this book, the Text Search Engine section of this chapter contained the handling information.

Handling large objects

This section describes the ways in which Digital Library handles large objects.

What is MAXPIECE?

`MAXPIECE` is an environment variable that indicates the maximum object size processed in Digital Library. This variable defines the largest object in megabytes that is processed as a whole.

When `MAXPIECE` is set during the input and output operation to the Digital Library datastore, an object that is larger than the `MAXPIECE` value will be broken down into chunks, sized equal to or less than the `MAXPIECE` value. If `MAXPIECE` is not set, the large object will be treated as a single entity.

How to set the MAXPIECE environment variable

In Windows NT, go to **Start** → **Setting** → **Control Panel** → **System** → **Environment** → **System Variables** or **User Variables**, enter `MAXPIECE` in the entry field and a number (for example, `4`) in the value field, then click **Set** → **OK**. This will set `MAXPIECE` to `4` megabytes.

In AIX, type: `export MAXPIECE=4` , then press the Enter key. This will set `MAXPIECE` to `4` megabytes.

Exception handling

A `DKException`, once caught, allows you to see any error messages, error codes and error states that occured while running. If an error is caught below the `DKException` name and error text will be printed, as well as the location of where the exception was thrown. The error id and exception id is also printed.

```
try {
    DKDatastoreTS dsTS;
    dsTS.connect("TM","","","");
    dsTS.disconnect();
    }
    catch(DKException &exc)  {
       cout << "Error id "     << exc.errorId() << endl;
       cout << "Exception id " << exc.exceptionId() << endl;
       for(unsigned long i = 0; i< exc.textCount();i++) {
          cout << "Error text: " << exc.text(i) << endl;
       }
       for (unsigned long g=0; g< exc.locationCount();g++) {
          const DKExceptionLocation* p = exc.locationAtIndex(g);
          cout << "Filename: " << p->fileName() << endl;
          cout << "Function: " << p->functionName() << endl;
          cout << "LineNumber: " << p->lineNumber() << endl;
       }
       cout << "Exception Class Name: " << exc.name() << endl;
}
```

Tracing Information

You can use any of the following environment variable settings to get tracing information.

For Text Queries Using Text Search Engine

- `DKTMDSTREAMTRACE=<fileName>` (for example, `<.\tm.out>` for Windows NT or `<./tm.out>` for AIX)

  This environment variable setting puts the Text Search Engine query datastream, in binary format, into the specified file.
- `DKTMTRACE=<fileName>` (for example, `<.\tm.api>` for Windows NT or `<./tm.api>` for AIX)

  This environment variable setting puts some of the Text Search Engine API calls used during a text query into the specified file.

For Parametric Queries Against Digital Library

- `DKDLQRYTRACE=<fileName>` (for example, `<.\dlqry.out>` for Windows NT or `<./dlqry.out>` for AIX)

  This environment variable setting puts the parametric query passed to the Folder Manager into the specified file.

Multi-Search facilities

Use the multi-search facilities to:

- Search within a given datastore, using one or a combination of supported query types:

Parametric query
Queries requiring an exact match on the condition specified in the query predicate and the data values stored in the datastore.

Text query
Queries on the content of text fields for approximate match with the given text search expression; for example, the existence (or non existence) of certain phrases or word-stems.

Each search type is supported by one or more search-engines.

- Search on the results of previous search.

Understanding a Digital Library datastore

A DKDatastoreDL represents and manages a connection to a Digital Library datastore (or datastore DL); it also provides transaction support and executes datastore commands.

Establishing a connection

The Digital Library datastore provides a method for connect as well as a method for disconnect. The following is an example of connecting to the Digital Library server name `LIBSRVRN`, using the user ID `USER1` and password `PASSWORD`. Normally, you would create a datastore DL, connect to it, do some work, then disconnect when done.

A complete sample, `TConnectDL.cpp`, is available in the samples directory.

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","USER1","PASSWORD");
...                       // do some work
dsDL.disconnect();
```

Setting and getting datastore options

The datastore provides some processing or informational options that you can set or get using its methods. The following example shows how to set and get the option for establishing an administrative session. See the *Application Programming Reference* for the list of options and their descriptions.

```
DKAny input_option = DK_SS_CONFIG;
DKAny output_option;
dsDL.setOption(DK_OPT_DL_ACCESS,input_option);
dsDL.getOption(DK_OPT_DL_ACCESS,output_option);
```

The option value usually takes an Integer value, but it could be of any object.

List servers

The datastore provides a method to list the library servers it can connect to. The following example shows how to retrieve the list of servers. The list of servers is returned in a sequential collection of `DKAny` containing a `DKServerInfoDL` object. Once a `DKServerInfoDL` object is obtained, you can pull out the server name and server type, and use the server name to establish a connection to it.

A complete sample, `TListCatalogDL.cpp`, is available in the samples directory.

```
DKDatastoreDL dsDL;
DKServerInfoDL *pSV = 0;
DKString strServerName;
DKString strServerType;

DKAny a = dsDL.listServers();
DKSequentialCollection* pCol = (DKSequentialCollection*)((dkCollection*)a);
dkIterator* pIter = pCol->createIterator();
while (pIter->more()) {
  pSV = (DKServerInfoDL*)((void*)(*pIter->next()));
  strServerName = pSV->serverName();
  strServerType = pSV->serverType();
  cout << "Server Name : " << strServerName << endl;
  cout << "Server Type : " << strServerType << endl;
  delete pSV;
}
delete pIter;
delete pCol;
```

List schema and schema attributes

The datastore provides methods for listing the schema in the Digital Library datastore; these are index-classes and their attributes. The following example shows how to retrieve the list of index classes, as well as the list of attributes. The list of index classes is returned in a DKSequentialCollection of DKStrings. The attributes are returned as a DKSequentialCollection of DKAttributeDef objects. Once a DKAttributeDef object is obtained, you can pull out information about the attribute (such as its name and type), and use the information to form a query.

For further details, see the *Application Programming Reference* on DKDatastoreDL for these two methods.

A complete sample, `TListCatalogDL.cpp`, is available in the samples directory.

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","USER1","PASSWORD");
...
DKSequentialCollection *pCol2 = 0;
dkIterator *pIter2 = 0;
DKString strIndexClass;
DKAttributeDef *pDef = 0;
DKAny a;
DKSequentialCollection* pCol = (DKSequentialCollection*)
    ((dkCollection*)dsDL.listSchema());
dkIterator* pIter = pCol->createIterator();
while (pIter->more()) {
  a = (*pIter->next());
  strIndexClass = a;
  pCol2 = (DKSequentialCollection*)((dkCollection*)
  dsDL.listSchemaAttributes(strIndexClass));
  pIter2 = pCol2->createIterator();
  while (pIter2->more()) {
      pDef = (DKAttributeDef*)((void*) (*pIter2->next()));
      cout << " name  = " << pDef->name << endl;
      cout << " type  = " << pDef->type << endl;
          delete pDef;   // see programming tips below
  }
  delete pIter2;
  // see programming tips below
  delete pCol2;
}
delete pIter;
delete pCol;
```

Note that these two methods require a valid connection to the datastore.

Programming tips

Instead of deleting `pDef` immediately, you could defer it at a later time and use the apply function:

```
...
pCol2->apply(deleteDKAttributeDef);
delete pCol2;
...
```

This will delete the whole attribute definition inside the collection. See “Memory management in collections” on page 23, for more information.

Understanding datastore text search (Text Search Engine)

A DKDatastoreTS represents the Text Search Engine. Text Search Engine does not actually store the data; it merely indexes the data stored in the Digital Library datastore to support text searches on that data. The result of a text search is a document identifier (`item-id`) describing the location of the document in the Digital Library datastore. These identifiers can be used to retrieve the document from the Digital Library datastore.

The datastore for text search does not support add, update, retrieve and delete operations. You can perform queries against this datastore.

To learn how to add data to Digital Library that is indexed by Text Search Engine, see “Loading data to be indexed by Text Search Engine” on page 71, for more information.

Establishing a connection

The datastore for Text Search Engine provides two methods for connect as well as a method for disconnect. The following is an example of the first connect method to the text search server TM. Normally, you would create a datastore text search (or datastore TS), connect to it, do some work, then disconnect when done.

A complete sample, `TConnectTS.cpp`, is available in the samples directory.

```
DKDatastoreTS dsTS;
dsTS.connect("TM","","","");
...                    // do some work
dsTS.disconnect();
```

The following example shows the second connect method using a TS server hostname `apollo`, port number `7502`, and TCP/IP communication type:

```
dsTS.connect("apollo","7502",DK_CTYP_TCPIP);
```

The following example shows the first connect method using text search server hostname `apollo`, port number `7502`, communication type T (TCPIP):

```
dsTS.connect("apollo","","", "PORT=7502;COMMTYPE=T");
```

The following example shows the second connect method using the text search server name TM:

```
dsTS.connect("TM","",'');
```

The next example shows the first connect method using the text search server name TM and using Digital Library server LIBSRVR2, user ID USER1 and password PASSWORD:

```
dsTS.connect("TM","","", "LIBACCESS=(LIBSRVR2,USER1,PASSWORD)");
```

Note that the last parameter, also called connect-string, can be used to pass a sequence of parameters in one string.

Getting and setting datastore TS options

The datastore TS provides some options that you can set or get using its methods. The following example shows how to set and get the option for a TS character code set. For further information, see the *Application Programming Reference* for the list of options and their descriptions.

```
DKAny input_option = DK_CCSID_00850;
DKAny output_option;
dsTS.setOption(DK_OPT_TS_CCSID,input_option);
dsTS.getOption(DK_OPT_TS_CCSID,output_option);
```

The option value usually takes a long integer value, but it could be of any type.

List servers

The datastore TS provides a method to list the Text Search Engine servers it can connect to. The following example shows how to retrieve the list of servers. The list of servers is returned in a sequential collection of DKAny, containing DKServerInfoTS objects. Once a DKServerInfoTS object is obtained, you can pull out the server name and server location, and use the server name to establish a connection to it.

A complete sample, TListCatalogTS.cpp, is available in the samples directory.

```
DKString strServerName;
char chServerLocation = ' ';
DKServerInfoTS *pSV = 0;
DKAny a = dsTS.listServers();
DKSequentialCollection* pCol = (DKSequentialCollection*)((dkCollection*)a);
dkIterator* pIter = pCol->createIterator();
while (pIter->more()) {
  pSV = (DKServerInfoTS*)((void*)
  (*pIter->next()));
  strServerName    = pSV->serverName();
  chServerLocation = pSV->serverLocation();
  cout << "Server Name : " << strServerName << endl;
  cout << "   Location : " << chServerLocation << endl;
  delete pSV;
}
delete pIter;
delete pCol;
```

List schema

The datastore provides methods for listing the schema; in the Text Search Engine case, these are text search-indexes. The following example shows how to retrieve the list of search-indexes. The list of returned indexes is in a sequential collection of DKAny containing DKIndexTS objects. Once a DKIndexTS object is obtained, you can pull out information about the index (such as its name and library id) and use the information to form a query.

A complete sample, TListCatalogTS.cpp, is available in the samples directory.

```
DKString strIndexName;
DKString strLibId;
DKIndexTS *pIndx = 0;
DKSequentialCollection* pCol = (DKSequentialCollection*)
((dkCollection*) dsTS.listSchema());
dkIterator* pIter = pCol->createIterator();
while (pIter->more()) {
   pIndx = (DKIndexTS*)((void*)
   (*pIter->next()));
   strIndexName = pIndx->indexName();
   strLibId = pIndx->libraryId();
   cout << " Index Name : " << strIndexName << endl;
   cout << " Library Id : " << strLibId     << endl;
   delete pIndx;
}
delete pIter;
delete pCol;
```

See “Memory management in collections” on page 23 for programming tips on how to delete the collection.

Using DKAny

DKAny contains an object which type varies at run-time. It can hold types null, (unsigned) short, (unsigned) long, double, DKBoolean, char, DKString, DKDate, DKTime, DKTimestamp, and TypeCode. In addition to the above types, DKAny can also hold object reference types, that is dkDataObjectBase*, dkCollection*, and void*.

Type code

The current type can be obtained by calling typeCode() method, which will return a TypeCode object, that is, tc_null for null, tc_short for short, etcetera. Please refer to *Application Programming Reference* to get the complete listing of type-codes.

Memory management in DKAny

DKAny manages memory for the object it contains, EXCEPT object reference types. Copy-related operations involving object reference will do shallow copy, that is, copy the pointer only. You need to keep track of object reference types during copying and deletion.

Using constructors

DKAny provides a constructor for each type it supports. The following is an example showing how to create DKAny containing some the above types:

```
DKAny any1((unsigned short) 10);            // contains unsigned short 10
DKAny any2((long) 200);                     // contains long 200
DKAny any3(DKString("any string"));         // contains DKString
DKAny any4(DKTime(10,20,30));               // contains DKTime
DKAny any5((dkDataObjectBase*) new DKDDO);  // contains DKDDO
DKAny any6(new MyObject(5,"abc"));          // contains MyObject
DKAny any7(new DKDDO);                      // shorter form of any5
```

Getting the type_code

Use the typeCode() method to find the type-code of the object inside DKAny:

```
DKAny::TypeCode type_code;
type_code = any1.typeCode();              // type_code is tc_ushort
type_code = any4.typeCode();              // type_code is tc_time
type_code = any5.typeCode();              // type_code is tc_dobase (object ref)
type_code = any6.typeCode();              // type_code is tc_voidptr since
                                          // MyObject is not recognized by DKAny
```

Assignment to DKAny: DKAny = <TYPE>

To assign a new value to an existing DKAny, you use the assignment operator =. DKAny provides an assignment for each type-code.

```
DKAny any;                                // any contains null
long vlong = 300;
DKTimestamp vts(1997,8,28,10,11,12,999);
dkDataObjectBase* dobase =
(dkDataObjectBase*) new DKDDO;
any = vlong;                              // any contains long 300
any = vts;                                // any contains timestamp
any = dobase;                             // any contains ddo
any = new DKDDO;                          // any contains ddo
```

Assignment from DKAny: TYPE = <DKAny>

Assigning a DKAny back to a regular type requires a cast operator. For example:

```
vlong      = (long) any2;                         // sets vlong to 200
DKTime at  = (DKTime) any4;                       // sets at to (10,20,30)
DKDDO* ddo = (DKDDO*) ((dkDataObjectBase*) any5); // extract the ddo
dkDataObjectBase* dobase = any7;                  // extract the DDO
```

You will get an exception ″Invalid type conversion″ if the type does not match. Therefore, you must check the type-code first before converting DKAny to a regular type:

```
if (any5.typeCode() == DKAny::tc_dobase)
    dobase = (dkDataObjectBase*) any5;
```

In practice, it is common to have a case statement to check the type of DKAny, as follows:

```
       switch(any.typeCode()) {
          case DKAny::tc_short:
               // operation for short
               ...
               break;
          case DKAny::tc_ushort:
               // operation for unsigned short
               ...
               break;
          ...  etc.
       }
```

For object reference, you could get DKAny content as a void pointer, then cast it to the proper type. However, this operation should only be done if you know exactly the type-code inside DKAny:

```
// knows exactly any5 contains DKDDO
ddo = (DKDDO*) any5.value();
```

Displaying DKAny

You can use cout to display the content of DKAny:

```
cout << any3 << endl;       // displays "any string"
cout << any4 << endl;       // displays "10:20:30"
cout << any5 << endl;       // displays "(dkDataObjectBase*) <address>",
                            // where address is the memory location of the ddo
```

Destroying DKAny

Since DKAny can hold an object reference but does not manage memory for object-reference types, you need to manage memory for these types. The following is an example of such a situation:

```
DKDDO* ddo = new DKDDO;                         // creates a DKDDO in the heap
DKAny anyA((dkDataObjectBase*)ddo);
DKAny* anyB = new DKAny(anyA);                  // creates anyB in the heap
                                                // anyA and anyB contains a
                                                // reference to the same ddo
...
delete anyB;                                    // delete anyB, does not delete ddo
if (anyA.typeCode() == DKAny::tc_dobase)
    delete ((dkDataObjectBase*) anyA.value()); // deletes the ddo
```

The last delete statement must be done before exiting the current scope, otherwise anyA will be deleted, leaving the DDO as a memory leak. For types other than object-reference, you do not need to perform the above steps, since DKAny does it automatically.

Programming tips

When converting an integer literal to DKAny, it is advisable to state the type explicitly to avoid undesirable type conversion.

```
any = 10;                       // ambiguous
any = (unsigned long) 10;       // unambiguous
any = (short) 4;                // unambiguous
```

Collections and iterators

dkCollection is an abstract class which provides the interface to collection functions. DKSequentialCollection provides the concrete implementation of those functions. Other collections are derived as a subclass of DKSequentialCollection. In essence, these collections contain DKAny objects as members. When a new member is added, the collection owns it. When the member is retrieved back, you get a pointer to DKAny object inside the collection. This object belongs to the collection, meaning that the collection manages the memory for its DKAny members. Since DKAny can hold an object reference but does not manage memory for object-reference types, you need to manage these. Usually, collection members have the same type-code, however, it is not prohibited to have members with different type-codes in one collection.

Sequential collection

DKSequentialCollection provides methods for adding, retrieving, removing, and replacing its members. In addition, it has the apply() and sort() functions. The following example illustrates how to add a new member to a collection:

```
DKSequentialCollection sq;
DKAny any = DKString(" first member ");
sq.addElement(any);         // add a new element at last position
```

```
                                        // any will be copied into the collection
                                        // you own the original any, the collection
                                        // owns the copy
```

Sequential iterator

Iterators are provided to let you iterate over collection members. There are two types of iterators, base iterator `dkIterator` (which supports `next()`, `more()`, and `reset()` operations), and its subclass `DKSequentialIterator`, which has more methods. An iterator is created by calling the method `createIterator()` on the collection. This method will create a new iterator and return it to you. You need to delete it when it is no longer needed. Typically, you would write the following code to iterate over a collection:

```
dkIterator* iter = sq.createIterator();   // create an iterator for sq
DKAny* member;
                                          // while there are more members
                                          // get the current member and
                                          // advance iter to the next member
while(iter->more()) {
    member = iter->next();

    cout << *member << endl;              // display it, if you want to
    ...                                 // do other processing
    }
delete iter;                              // do not forget to delete iter
```

`DKSequentialIterator` provides additional methods to move the iterator in a bidirectional manner. The above code could be rewritten as follows:

```
DKSequentialIterator* iter =                  // create an iterator for sq
(DKSequentialIterator*) sq.createIterator();
DKAny* member;
      while(iter->more()) {
         member = iter->at();                 // get the current member
         ...                                // do other processing
         iter->setToNext();                   // advance to the next position
      }
      delete iter;
```

These code fragments allow you to do some operations at the current member before moving along to the next member. Such an operation would be, for example, replacing a member with a new one, or removing it.

```
any = DKString("the new first member");

sq.replaceElementAt(any, *iter);        // replace current member with a new one
...                                   // or
sq.removeElementAt();                   // remove the current member
...
```

Note that when you remove the current member, the iterator will be advanced to the next member. Therefore, when removing members inside a loop construct, do not forget to take this fact into account, that is, you need to do some checking, like the following:

```
...
if (removeCondition == TRUE)
    sq.removeElementAt(*iter);   // remove current member, do not advance iter
                                 // since it is advanced to the next after
                                 // the removal operation
else
    iter->setToNext();           // no removal, advance the iterator
...                         // to the next position
```

The above check is necessary, to avoid skipping the next member after removing the current one.

Memory management in collections

As mentioned before, the collection manages the memory for its members, which are DKAny objects. The same rules governing DKAny objects applies here also, that is, if the object inside DKAny is of type object-reference, then you are responsible for managing the storage. This step needs to be performed, as appropriate, when you are:

- destroying the collection
- replacing a member
- removing a member

The example below illustrates these situations:

```
// retrieve the member and hang-on to it
member = iter->at();

// code to handle this member as to prevent memory leaks
if (member->typeCode() == DKAny::tc_dobase) {
    // delete it if no longer needed
    delete ((dkDataObjectBase*) member->value());
 }

sq.removeElementAt(*iter);           // remove it from the collection
```

Instead of deleting the member, you could add it into another collection, as required by your application. Similar steps should be done before using replaceElementAt() and removeAllElement(). Before destroying a collection, you need to delete its members one by one as appropriate. You can write a function to perform this task and pass this function to the apply() method of the collection. Suppose you have a collection of DKAny containing DKAttributeDef objects, as a return result from the call to the listSchemaAttributes() method in the DKDatastoreDL class. To delete this collection, you need to do the following steps:

```
DKDatastoreDL dsDL;
...
DKAny any = dsDL.listSchemaAttributes("GRANDPA");
dkCollection* acoll = (dkCollection*) any;
...                                          // use the attributes
acoll->apply(deleteDKAttributeDef);                // deletes all members
delete acoll;
```

deleteDKAttributeDef is a function that takes DKAny as a parameter. It is defined in DKCollectionFunctions.hpp as follows:

```
void deleteDKAttributeDef(DKAny& any) {
    delete ((DKAttributeDef*) any.value());
    any.setNull();                           // good practice
}
```

You could write your own delete function to delete your collection.

Optionally, you could remove some members you want to keep around before deleting the collection.

Again, if the collection members are not of type object-reference, you do not need to do the above steps. DKAny will perform the cleanup for you.

Some known collections, like DKParts, DKFolder, and DKResults perform these necessary clean-up steps in their destructor. However, they do not manage storage during replaceElementAt(), removeElementAt(), and removeAllElement(); you still need to manage storage in these cases.

Sorting the collection

The sort() method allows you to sort collection members based on a specified key in either ascending or descending order. You need to pass in a sort function object and the desired order. The interface for sort function objects is defined in dkSort.hpp, so you can write your own sort function for sorting your specific collection. The following example illustrates how to sort a collection of DDOs based on each DDO's item-id:

```
DKResults* rs;
...                              // execute a query to fill DKResults with DDOs
...

DKSortDDOId sortId;              // the sort function; sort on item-id
rs->sort(&sortId);              // by default, sort in ascending order
...
```

The definition of function object DKSortString is provided in the sample directory as an example to create a new sort function object.

Programming tips

The above sort function object is created in the stack, so it does not have to be deleted explicitly. The function is re-entrant; a single copy can be shared, reused, and passed around.

Using DDO (Dynamic Data Object)

DKDDO can be regarded as a container of attributes. An attribute (also called data-item; these terms will be used interchangeably) has properties, a name, and value.

Each attribute is identified by a data_id, which is a number started from 1 up to the total number of attributes in the DDO. Since the number, name, value, and property of an attribute can vary, DKDDO provides flexible mechanisms to represent data originated from a variety of datastores and format, whether these are items from different index classes in Digital Library, or rows from different tables in a relational database. The DKDDO itself can have properties which apply to the whole DKDDO, instead of only to one particular attribute.

You need to associate a DKDDO with a datastore to be able to call the methods add, retrieve, update, delete, to send its attributes into the datastore and retrieve them back. This is done by calling the proper DKDDO constructor, or by calling the setDatastore() method.

Every DKDDO has a persistent object identifier (PID), which contains information to locate the data-items in the datastore. Recall that in Digital Library, a DDO represents an item which can be a document or a folder.

Creating a DKDDO

The simplest way to create a DKDDO is by calling its constructor, which takes no parameter:

```
DKDDO addo;
```

If you have some ideas on how many attributes the DDO will have, you can pass this information to the constructor:

```
DKDDO bddo(10);
```

The DKDDO will be constructed, leaving enough room to hold 10 attributes. This is more efficient than the previous statement, since `bddo` does not have to grow on the fly to accommodate more attributes.

You can create a DKDDO by supplying datastore and object type as input:

```
// create a digital library datastore
DKDatastoreDL dsDL;
// create a DDO to hold an object type GRANDPA in dsDL
DKDDO* cddo = new DKDDO(&dsDL, "GRANDPA");
```

Pid

All DDO must have a persistent object identifier, or Pid. This contains information about the datastore name, datastore type, id, and object type. The id identifies the location of the DDO's persistent data in the datastore. For the Digital Library datastore, this id is the item-id. The item-id is one of the most important pieces of information for the retrieve, update, and delete operations. For add, the item-id will be created and returned by the datastore.

To create a DDO of a known item for retrieval, you could do the following:

```
DKDatastoreDL dsDL;                  // create a digital library datastore
DKPid pid;
pid.setObjectType("GRANDPA");        // set the index-class name it belongs to
pid.setId("LN#U5K6ARLGM3DB4");       // set the item-id
DKDDO* ddo = new DKDDO(&dsDL, pid);  // create a DDO with pid and associated to dsDL
```

You can then connect to the datastore and call retrieve to retrieve this DDO. This topic is discussed further in "Retrieving a document or folder" on page 59.

Adding data-items and properties

Suppose the index-class `GRANDPA` has the following two attributes:

Attribute:	**data_ID=1**	**2**
Name:	Title	Subject
Type:	String	String
Can be null?	no	yes

You can represent the above information in a DKDDO as follows:

```
// create a digital library datastore
DKDatastoreDL dsDL;
// create a DDO to hold an object type GRANDPA in dsDL
DKDDO* cddo = new DKDDO(&dsDL, "GRANDPA");
DKAny any;
DKBoolean yes = TRUE;
```

```
DKBoolean no = FALSE;
// add the first attribute named "Title"
unsigned short data_id = cddo->addData("Title");

// add a property named: "type", set to value : variable length string
any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_PROPERTY_TYPE, any);

// add a property named: "nullable", set to value : boolean false
any = no;
cddo->addDataProperty(data_id, DK_PROPERTY_NULLABLE, any);

// add the second attribute named "Subject"
data_id = cddo->addData("Subject");

// add a property named: "type", set to value : variable length string
any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_PROPERTY_TYPE, any);

// add a property named: "nullable", set to value : boolean true
any = yes;
cddo->addDataProperty(data_id, DK_PROPERTY_NULLABLE, any);
```

The above example illustrates the standard properties that an attribute should have, namely `type` and `nullable`. In practice, you can have as many additional properties as required by your application.

Adding properties to a DDO

So far, the above DKDDO has all the attributes information it needs to have. However, there is no information to indicate the kind of DKDDO it is; it must be either a document or a folder. This information is recorded under DKDDO properties, as opposed to attribute properties.

```
any = DK_DOCUMENT;                          // it is a document
cddo->addProperty(DK_PROPERTY_ITEM_TYPE, any);
```

Setting and getting data_item values

Once you created a data_item and its properties, you can set its value:

```
// set Title value to the given string
// assume we know the data_id for the data_item "Title" is 1
any = DKString("One dark and stormy night");
cddo->setData(1, any);
// set Subject value to the given string
// assume we do not know the data_id for the data_item "Subject"
// find data_id for data_item named "Subject"
data_id = cddo->dataId("Subject");
any = DKString("Mystery");
cddo->setData(data_id, any);
```

To get the value back, you use the `getData()` method:

```
any = cddo->getData(1);
// displays "One dark and stormy night"
cout << "Title   = " << any << endl;
// displays "Mystery"
cout << "Subject = " << cddo->getData(data_id) << endl;
```

Getting the properties

When processing a DKDDO, the first thing you want to know is probably its type, a document or a folder. The following code illustrates such a check:

```
unsigned short prop_id =
    cddo->propertyId(DK_PROPERTY_ITEM_TYPE);
if (prop_id > 0) {
    unsigned short type = (unsigned short) cddo->getProperty(prop_id);
    switch(type) {
        case DK_DOCUMENT:
        // process document
        ...
        break;
        case DK_FOLDER:
        // process folder
        ...
        break;
    }
}
```

To retrieve properties of an attribute, you need to have the data_id of the attribute:

```
// get data_id of Title
data_id = cddo->dataId("Title");
// how many props does it have?
unsigned short number_of_data_prop = cddo->dataPropertyCount(data_id);
// displays all data properties belonging to this attribute
// notice that the loop index starts from 1, where
// 1 <= i <= number_of_data_prop
for(unsigned short i = 1; i <= number_of_data_prop; i++) {
    cout << i << "  Property Name = " << cddo->
    getDataPropertyName(data_id, i)  <<  "  value = "   << cddo->
    getDataProperty(data_id, i) << endl;
}
```

It is important to notice that data_id and property_id start from 1; if you specify zero, you will get an exception.

Displaying the whole DDO

During the course of application development, a developer might have need to display the content of a DKDDO for debugging purposes. The following code will do the task:

```
unsigned short number_of_attribute = cddo->dataCount();
unsigned short number_of_prop;
unsigned short number_of_data_prop;
// list DDO properties
for (short k = 1; k <= number_of_prop; k++) {
     cout << k << " Property Name = " << cddo->getPropertyName(k) <<
     ",\t   value = " << cddo->getProperty(k)  << endl;
}
// list data-items and their properties
for (unsigned short i = 1; i <= number_of_attribute; i++) {
     cout << i << " Attr. Name = " << cddo->getDataName(i) <<
     << ",\t   value = " << cddo->getData(i) << endl;
     number_of_data_prop = cddo->dataPropertyCount(i);
     for (unsigned short j = 1; j <= number_of_data_prop; j++) {
          cout << "\t" << j << " Data Prop. Name = "
               << cddo->getDataPropertyName(i, j)
               << ",\t   value = " << cddo->getDataProperty(i, j)
               << endl;
     }
}
```

DDO representing Digital Library information

A DDO associated with DKDatastoreDL has some specific information to represent the Digital Library document metaphor: document, folder, parts, item, item-id, rank, etcetera. The following sections explain how this information is recorded.

DDO properties

As discussed before, the type of item, either a document or folder, is recorded as a property of the DDO, under the name DK_PROPERTY_ITEM_TYPE. To get the item-type of the DDO, you call:

```
DKAny any = cddo->getPropertyByName(DK_PROPERTY_ITEM_TYPE);
if (!any.isNull()) {
    unsigned short item_type = (unsigned short) any;
}    ...            // do something
```

After the call, the item_type will be equal to DK_DOCUMENT for document, or DK_FOLDER if it is a folder. The if-statement is a check to make sure that the property exists. See “Adding properties to a DDO” on page 26 and “Getting the properties” on page 27 for more information.

Pid

The Pid records two important information specific to Digital Library. The object-type indicates the index-class the DDO belongs to. The id contains the item-id of the associated item in the datastore. See “Pid” on page 25.

Representing Digital Library documents

A DDO representing a Digital Library document has a property DK_PROPERTY_ITEM_TYPE equal to DK_DOCUMENT, as described above.

A Pid of a document DDO contains an index-class name as the object-type, and an item-id in the Pid’s id. The parts inside a document are represented as a DKParts object, which is a collection of blobs, each of which is represented as a DKBlobDL object. A document DDO has a specific attribute with a reserved name DKPARTS, the value of which is a DKAny containing a reference to such a DKParts object. To get to each part in a document, you need to pull out DKParts first, then create an iterator to iterate over each part as you would normally process a collection. If the document does not have a part at all, the DKPARTS will have a null value; the DKPARTS attribute will always be present in a document DDO created by the datastore.

Documents returned as the result of parametric and text (or combined) queries can have a transient attribute DKRANK, the value of which is a DKAny containing a long integer rank computed by the text search engine.

Sample code to create and process DKParts is given in “Creating and using a DKPARTS attribute” on page 30, “Retrieving a document or folder” on page 59, and “Creating, updating, and deleting documents or folders” on page 61.

Representing Digital Library folders

A DDO representing a Digital Library folder has a property DK_PROPERTY_ITEM_TYPE equal to DK_FOLDER. Similar to a document DDO, its Pid contains an index-class name as the object-type, and item-id in the Pid’s id. The folder table of contents

inside a Digital Library folder is represented as a DKFolder object, which is a collection of DDOs, each representing an item (either a document or another folder). A folder DDO has a specific attribute with a reserved name DKFOLDER, the value of which is a DKAny containing a reference to such a DKFolder object.

To get to each DDO member of the folder, you need to pull out DKFolder first, then create an iterator to iterate over each item member as you would normally process a collection. If the folder does not have a member at all, the DKFOLDER will have a null value; the DKFOLDER attribute will always be present in a folder DDO created by the datastore.

Sample code to create and process DKFolder is given in "Creating and using DKFOLDER attribute" on page 31, "Retrieving a document or folder" on page 59, and "Creating, updating, and deleting documents or folders" on page 61.

DDO representing Text Search Engine information

A DDO associated with DKDatastoreTS has some specific information to represent text search results. This does not have a property item-type like DKDatastoreDL. The format of the id is also different. A DDO resulting from a text query corresponds to a text part inside a Digital Library item; it has a set of standard attributes as described in the listing below. As a convention, the attribute value is always contained inside an DKAny object.

DKDLITEMID
The Digital Library item-id where this text part belongs. This item-id is used to retrieve the whole item from the Digital Library datastore.

DKPARTNO
A long integer part-number of this text part. It is used together with the item-id to retrieve this part from the Digital Library datastore.

DKREPTYPE
The Digital Library rep-type of this text part. This attribute, when used in conjunction with the item-id and part-number, retrieves the part from the Digital Library datastore.

DKRANK
A long integer rank signifies the relevance of this part to the results set of the text query. A higher rank means a better match. See the *Text Search Engine: Application Programming Reference* manual for further information.

DKDSIZE
An integer number of word-occurences (only in the results of boolean queries). See the *Text Search Engine: Application Programming Reference* manual for further information.

DKRCNT
An integer number of matches (only in the results of boolean queries). See the *Text Search Engine: Application Programming Reference* manual for further information.

The Pid for a text search DDO has the following information:

datastore type
TS

datastore name
The server name used to connect to the datastore

object type
Text search index

id Text Search Engine document ID

Creating and using a DKPARTS attribute

As mentioned earlier, a DKPARTS attribute in a DDO represents the collection of parts in a document. The value of this attribute is a DKParts object, a collection of XDOs. This attribute is set during DDO retrieve; or it can be created and set by you, as shown below:

```
DKDatastoreDL dsDL;
// create a new DKParts, collection of parts
DKParts* parts = new DKParts;
// create a new XDO blob
DKBlobDL* blob = new DKBlobDL(&dsDL);
// create Pid for this XDO object
DKPidXDODL pid;
// set part number to 5
pid.setPartId(5);
// the item-id this part belongs to
pid.setId("LN#U5K6ARLGM3DB4");
// set the Pid for the XDO blob
blob->setPid(&pid);
// set content class type GIF
blob->setContentClass(DK_CC_GIF);
// set rep type for the part
blob->setRepType(DK_REP_NULL);
// set the blob's content
blob->setContentFromClientFile("choice.gif");
// the viewer program on AIX
blob->setInstanceOpenHandler("xv");

DKAny any = (dkDataObjectBase*) blob;
// add the blob to the parts collection
parts->addElement(any);

...            // create and add some more blobs to the
...            // collection as necessary
// create a ddo
DKDDO* ddo = new DKDDO;
...            // sets some of its attributes
// set the type to document DDO
DKAny any = DK_DOCUMENT;
ddo->addProperty(DK_PROPERTY_ITEM_TYPE, any);

// create DKPARTS attribute and sets it to refer to the DKParts object
// add attribute "DKParts"
unsigned short data_id = ddo->addData(DKPARTS);
// add type property
any = DK_COLLECTION_XDO;
ddo->addDataProperty(data_id,DK_PROPERTY_TYPE,any);
// add nullable property
any = (DKBoolean) TRUE;
ddo->addDataProperty(data_id,DK_PROPERTY_NULLABLE,any);
any = (dkCollection*) parts;
// sets the attribute value
ddo->setData(data_id, any);
```

Once a DKParts is set to be an attribute value of a DDO, the DDO owns it and will take care of its deletion.

Getting back the parts from a DDO is a lot simpler:

```
// get DKPARTS data-id
data_id = ddo->dataId(DKPARTS);
// parts not found
if (data_id == 0) {
     DKException exc(" parts data-item not found");
     DKTHROW exc;
     }
// get the parts collection
any = ddo->getData(data_id);
DKParts* pCol = (DKParts*) any.value();
// create iterator and process the part collection member one by one
if (pCol != NULL) {
    DKAny* element;
    DKBlobDL* blob;
    dkIterator* iter = pCol->createIterator();
    while (iter->more()) {
           element = iter->next();
           blob = (DKBlobDL*) element->value();
           if (blob != NULL) {
               // display the blob using the viewer
               blob->open();
               // other processing
               ...
           }
    }
    delete iter;
}
```

Creating and using DKFOLDER attribute

In a folder DDO, the DKFOLDER attribute represents a collection of folders and documents belonging to this folder. The value of this attribute is a `DKFolder` object, a collection of DDOs. Similar to `DKPARTS`, `DKFOLDER` is set during DDO retrieve; it can also be created and set by you, as shown below:

```
DKDatastoreDL dsDL;
DKFolder* folder = new DKFolder;
// create a new DKFolder, collection of DDO
DKDDO* member = new DKDDO;
// create the first member of this folder
// sets the member DDO attributes and properties
...
// add member to the folder collection
folder->addElement(member);
...
// create and add some more member DDO to the
// DDO collection as necessary
...
// create a folder ddo
DKDDO* ddo = new DKDDO;
// sets some of its attributes
...
// set the type to folder DDO
DKAny any = DK_FOLDER;
ddo->addProperty(DK_PROPERTY_ITEM_TYPE, any);

 // create DKFOLDER attribute and sets it to refer to the DKFolder object
 unsigned short data_id = ddo->addData(DKFOLDER);
 // add attribute "DKFolder"

any = DK_COLLECTION_DDO;
// add type property
ddo->addDataProperty(data_id,DK_PROPERTY_TYPE,any);
// add nullable property
any = (DKBoolean) TRUE;
```

```
ddo->addDataProperty(data_id,DK_PROPERTY_NULLABLE,any);
any = (dkCollection*) folder;
// sets the attribute value
ddo->setData(data_id, any);
```

Once a DKFolder is set to be an attribute of a DDO, the DDO owns it and will take care of its deletion.

Getting back the folder from a DDO is also a lot simpler:

```
// get DKFOLDER data-id
data_id = ddo->dataId(DKFOLDER);
if (data_id == 0) {  // folder not found
     DKException exc(" folder data-item not found");
     DKTHROW exc;
     }
// get the parts collection
any = ddo->getData(data_id);
DKFolder* fCol = (DKFolder*) any.value();
// create iterator and process the DDO collection member one by one
if (fCol != NULL) {
    DKAny* element;
    DKDDO* item;
    dkIterator* iter = fCol->createIterator();
    while (iter->more()) {
           element = iter->next();
           item = (DKDDO*) element->value();
           if (item != NULL) {
               // process the member DDO
               item->retrieve();
               // other processing
               ...
           }
    }
    delete iter;
}
```

Deleting a DDO

A DKDDO is deleted by calling its destructor. The effect would be deleting the DDO in memory; the persistent copy in the datastore is unchanged. In contrast, the method del() in the DDO deletes the persistent copy in the datastore, with its representation in memory being unchanged. Since its attribute values are kept in DKAny, the deletion follows the rule of DKAny deletion. In addition to that, the destructor will delete object-references to dkCollection and dkDataObjectBase; this would include references to DKParts, DKFolder, DKDDO, and DKBlobDL. In this sense, DKDDO destructor performs deep delete.

Using XDO

An XDO represents a single part in Digital Library. There are two types of XDO; DKBlobDL and DKClobDL. DKBlobDL is for binary objects and DKClobDL is for character objects. Both DKBlobDL and DKClobDL require datastore DKDatastoreDL as input to create the object instance.

XDO Pid

An XDO needs to have a Pid in order to store its data persistently. The item ID and part ID of DKPidXDODL are required for XDO to locate the persistent data in the Digital Library datastore.

XDO data members

You need to be aware of some important information about an XDO, such as:

- RepType
- ContentClass
- AffiliatedType
- AffiliatedData
- SearchEngine
- SearchIndex
- SearchInfo

Without setting any value for these members, the object content will not be indexed by a search engine, and the default values will be provided as following:

```
RepType        is  "FRN$NULL"
ContentClass   is   DK_CC_UNKNOWN
AffiliatedType is   DK_BASE
AffiliatedData is   NULL
```

NOTE: For the valid values of ContentClass, please see the file DKConstant2.h, shipped with this product in the include directory.

Indexing XDOs by search engines

If you want the object content to be indexed by a search engine (for example Text Search Engine), the values of **SearchEngine**, **SearchIndex** and **SearchInfo** are required. Please be aware of the SearchIndex value being a combination of two names: search service name and search index name. For example, if in System Admin Client you have the text search server named `TM`, and you defined a SearchIndex named `TMINDEX` under it, the correct value for the SearchIndex is `TM-TMINDEX`.

For the object that is to be indexed by Text Search Engine, the value of **SearchEngine** must be `SM`, and for the object that is to be indexed by QBIC, the value of **SearchEngine** must be `QBIC`. The **SearchIndex** of QBIC is a combination of three values: QBIC database name, QBIC catalog name, and QBIC server name. For example, if the QBIC database name is `SAMPLEDB`, the QBIC catalog name is `SAMPLECAT`, and the QBIC server name is `QBICSRV`, then the correct value for the **SearchIndex** would be `SAMPLEDB-SAMPLECAT-QBICSRV`.

Please refer to the files LoadSampleTSQBIC.cpp and LoadFolderTSQBIC.cpp included with the product, as working code examples to load data, or to create a folder and load data. These are located in the samples directory.

NOTE: When adding a part object to be indexed by a search engine, don't set the repType. Currently the Text Search Engine search manager only works with the default repType `FRN$NULL`.

Programming tips

The combination of item ID, part ID and repType is the key to identify an XDO. To handle a stand-alone XDO, you need to provide the item ID and part ID, while repType is optional since the system will provide a default value for it.

During the add() operation, if you set part ID to zero, the system will assign an available part ID for it. You can retrieve the part ID value after add(), if you want to record the assigned part ID for later use.

You can use the following statement immediately after add(), to obtain the system assigned part ID:

```
unsigned long  partID = ((DKPidXDODL*)(axdo->getPid()))
                        ->getPartId();
```

Important note: There are two situations that a valid part ID is required and you can't set part ID to 0:

1. Adding part to be indexed by a search engine, by search manager
2. Adding a large object that will be divided into MAXPIECE size pieces (see "Handling large objects" on page 13)

XDO as a part of DDO instead of stand alone XDO

In Digital Library, when DDO is a document which has a collection of part objects, an XDO represents a single part object. You can manipulate the XDO as a component of DDO, or as a stand alone object. To handle it as a stand alone object, you need to know the existing item ID for the XDO. To handle it as a part of DDO you do not need to know the item ID; the DDO will provide the item ID for the XDO.

XDO as a part of DDO

For the XDO as a part of DDO, please refer to the file TLoadSample.cpp, shipped with this product as the working code example. It is located in the samples directory.

The following are the major statements to relate the XDO with DDO:

```
//create DDO
DKPid pid = new DKPid();
pid.setObjectType(indexClassName);
DKDDO ddo = new DKDDO(dsDL, pid);
ddo.addProperty(DK_PROPERTY_ITEM_TYPE, new Short(DK_DOCUMENT));
...
...
DKParts* parts = new DKParts;
//create XDO
DKBlobDL  axdo = new DKBlobDL(dsDL);
DKPidXDODL apid = new DKPidXDODL();
apid.setPartId(partId);
axdo.setPid(apid);
axdo.setContentClass(DK_CC_GIF);
axdo.setAffiliatedType(DK_BASE);
axdo.setContentFromClientFile(imageNames[i]);

//add XDO to the DKParts collection
parts.addElement(axdo);
...
...
//add DDO
dataId = ddo.addData(DKPARTS);
ddo.addDataProperty(dataId, DK_PROPERTY_TYPE, new Short(DK_COLLECTION_XDO));
ddo.setData(dataId, parts);
ddo.add();
```

Stand alone XDO

The following code examples are for a stand alone XDO.

Add an XDO from buffer:

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <DKString.hpp>

#include <DKConstant.h>
#include <DKDatastoreDL.hpp>
#include <DKException.hpp>
#include <DKBlobDL.hpp>

void main(int argc, char *argv[])
{
  DKDatastoreDL dsDL;
  DKString itemId, fileName;
  unsigned long partId;
  partId = 0;                              //let system decide the part ID
  itemId = "CPPIORH4JBIXWIY0";             //existing item ID
  fileName = "g:\\test\\cheetah.gif";  //a Windows NT file to be added
  try
  {
                                           //connection to datastore
    dsDL.connect("LIBSRVR","FRNADMIN","PASSWORD","");
    DKBlobDL* axdo = new DKBlobDL(&dsDL);      //create XDO
    DKPidXDODL*  apid = new DKPidXDODL;        //create Pid
    apid->setPartId(partId);                   //set part ID
    apid->setId(itemId);                       //set item ID
    axdo->setPid(apid);                        //set Pid to XDO
    axdo->setContentClass(DK_CC_GIF);          //set ContentClass
    axdo->setContentFromClientFile(fileName);  //set file content to buffer area
    axdo->add();                               //add from buffer
                                               //display the part ID after add
    cout<<"after add partId="<<((DKPidXDODL*)
         (ablobDL->getPid()))->getPartId()<<endl;
    delete axdo;                               //call destructor
    delete apid;                               //call destructor
    dsDL.disconnect();                         //disconnect from datastore
  }
  catch (DKException &exc)
  {
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i = 0; i< exc.textCount();i++)
    {
        cout << "Error text:" << exc.text(i) << endl;
    }
    for (unsigned long g=0; g< exc.locationCount();g++)
    {
        const DKExceptionLocation* p = exc.locationAtIndex(g);
        cout << "Filename: " << p->fileName() << endl;
        cout << "Function: " << p->functionName() << endl;
        cout << "LineNumber: " << p->lineNumber() << endl;
    }
    cout << "Exception Class Name: " << exc.name() << endl;
  }
}//end of main
```

Add an XDO from a file:

```
// txdoaddf.cpp - Test DKBlobDL for add(fileName)
// (this test case requires you to know the existing item ID in the library.)
```

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <DKString.hpp>

#include <DKConstant.h>
#include <DKDatastoreDL.hpp>
#include <DKException.hpp>
#include <DKBlobDL.hpp>

void main(int argc, char *argv[])
{
  DKDatastoreDL dsDL;
  DKString itemId, fileName;
  unsigned long partId;
  partId = 37;                              // part ID 37 not being used yet
  itemId = "CPPIORH4JBIXWIY0";              // existing item ID
  fileName = "/u4/mmdb/test/choice.gif";  // file(in AIX system) to be add
  try
  {
                                             // connection to datastore
    dsDL.connect("LIBSRVR","FRNADMIN","PASSWORD","");
    DKBlobDL* axdo = new DKBlobDL(&dsDL);  // create XDO
    DKPidXDODL*  apid = new DKPidXDODL;    // create Pid
    apid->setPartId(partId);                // set part ID
    apid->setId(itemId);                    // set item ID
    axdo->setPid(apid);                     // set Pid to XDO
    axdo->setRepType("ABCD");               // set representation type
    axdo->setContentClass(DK_CC_GIF);       // set ContentClass
    axdo->add(fileName);                    // add from file
    delete axdo;                            // call destructor
    delete apid;                            // call destructor
    dsDL.disconnect();                      // disconnect from datastore
  }
  catch (DKException &exc)
  {
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i = 0; i< exc.textCount();i++)
    {
         cout << "Error text:" << exc.text(i) << endl;
    }
    for (unsigned long g=0; g< exc.locationCount();g++)
    {
         const DKExceptionLocation* p = exc.locationAtIndex(g);
         cout << "Filename: " << p->fileName() << endl;
         cout << "Function: " << p->functionName() << endl;
         cout << "LineNumber: " << p->lineNumber() << endl;
    }
    cout << "Exception Class Name: " << exc.name() << endl;
  }
}//end of main
```

Add an XDO to be indexed by Text Search Engine:

```
// txdosadd.cpp - Test DKBlobDL for adding an indexed part object
// (this test case requires you to know the existing item ID in the library.)

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <DKString.hpp>

#include <DKConstant.h>
#include <DKDatastoreDL.hpp>
#include <DKException.hpp>
#include <DKBlobDL.hpp>
```

```
void main(int argc, char *argv[])
{
  DKDatastoreDL dsDL;
  DKString itemId, fileName;
  unsigned long partId;
  partId = 37;                              //part ID 37 not being used yet
  itemId = "CPPIORH4JBIXWIY0";              //existing item ID
  fileName = "g:\\test\\content.txt";       //file content will be indexed
  try
  {
                                            //connection to datastore
    dsDL.connect("LIBSRVR","FRNADMIN","PASSWORD","");
    DKBlobDL* axdo = new DKBlobDL(&dsDL);  //create XDO
    DKPidXDODL* apid = new DKPidXDODL;     //create Pid
    apid->setPartId(partId);                //set part ID
    apid->setId(itemId);                    //set item ID
    axdo->setPid(apid);                     //set Pid to XDO
    axdo->setContentClass(DK_CC_ASCII);    //set ContentClass to text

    //---set searchEngine ----- (deprecated)
    axdo->setSearchEngine("SM");
    axdo->setSearchIndex("TM-TMINDEX");
    axdo->setSearchInfo("ENU");

    //---set searchEngine ----- (using extension object)
    DKSearchEngineInfoDL aSrchEx;
    aSrchEx.setSearchEngine("SM");
    aSrchEx.setSearchIndex("TM-TMINDEX");
    aSrchEx.setSearchInfo("ENU");
    axdo->setExtension("DKSearchEngineInfoDL", (dkExtension*)&aSrchEx);

    axdo->setContentFromClientFile(fileName);  //set file content to buffer area
    axdo->add();                               //add from buffer
    cout<<"about to call retrieve()"<<endl;
    axdo->retrieve();                               //retrieve the added object
    axdo->setInstanceOpenHandler("notepad", TRUE);  //set open viewer
    cout<<"about to call open()"<<endl;
    axdo->open();                              //open content with viewer

    delete axdo;                               //call destructor
    delete apid;                               //call destructor
    dsDL.disconnect();                         //disconnect from datastore
  }
  catch (DKException &exc)
  {
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i = 0; i< exc.textCount();i++)
    {
        cout << "Error text:" << exc.text(i) << endl;
    }
    for (unsigned long g=0; g< exc.locationCount();g++)
    {
        const DKExceptionLocation* p = exc.locationAtIndex(g);
        cout << "Filename: " << p->fileName() << endl;
        cout << "Function: " << p->functionName() << endl;
        cout << "LineNumber: " << p->lineNumber() << endl;
    }
    cout << "Exception Class Name: " << exc.name() << endl;
  }
}//end of main
```

Adding an annotation object to XDO: To add an annotation object, insert following statements before the add operation:

```
C++:
//------- set DKAnnotation------- (deprecated)
  axdo –>setAffiliatedType(DK_ANNOTATION);
  DKAnnotation ann;
  ann.setPart(14);
  ann.setPageNumber(1);
  ann.setX(5);
  ann.setY(5);
  axdo–>setAffiliatedData(ann);

//------ set DKAnnotationDL ------ (using extension object)
  axdo->setAffiliatedType(DK_ANNOTATION);
  DKAnnotationDL ann;
  ann.setPart(14);
  ann.setPageNumber(1);
  ann.setX(5);
  ann.setY(5);
  axdo->setExtension("DKAnnotationDL", (dkExtension*)&ann);
```

Retrieve, update, and delete an XDO: To retrieve, update or delete the object in the datastore, you should provide the correct item ID, part ID and repType to identify the object.

```
// txdorud.cpp - Test DKBlobDL for retrieve/update/delete object.

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <DKString.hpp>

#include <DKConstant.h>
#include <DKDatastoreDL.hpp>
#include <DKException.hpp>
#include <DKBlobDL.hpp>

void main(int argc, char *argv[])
{
  DKDatastoreDL dsDL;
  DKString itemId, fileName;
  unsigned long partId;
  partId = 17;                                    //part ID of object
  itemId = "CPPIORH4JBIXWIY0";                    //existing item ID
  fileName = "g:\\test\\choice.gif";              //file content to update
  try
  {
                                                  //connection to datastore
    dsDL.connect("LIBSRVR","FRNADMIN","PASSWORD","");
    DKBlobDL* axdo = new DKBlobDL(&dsDL);         //create XDO
    DKPidXDODL* apid = new DKPidXDODL;            //create Pid
    apid–>setPartId(partId);                      //set part ID
    apid–>setId(itemId);                          //set item ID
    axdo–>setPid(apid);                           //set Pid to XDO
    axdo–>retrieve();                             //retrieve the object
    axdo–>setContentFromClientFile(fileName);     //set file content to buffer area
    axdo–>update();                               //update the object with buffer data
    axdo–>retrieve("new.gif");                    //retrieve content to a file
    axdo–>del();                                  //delete object from datastore
    delete axdo;                                  //call destructor
    delete apid;                                  //call destructor
    dsDL.disconnect();                            //disconnect from datastore
  }
  catch (DKException &exc)
  {
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i = 0; i< exc.textCount();i++)
    {
```

```
        cout << "Error text:" << exc.text(i) << endl;
    }
    for (unsigned long g=0; g< exc.locationCount();g++)
    {
        const DKExceptionLocation* p = exc.locationAtIndex(g);
        cout << "Filename: " << p->fileName() << endl;
        cout << "Function: " << p->functionName() << endl;
        cout << "LineNumber: " << p->lineNumber() << endl;
    }
    cout << "Exception Class Name: " << exc.name() << endl;
  }
}//end of main
```

Indexing an existing XDO using search engines: This example tests the setToBeIndexed method of the DKBlobDL API.

```
//*****************************************************************************
// indexPartxs.cpp - Test setToBeIndexed method of DKBlobDL
// (this test case requires you to know the existing ItemId and partId first)
// invoke: indexPartxs <partId> <repType> <itemId>
// if itemId, partd or repType are not provided, default values will be used.
//*****************************************************************************
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <DKString.hpp>
#include <DKConstant.h>
#include <DKDatastoreDL.hpp>
#include <DKSearchEngineInfoDL.hpp>
#include <DKException.hpp>
#include <DKBlobDL.hpp>

void main(int argc, char *argv[])
{
  DKDatastoreDL dsDL;
  DKString itemId, repType;
  int partId;
  itemId = "N2JJBERBQFK@WTVL";
  repType = "FRN$NULL";
  partId = 10;
  if (argc == 1)
  {
    cout<<"invoke: indexPartxs <partId> <repType> <itemId>"<<endl;
    cout<<" no parameter, following default will be provided:"<<endl;
    cout<<"The supplied default partId = "<<partId<<endl;
    cout<<"The supplied default repType = "<<repType<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 2)
 {
    partId = atoi(argv[1]);
    cout<<"you enter: indexPartxs "<<argv[1]<<endl;
    cout<<"The supplied default repType = "<<repType<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 3)
 {
    partId = atoi(argv[1]);
    repType = DKString(argv[2]);
    cout<<"you enter: indexPartxs "<<argv[1]<<" "<<argv[2]<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 4)
 {
    partId = atoi(argv[1]);
    repType = DKString(argv[2]);
    itemId = DKString(argv[3]);
```

```
    cout<<"you enter: indexPartxs "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
 }
    cout << "connecting Datastore" << endl;
    try
 {
    //replace following with your library server, userid, password
    dsDL.connect("LIBSRVRP","FRNADMIN","PASSWORD");
    cout << "datastore connected" << endl;

    DKBlobDL* axdo = new DKBlobDL(&dsDL);
    DKPidXDODL*  apid = new DKPidXDODL;
    apid ->setPartId(partId);
    apid ->setId(itemId);
    axdo ->setPid(apid);
    axdo ->setRepType(repType);
    cout<<"itemId= "<<(axdo->getPid())->getId()<<endl;
    cout<<"partId= "<<((DKPidXDODL*)(axdo->getPid()))->getPartId()<<endl;
    cout<<"repType= "<<axdo->getRepType()<<endl;

    //--- set searchEngine -----
    cout<<"set search engine and setToBeIndexed()"<<endl;
    DKSearchEngineInfoDL aSrchEx;
    aSrchEx.setSearchEngine("SM");
    aSrchEx.setSearchIndex("TM-TMINDEX");
    aSrchEx.setSearchInfo("ENU");
    axdo->setExtension("DKSearchEngineInfoDL", (dkExtension*)&aSrchEx);
    axdo->setToBeIndexed();
    cout<<"setToBeIndexed() done..."<<endl;

    delete apid;
    delete axdo;
    dsDL.disconnect();
    cout<<"datastore disconnected"<<endl;
 }
  catch(DKException &exc)
 {
   cout << "Error id" << exc.errorId() << endl;
   cout << "Exception id " << exc.exceptionId() << endl;
   for(unsigned long i=0;i< exc.textCount();i++)
   {
    cout << "Error text:" << exc.text(i) << endl;
   }
   for (unsigned long g=0;g< exc.locationCount();g++)
   {
    const DKExceptionLocation* p = exc.locationAtIndex(g);
    cout << "Filename: " << p->fileName() << endl;
    cout << "Function: " << p->functionName() << endl;
    cout << "LineNumber: " << p->lineNumber() << endl;
   }
   cout << "Exception Class Name: " << exc.name() << endl;
  }
  cout << "done ..." << endl;
}
```

Invoking an XDO function: This example demonstrates how to test the DKBlobDL API.

```
//*****************************************************************************
// txdomisc.cpp - Test DKBlobDL
// (this test case requires you to know the existing ItemId and partId first)
// invoke: txdomisc <partId> <repType> <itemId>
// if itemId, partd or repType are not provided, default values will be used.
//*****************************************************************************
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <DKString.hpp>
```

```
#include <DKConstant.h>
#include <DKDatastoreDL.hpp>
#include <DKException.hpp>
#include <DKBlobDL.hpp>

void main(int argc, char *argv[])
{
  DKDatastoreDL dsDL;
  long hsession;
  DKString itemId, repType;
  int partId;
  itemId = "GAWCVGGVFUG428UJ";
  repType = "FRN$NULL";
  partId = 2;

  cout <<"argc is "<<argc<<endl;
  if (argc == 1)
  {
    cout<<"invoke: txdomisc <partId> <repType> <itemId>"<<endl;
    cout<<" no parameter, following default will be provided:"<<endl;
    cout<<"The supplied default partId = "<<partId<<endl;
    cout<<"The supplied default repType = "<<repType<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
  }
  else if (argc == 2)
  {
    partId = atoi(argv[1]);
    cout<<"you enter: txdomisc "<<argv[1]<<endl;
    cout<<"The supplied default repType = "<<repType<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
  }
  else if (argc == 3)
  {
    partId = atoi(argv[1]);
    repType = DKString(argv[2]);
    cout<<"you enter: txdomisc "<<argv[1]<<" "<<argv[2]<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
  }
  else if (argc == 4)
  {
    partId = atoi(argv[1]);
    repType = DKString(argv[2]);
    itemId = DKString(argv[3]);
    cout<<"you enter: txdomisc "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
  }
    cout << "connecting Datastore" << endl;
    try
  {
    dsDL.connect("LIBSRVRH","FRNADMIN","PASSWORD");
    cout << "datastore connected" << endl;
    hsession = (long)(dsDL.connection()->handle());
    cout << "datastore handle" << hsession << endl;

    DKBlobDL* axdo = new DKBlobDL(&dsDL);
    DKPidXDODL*  apid = new DKPidXDODL;
    apid ->setPartId(partId);
    apid ->setId(itemId);
    axdo ->setPid(apid);
    axdo ->setRepType(repType);
    cout<<"itemId= "<<(axdo->getPid())->getId()<<endl;
    cout<<"partId= "<<((DKPidXDODL*)(axdo->getPid()))->getPartId()<<endl;
    cout<<"repType= "<<axdo->getRepType()<<endl;

    //== before retrieve
    cout<<"before retrieve:"<<endl;
    cout<<" content class="<<axdo->getContentClass()<<endl;
```

```
    cout<<" content length="<<axdo->length();
    cout<<" (the length of this object instance - in memory)"<<endl;
    cout<<" getSize="<<axdo->getSize();
    cout<<" (get the object size without retrieving object)"<<endl;
    cout<<" createdTimestamp="<<axdo->getCreatedTimestamp()<<endl;
    cout<<" updatedTimestamp="<<axdo->getUpdatedTimestamp()<<endl;
    axdo->retrieve();
    //== after retrieve
    cout<<"after retrieve:"<<endl;
    cout<<" content class="<<axdo->getContentClass()<<endl;
    cout<<" content length="<<axdo->length();
    cout<<" (the length of this object instance - in memory)"<<endl;
    cout<<" getSize="<<axdo->getSize();
    cout<<" (get the object size without retrieving object)"<<endl;
    cout<<" createdTimestamp="<<axdo->getCreatedTimestamp()<<endl;
    cout<<" updatedTimestamp="<<axdo->getUpdatedTimestamp()<<endl;
    cout<<" mimeType="<<axdo->mimeType()<<endl;
    int atype = axdo->getAffiliatedType();
    cout<<" affiliatedType= "<<axdo->getAffiliatedType()<<endl;
    if (atype == DK_ANNOTATION)
    {
     cout <<"  pageNumber= "<<axdo->getAffiliatedData().getPageNumber()<<endl;
     cout <<"  partId= "<<axdo->getAffiliatedData().getPart()<<endl;
     cout <<"  X=<<axdo->getAffiliatedData().getX()<<endl;
     cout <<"  Y=<<axdo->getAffiliatedData().getY()<<endl;
    }
    //== open content
    int concls = axdo->getContentClass();
    if (concls == DK_CC_ASCII)
        axdo->setInstanceOpenHandler("notepad", TRUE);
    else if (concls == DK_CC_GIF)
        axdo->setInstanceOpenHandler("lviewpro", TRUE);
    else if (concls == DK_CC_AVI)
        axdo->setInstanceOpenHandler("mplay32", TRUE);
    axdo->open();

    delete apid;
    delete axdo;
    dsDL.disconnect();
    cout<<"datastore disconnected"<<endl;
  }
  catch(DKException &exc)
  {
   cout << "Error id" << exc.errorId() << endl;
   cout << "Exception id " << exc.exceptionId() << endl;
   for(unsigned long i=0;i< exc.textCount();i++)
   {
    cout << "Error text:" << exc.text(i) << endl;
   }
   for (unsigned long g=0;g< exc.locationCount();g++)
   {
    const DKExceptionLocation* p = exc.locationAtIndex(g);
    cout << "Filename: " << p->fileName() << endl;
    cout << "Function: " << p->functionName() << endl;
    cout << "LineNumber: " << p->lineNumber() << endl;
   }
   cout << "Exception Class Name: " << exc.name() << endl;
  }
  cout << "done ..." << endl;
}
```

Add an XDO Media Object: For every media object added, an entry is created in the Digital Library FRN$MEDIA table. That entry contains the information about the media user data. The physical media object is stored in the VideoCharger server specified in the network table of Digital Library.

```
//*****************************************************************************
// txdoAddVS.cpp - Test DKBlobDL for adding the media object
// (this test case requires you to know the existing ItemId and partId first)
// invoke: txdoAddVS <fileName to be added> <partId> <itemId>
// you can also code the itemId, partId and fileName in this program and
// just invoke as:  txdoAddVS
//*****************************************************************************
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <DKString.hpp>
#include <DKConstant.h>
#include <DKDatastoreDL.hpp>
#include <DKMediaStreamInfoDL.hpp>
#include <DKException.hpp>
#include <DKBlobDL.hpp>

void main(int argc, char *argv[])
{
  DKString itemId, fileName;
  int partId;
  itemId = "K1A04EWBVHJAV1D7";
  partId = 22;
  fileName = "/icing1.mpg1";
  if (argc == 1)
  {
    cout<<"invoke: txdoAddVS <fileName> <partId> <itemId>"<<endl;
    cout<<" no parameter, following default will be provided:"<<endl;
    cout<<"The supplied default fileName = "<<fileName<<endl;
    cout<<"The supplied default partId = "<<partId<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 2)
 {
    fileName = DKString(argv[1]);
    cout<<"you enter: txdoAddVS "<<argv[1]<<endl;
    cout<<"The supplied default partId = "<<partId<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 3)
 {
    fileName = DKString(argv[1]);
    partId = atoi(argv[2]);
    cout<<"you enter: txdoAddVS "<<argv[1]<<" "<<argv[2]<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 4)
 {
    fileName = DKString(argv[1]);
    partId = atoi(argv[2]);
    itemId = DKString(argv[3]);
    cout<<"you enter: txdoAddVS "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
 }
  try
 {
  // connect to datastore
  cout << "Connecting datastore ..." << endl;
  DKDatastoreDL dsDL;
  // replace following with your library server, userid, password
  dsDL.connect("LIBXDB2","FRNADMIN","PASSWORD");
  cout << "datastore connected" << endl;

  // *** create xdo and pid
  DKBlobDL* axdo = new DKBlobDL(&dsDL);
  DKPidXDODL*  apid = new DKPidXDODL;
  apid ->setPartId(partId);
  apid ->setId(itemId);
```

```
                        axdo ->setPid(apid);
                        // *** you must use the content class DK_CC_IBMVSS for a media object
                        axdo ->setContentClass(DK_CC_IBMVSS);
                        cout <<"itemId= "<<(axdo->getPid())->getId()<<endl;
                        cout <<"partId= "<<((DKPidXDODL*)(axdo->getPid()))->getPartId()<<endl;
                        cout <<"repType= "<<axdo->getRepType()<<endl;
                        cout <<"content class="<< axdo->getContentClass()<<endl;

                        // *** setup DKMediaStreamInfoDL
                        DKMediaStreamInfoDL aVS;
                        aVS.setMediaFullFileName(fileName);
                        aVS.setMediaObjectOption(DK_VS_SINGLE_OBJECT);
                        aVS.setMediaHostName("vcharger1a.stl.ibm.com");
                        aVS.setMediaUserId("root");
                        aVS.setMediaPassword("video2");

                        //following are optional, if not set then default value will be provided
                        aVS.setMediaNumberOfUsers(1);
                        aVS.setMediaAssetGroup("AG");
                        // *** same as defined in VideoCharger server
                        aVS.setMediaType("MPEG1");
                        aVS.setMediaResolution("SIF");
                        aVS.setMediaStandard("NTSC");
                        aVS.setMediaFormat("SYSTEM");

                        axdo ->setExtension("DKMediaStreamInfoDL", (dkExtension*)&aVS);
                        cout <<"about to do add()"<<endl;
                        axdo ->add();
                        cout<<"Object added successfully "<<endl;

                        cout<<"after added check for status:"<<endl;
                        DKBoolean flag2 = axdo->isCategoryOf(DK_MEDIA_OBJECT);
                        if (flag2)
                        {
                          DKMediaStreamInfoDL* mediaInfo = (DKMediaStreamInfoDL*)
                                                           axdo->getExtension("DKMediaStreamInfoDL");
                          cout<<" copyRate="<<mediaInfo->getMediaCopyRate()<<endl;
                          cout<<" mediaType="<<mediaInfo->getMediaType()<<endl;
                          cout<<" mediaFrameRate="<<mediaInfo->getMediaFrameRate()<<endl;
                          cout<<" mediaState="<<mediaInfo->getMediaState()<<endl;
                          cout<<" mediaTimestamp="<<mediaInfo->getMediaTimestamp()<<endl;
                          cout<<" MediaState(dynamic)="<<axdo->retrieveObjectState(DK_MEDIA_OBJECT)<<endl;
                        }

                        dsDL.disconnect();
                        cout<<"datastore disconnected"<<endl;
                       }
                        catch(DKException &exc)
                       {
                         cout << "Error id" << exc.errorId() << endl;
                         cout << "Exception id " << exc.exceptionId() << endl;
                         for(unsigned long i=0;i< exc.textCount();i++)
                         {
                          cout << "Error text:" << exc.text(i) << endl;
                         }
                         for (unsigned long g=0;g< exc.locationCount();g++)
                         {
                          const DKExceptionLocation* p = exc.locationAtIndex(g);
                          cout << "Filename: " << p->fileName() << endl;
                          cout << "Function: " << p->functionName() << endl;
                          cout << "LineNumber: " << p->lineNumber() << endl;
                         }
                         cout << "Exception Class Name: " << exc.name() << endl;
                        }
                        cout << "done ..." << endl;
                      }
```

Delete an XDO Media Object: This example shows how to delete an XDO media object.

```
//*************************************************************************
// txdodelvs.cpp - test DKBlobDL for delete and using extension object
// (this test case requires you to know the existing itemId and partId first)
// invoke: txdodelvs <partId> <repType> <itemId>
// You can code the partId, repType, and itemId in the program and invoke
// as: txdodelvs
//*************************************************************************
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <DKString.hpp>
#include <DKConstant.h>
#include <DKDatastoreDL.hpp>
#include <DKException.hpp>
#include <DKMediaStreamInfoDL.hpp>
#include <DKBlobDL.hpp>

void main(int argc, char *argv[])
{
  DKDatastoreDL dsDL;
  DKString itemId, repType;
  int partId;
  itemId = "Y68M1I@VYDG8SPQ4";
  partId = 1;
  repType = "FRN$NULL";
  if (argc == 1)
  {
    cout<<"invoke: txdodelvs <partId> <repType> <itemId>"<<endl;
    cout<<" no parameter, following default will be provided:"<<endl;
    cout<<"The supplied default partId = "<<partId<<endl;
    cout<<"The supplied default repType = "<<repType<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 2)
 {
    partId = atoi(argv[1]);
    cout<<"you enter: txdodelvs "<<argv[1]<<endl;
    cout<<"The supplied default repType = "<<repType<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 3)
 {
    repType = DKString(argv[2]);
    partId = atoi(argv[1]);
    cout<<"you enter: txdodelvs "<<argv[1]<<" "<<argv[2]<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 4)
 {
    itemId = DKString(argv[3]);
    repType = DKString(argv[2]);
    partId = atoi(argv[1]);
    cout<<"you enter: txdodelvs "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
 }

  try
  {
    cout << "Connecting datastore ..." << endl;
    // replace following with your library server, userid, password
    dsDL.connect("LIBSRVRP","FRNADMIN","PASSWORD");
    cout << "datastore connected" << endl;

    DKBlobDL* axdo = new DKBlobDL(&dsDL);
    DKPidXDODL*  apid = new DKPidXDODL;
```

```
    apid ->setPartId(partId);
    apid ->setId(itemId);
    axdo ->setPid(apid);
    axdo ->setRepType(repType);
    cout <<"itemId= "<<(axdo->getPid())->getId()<<endl;
    cout <<"partId= "<<((DKPidXDODL*)(axdo->getPid()))->getPartId()<<endl;

    DKBoolean flag2 = axdo->isCategoryOf(DK_MEDIA_OBJECT);
    cout <<"isMediaObject? = "<<flag2<<endl;
    if (flag2)
    {
      DKMediaStreamInfoDL* mediaInfo = (DKMediaStreamInfoDL*)
                                        axdo->getExtension("DKMediaStreamInfoDL");
      cout<<" copyRate="<<mediaInfo->getMediaCopyRate()<<endl;
      cout<<" mediaType="<<mediaInfo->getMediaType()<<endl;
      cout<<" mediaFrameRate="<<mediaInfo->getMediaFrameRate()<<endl;
      cout<<" mediaState="<<mediaInfo->getMediaState()<<endl;
      cout<<" mediaTimestamp="<<mediaInfo->getMediaTimestamp()<<endl;
      cout<<" MediaState(dynamic)="<<axdo->retrieveObjectState(DK_MEDIA_OBJECT)<<endl;

      cout<<"about to set the delete option for media object..."<<endl;
      DKAny delOpt = DK_DELETE_NO_DROPITEM_MEDIA_AVAIL;
      axdo->setOption(DK_OPT_DL_DELETE_OPTION, delOpt);
      DKAny opt;
      axdo->getOption(DK_OPT_DL_DELETE_OPTION, opt);
      long lopt = opt;
      cout<<"The setted delete option = "<<lopt<<endl;

    }
    cout<<"about to do del()"<<endl;
    axdo->del();
    cout<<"del successfully..."<<endl;
    flag2 = axdo->isCategoryOf(DK_MEDIA_OBJECT);
    cout<<"after delete isMediaObject? = "<<flag2<<endl;
    delete axdo;
    delete apid;
    dsDL.disconnect();
    cout<<"datastore disconnected"<<endl;
 }
  catch(DKException &exc)
 {
   cout << "Error id" << exc.errorId() << endl;
   cout << "Exception id " << exc.exceptionId() << endl;
   for(unsigned long i=0;i< exc.textCount();i++)
   {
    cout << "Error text:" << exc.text(i) << endl;
   }
   for (unsigned long g=0;g< exc.locationCount();g++)
   {
    const DKExceptionLocation* p = exc.locationAtIndex(g);
    cout << "Filename: " << p->fileName() << endl;
    cout << "Function: " << p->functionName() << endl;
    cout << "LineNumber: " << p->lineNumber() << endl;
   }
   cout << "Exception Class Name: " << exc.name() << endl;
  }
  cout << "done ..." << endl;
}
```

Retrieve an XDO Media Object: This example shows how to retrieve an XDO media object. The retrieved object contains only the media meta data, not the media object itself.

```
//*************************************************************************
// txdoretxs.cpp - test the xdo retrieve/open and using the extension object.
// (this test case requires you to know the existing itemId and partId first)
// invoke: txdoretxs <partId> <repType> <itemId>
```

```
// You can code the itemId, partId and repType in the program and invoke
// as: txdoretxs
//*****************************************************************************
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <DKString.hpp>
#include <DKConstant.h>
#include <DKDatastoreDL.hpp>
#include <DKException.hpp>
#include <DKBlobDL.hpp>
#include <DKAnnotationDL.hpp>
#include <DKSearchEngineInfoDL.hpp>
#include <DKMediaStreamInfoDL.hpp>

void main(int argc, char *argv[])
{
  DKDatastoreDL dsDL;
  DKString itemId, repType;
  int partId;
  itemId = "K1A04EWBVHJAV1D7";
  partId = 1;
  repType = "FRN$NULL";
  if (argc == 1)
  {
    cout<<"invoke: txdoretxs <partId> <repType> <itemId>"<<endl;
    cout<<" no parameter, following default will be provided:"<<endl;
    cout<<"The supplied default partId = "<<partId<<endl;
    cout<<"The supplied default repType = "<<repType<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 2)
 {
    partId = atoi(argv[1]);
    cout<<"you enter: txdoretxs "<<argv[1]<<endl;
    cout<<"The supplied default repType = "<<repType<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 3)
 {
    repType = DKString(argv[2]);
    partId = atoi(argv[1]);
    cout<<"you enter: txdoretxs "<<argv[1]<<" "<<argv[2]<<endl;
    cout<<"The supplied default itemId = "<<itemId<<endl;
 }
  else if (argc == 4)
 {
    itemId = DKString(argv[3]);
    repType = DKString(argv[2]);
    partId = atoi(argv[1]);
    cout<<"you enter: txdoretxs "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
 }

  try
 {
    cout << "Connecting datastore ..." << endl;
    // replace following with your library server, userid, password
    dsDL.connect("LIBSRVRX","FRNADMIN","PASSWORD");
    cout << "datastore connected" << endl;

    DKBlobDL* axdo = new DKBlobDL(&dsDL);
    DKPidXDODL*  apid = new DKPidXDODL;
    apid ->setPartId(partId);
    apid ->setId(itemId);
    axdo ->setPid(apid);
    axdo ->setRepType(repType);
    cout <<"itemId= "<<(axdo->getPid())->getId()<<endl;
```

```
cout <<"partId= "<<((DKPidXDODL*)(axdo->getPid()))->getPartId()<<endl;

DKBoolean flag = axdo->isCategoryOf(DK_INDEXED_OBJECT);
DKBoolean flag2 = axdo->isCategoryOf(DK_MEDIA_OBJECT);
cout <<"isIndexed? = "<<flag<<endl;
cout <<"isMediaObject? = "<<flag2<<endl;
if (flag)
{
  DKSearchEngineInfoDL* srchInfo = (DKSearchEngineInfoDL*)
                                   axdo->getExtension("DKSearchEngineInfoDL");
  cout<<" ServerName="<<srchInfo->getServerName()<<endl;
  cout<<" TextIndex="<<srchInfo->getTextIndex()<<endl;
  cout<<" srchEngine="<<srchInfo->getSearchEngine()<<endl;
  cout<<" srchIndex="<<srchInfo->getSearchIndex()<<endl;
  cout<<" indexedState="<<axdo->retrieveObjectState(DK_INDEXED_OBJECT)<<endl;
}

if (flag2)
{
  DKMediaStreamInfoDL* mediaInfo = (DKMediaStreamInfoDL*)
                                   axdo->getExtension("DKMediaStreamInfoDL");
  cout<<" copyRate="<<mediaInfo->getMediaCopyRate()<<endl;
  cout<<" mediaType="<<mediaInfo->getMediaType()<<endl;
  cout<<" mediaFrameRate="<<mediaInfo->getMediaFrameRate()<<endl;
  cout<<" mediaState="<<mediaInfo->getMediaState()<<endl;
  cout<<" mediaTimestamp="<<mediaInfo->getMediaTimestamp()<<endl;
  cout<<" MediaState(dynamic)="<<axdo->retrieveObjectState(DK_MEDIA_OBJECT)<<endl;
}

 cout<<"before retrieve..."<<endl;
 cout <<" length of lobdata = "<<axdo->length()<<endl;
 cout<<" size of lobdata = "<<axdo->getSize()<<endl;
 cout<<" created Timestamp = "<<axdo->getCreatedTimestamp()<<endl;
 cout<<" updated Timestamp = "<<axdo->getUpdatedTimestamp()<<endl;
 axdo->retrieve();
 cout<<"after retrieve..."<<endl;
 cout <<" length of lobdata = "<<axdo-><length()<<endl;
 cout <<" mimeType = "<<axdo->mimeType()<<endl;
 cout <<" size of lobdata = "<<axdo->getSize()<<endl;
 cout<<" created Timestamp = "<<axdo->getCreatedTimestamp()<<endl;
 cout<<" updated Timestamp = "<<axdo->getUpdatedTimestamp()<<endl;

 int atype = axdo->getAffiliatedType();
 cout <<"affiliatedType= "<<axdo->getAffiliatedType()<<endl;
 if (atype == DK_ANNOTATION)
 {
   DKAnnotationDL* ann = (DKAnnotationDL*)axdo->getExtension("DKAnnotationDL");
   cout<<"  pageNumber= "<<ann->getPageNumber()<<endl;
   cout<<"  partId= "<<ann->getPart()<<endl;
   cout<<"  X= "<<ann->getX()<<endl;
   cout<<"  Y= "<<ann->getY()<<endl;
 }
 cout<<"about to do open()..."<<endl;
 axdo->setInstanceOpenHandler("notepad", TRUE); //default use notepad in NT
 int concls = axdo->getContentClass();
 if (concls == DK_CC_GIF)
  axdo->setInstanceOpenHandler("lviewpro", TRUE);   //use lviewpro in NT
  else if (concls == DK_CC_AVI)
   axdo->setInstanceOpenHandler("mplay32", TRUE);    //use mplay32 in NT
   else if (concls == DK_CC_IBMVSS)
    axdo->setInstanceOpenHandler("iscoview", TRUE);   //use iscoview in NT
  axdo->open();

delete axdo;
delete apid;
dsDL.disconnect();
cout<<"datastore disconnected"<<endl;
```

```
  }
   catch(DKException &exc)
  {
   cout << "Error id" << exc.errorId() << endl;
   cout << "Exception id " << exc.exceptionId() << endl;
   for(unsigned long i=0;i< exc.textCount();i++)
   {
    cout << "Error text:" << exc.text(i) << endl;
   }
   for (unsigned long g=0;g< exc.locationCount();g++)
   {
    const DKExceptionLocation* p = exc.locationAtIndex(g);
    cout << "Filename: " << p->fileName() << endl;
    cout << "Function: " << p->functionName() << endl;
    cout << "LineNumber: " << p->lineNumber() << endl;
   }
   cout << "Exception Class Name: " << exc.name() << endl;
  }
  cout << "done ..." << endl;
}
```

Querying the datastore

You can issue a query to a datastore and get back results in the form of a `dkResultSetCursor` or `DKResults`. You can create a query object to represent your query, then invoke the `execute` or `evaluate` method in the query object. With the help of its datastores, the query object performs query processing tasks, such as preparing and executing a query, monitoring the status of query execution and storing the results.

There are three types of query objects: Parametric, Text, and Combined. The Combined query is composed of Text and Parametric queries.

The datastore uses two ways to run a query: execute and evaluate. The execute returns `dkResultSetCursor`, while evaluate returns `DKResults`. The `dkResultSetCursor` is used to handle a large result set, as well as perform delete and update operations on the current position of the result set cursor. You can use the `fetchNextN` method to fetch a group of objects into a collection.

The `dkResultSetCursor` can also be used to re-examine a query by calling the `close()` and `open()` methods. This is described in “Result set cursor” on page 57.

The `DKResults` contains all of the results from the query. You can iterate over the items in the collection forward or backwards. This collection can also be queried, and can be used as a scope for another query.

See “Queryable collection” on page 66 and “Combined query” on page 67, for more information.

Differences between dkResultSetCursor and DKResults

The difference between a `dkResultSetCursor` and a `DKResults` collection is the following:

- the `dkResultSetCursor` works like a datastore cursor; it can be used for large result sets, since the DKDDO it contains are fetched one at a time. It can also be used to re-execute a query, to refresh the first results.
- the `DKResults` contains the entire result set, and it supports the bidirectional iterator

Parametric query

This section explains the ways in which you use the parametric query function.

Formulating a parametric query

The following example is a query string representing a query against the index class GP2DLS2. The condition of the query is to search for all documents or folders with an attribute of DLSEARCH_DocType <> NULL. The maximum number of results that will be returned is limited to five. The content is set to ″yes″, so contents of the document or folder will be returned.

We also specify that Digital Library uses dynamic SQL for this query and that folders and documents be searched. If the attribute name has more than one word or is in DBCS, it should be enclosed in single quotes. If the attribute value is in DBCS, it should be enclosed in double quotes.

```
DKString cmd = "SEARCH=(INDEX_CLASS=GP2DLS2,";
cmd +=  "MAX_RESULTS=5,";
cmd +=  "COND=(DLSEARCH_DocType <> NULL));";
cmd +=  "OPTION=(CONTENT=YES;";
cmd +=  "TYPE_QUERY=DYNAMIC;";
cmd +=  "TYPE_FILTER=FOLDERDOC)";
```

Formulating a parametric query on multiple criteria

The parametric query allows you to specify more than one search criteria. The following example shows how to specify a query against two index classes.

```
DKString cmd = "SEARCH=(INDEX_CLASS=GP2DLS2,MAX_RESULTS=3,";
cmd +=  "COND=(DLSEARCH_DocType <> NULL);";
cmd +=  "INDEX_CLASS=GP2DLS1,MAX_RESULTS=8,";
cmd +=  "COND=('First name' == \"Robert\"));";
cmd +=  "OPTION=(CONTENT=YES;";
cmd +=  "TYPE_QUERY=DYNAMIC;";
cmd +=  "TYPE_FILTER=FOLDERDOC)";
```

Executing a parametric query

The datastore provides a method to create a query object. You use the query object to execute the query and obtain the results. The following example shows how to create a parametric query object and execute the query. Once the query is executed, the results are returned in a DKResults collection.

A complete sample, SamplePQryDL.cpp, is provided in the samples directory.

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN", "USERID", "PASSWORD");
DKAny *element;
DKDDO *item;
DKString cmd = "SEARCH=(INDEX_CLASS=GP2DLS3,";
cmd += "COND=('last name' == \"SUMMERS\"));";
cmd += "OPTION=(CONTENT=YES;";
cmd += "TYPE_QUERY=DYNAMIC;TYPE_FILTER=DOC)";

...
dkQuery* pQry = dsDL.createQuery(cmd);
pQry->execute();
DKAny any = pQry->result();
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more()) {
       element = pIter->next();
       item = (DKDDO*)element->value();
```

```
       // Process the DKDDO
}
delete pIter;
delete pResults;
delete pQry;
dsDL.disconnect();
```

Note: When you delete a DKResults object, all of its members will be deleted also. Make sure that you do not delete the element twice. See “Collections and iterators” on page 21 for further information.

Executing a parametric query from the datastore

The datastore provides a method to execute a query. The following example shows how to execute a parametric query against the datastore. Once the query is executed, the results are returned in a `dkResultSetCursor` object.

A complete sample, `TExecuteDL.cpp`, is provided in the samples directory.

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN", "USERID", "PASSWORD");
DKString cmd = "SEARCH=(INDEX_CLASS=GP2DLS4);";
cmd += "OPTION=(CONTENT=YES;";
cmd += "TYPE_QUERY=DYNAMIC;TYPE_FILTER=FOLDERDOC)";
...

dkResultSetCursor* pCur = dsDL.execute(cmd);
DKDDO *item = 0;
while (pCur->isValid()) {
       item = pCur->fetchNext();
       if (item != 0) {
           // Process the DKDDO
           ...
           delete item;
       }
}
delete pCur;
dsDL.disconnect();
```

Evaluating a parametric query from the datastore

The datastore provides a method to evaluate a query. The following example shows how to evaluate a parametric query against the datastore. Once the query is executed, the results are returned in a `DKResults` collection.

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN", "USERID", "PASSWORD");
DKAny *element;
DKDDO *item;
DKString cmd = "SEARCH=(INDEX_CLASS=GP2DLS5,";
cmd +=  "COND=((DLSEARCH_Date >= \"1995\") AND ";
cmd +=  "(DLSEARCH_Date <= \"1996\")));";
cmd += "OPTION=(CONTENT=NO;";
cmd += "TYPE_QUERY=DYNAMIC;TYPE_FILTER=FOLDERDOC)";

...
DKAny any = dsDL.evaluate(cmd);
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more()) {
       element = pIter->next();
       item = (DKDDO*)element->value();
       // Process the DKDDO
```

```
}
delete pIter;
delete pResults;
dsDL.disconnect();
```

Text query

This section explains the ways in which you use the text query function.

Formulating a text query

This is a query string representing a query against the text index `TMINDEX`. The condition of the query is to search for all text documents with the word `UNIX` or `member`. The maximum number of results that will be returned is five.

```
DKString cmd = "SEARCH=(COND=(UNIX OR member));";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";
```

Formulating a text query on multiple indexes

The text query allows you to specify that a search is to be performed against more than one index. The following example shows how to specify a query against two indexes.

Important: If you specify more than one text search index in the query, the indexes must be the same type. For example, you can specify two precise indexes in the query, but you cannot specify a precise index and a linguistic index within the query.

```
DKString cmd = "SEARCH=(COND=(UNIX OR member));";
cmd += "OPTION=(SEARCH_INDEX=(TMINDEX,TMINDEX2); MAX_RESULTS=5)";
```

Executing a text query

The datastore provides a method to create a query object. You use the query object to execute the query and obtain the results. The following example shows how to create a text query object and execute a query. Once the query is executed, the results are returned in a `DKResults` collection.

A complete sample, `SampleTQryTS.cpp`, is provided in the samples directory.

```
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' ');
DKAny *element;
DKDDO *item;
DKString cmd  = "SEARCH=(COND=('UNIX operating' AND system));";
     cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
...

dkQuery* pQry = dsTS.createQuery(cmd);
pQry->execute();
DKAny any = pQry->result();
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more()) {
       element = pIter->next();
       item = (DKDDO*)element->value();
       // Process the DKDDO
}
delete pIter;
delete pResults;
delete pQry;
dsTS.disconnect();
```

Executing a text query from the datastore

The datastore provides a method to execute a query. The following example shows how to execute a text query against the datastore. Once the query is executed, the results are returned in a `dkResultSetCursor` object.

A complete sample, `TExecuteTS.cpp`, is provided in the samples directory.

```
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' ');
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
...

dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
while (pCur->isValid()) {
       item = pCur->fetchNext();
       if (item != 0) {
           // Process the DKDDO
          ...
          delete item;
       }
}
delete pCur;
dsTS.disconnect();
```

Evaluating a text query from the datastore

The datastore provides a method to evaluate a query The following example shows how to evaluate a text query against the datastore. Once the query is executed, the results are returned in a `DKResults` collection.

```
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' ');
DKAny *element;
DKDDO *item;
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));";
         cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";

...
DKAny any = dsTS.evaluate(cmd);
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more()) {
       element = pIter->next();
       item = (DKDDO*) element->value();
       // Process the DKDDO
       ...
}
delete pIter;
delete pResults;
dsTS.disconnect();
```

Getting match highlighting information for each text query result item

This code sample retrieves match highlighting information for each text query result item during a text query. The MATCH_INFO option is set to YES, indicating that the match highlighting information will be returned. The MATCH_DICT option specifies whether the highlighting information will be obtained using a dictionary. Because the MATCH_DICT option is set to NO, the dictionary will not be used to retrieve highlighting information. The match information is returned in the attribute DKMATCHESINFO in the DKDDO returned from a text query. The value of the

attribute DKMATCHESINFO will be a DKMatchesInfoTS object. The DKMatchesInfoTS object will be deleted when the DKDDO is deleted; do not delete this class yourself.

Important: This process is time consuming because the document is retrieved from DL datastore and analyzed linguistically, and potential matches are determined. These processes will have an impact on the performance of a text query.

```
DKDatastoreTS dsTS;
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));"
cmd += "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5;MATCH_INFO=YES;MATCH_DICT=NO)";

...

dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
DKAny anyObj;
dkDataObjectBase *pDOBase = 0;
DKMatchesInfoTS *pMInfo = 0;
DKMatchesDocSectionTS *pMSect = 0;
DKMatchesParagraphTS *pMPara = 0;
DKMatchesTextItemTS  *pMText = 0;
long i = 0;
long j = 0;
long k = 0;
long m = 0;
long lCCSID = 0;
long lLang = 0;
long lOffset = 0;
long lLen = 0;
long numberSections = 0;
long numberParagraphs = 0;
long numberTextItems = 0;
long numberNewLines = 0;
DKString strDoc;
DKString strSection;
DKString strText;
while (pCur->isValid())
 {
  item = pCur->fetchNext();
  if (item != 0)
  {
   // Process the DKDDO
   for (i = 1; i <= item->dataCount(); i++)
   {
    anyObj = item->getData(i);
    switch (anyObj.typeCode())
    {
     case DKAny::tc_string :
     {
      ...
     break;
     }
     case DKAny::tc_long :
        {
         ...
        break;
        }
        case DKAny::tc_short :
        {
         ...
        break;
        }
        case DKAny::tc_dobase :
        {
```

```
            // process the Match Hightlighting information
            pDOBase = a;
            pMInfo = (DKMatchesInfoTS*)pDOBase;
        if (pMInfo != 0)
        {
          strDoc = pMInfo->getDocumentName();
          numberSections = pMInfo->numberOfSections();
               // loop thru document sections
          for (j = 1; j <= numberSections; j++)
          {
           pMSect = pMInfo->getSection(j);
           strSection = pMSect->getSectionName();
           numberParagraphs = pMSect->numberOfParagraphs();
                 // loop thru section paragraphs
           for (k = 1; k <= numberParagraphs; k++)
           {
            pMPara = pMSect->getParagraph(k);
            lCCSID = pMPara->getCCSID();
            lLang = pMPara->getLanguageId();
            numberTextItems = pMPara->numberOfTextItems();
                 // loop thru paragraph text items
             for (m = 1; m <= numberTextItems; m++)
             {
              pMText = pMPara->getTextItem(m);
              strText = pMText->getText();
                  // if match found in text item get offset and
                  // length of match in text item
              if (pMText->isMatch() == TRUE)
              {
               lOffset = pMText->getOffset();
               lLen = pMText->getLength();
                   }
                numberNewLines = pMText->numberOfNewLines();
               }
              }
             }
             }
             break;
            }
            default :
            {
             break;
            }
           }
          }
          ...
          delete item;
       }
    }
    delete pCur;
    dsTS.disconnect();
```

Getting match highlighting information for a particular text query result item

This code sample retrieves the match highlighting information for a specific item returned from a text query. The match information contains the text of the document and the highlighting information for every match of the corresponding query. The `resultSetCursor` passed into this routine must be in an open state.

Important: This process is time consuming because the document is retrieved from DL datastore and analyzed linguistically, and potential matches are determined. These processes will have an impact on the performance of a text query.

```
DKDatastoreTS dsTS;
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));"
cmd += "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";

...

dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
DKAny anyObj;
dkDataObjectBase *pDOBase = 0;
DKMatchesInfoTS *pMInfo = 0;
DKMatchesDocSectionTS *pMSect = 0;
DKMatchesParagraphTS *pMPara = 0;
DKMatchesTextItemTS  *pMText = 0;
long i = 0;
long j = 0;
long k = 0;
long m = 0;
long lCCSID = 0;
long lLang = 0;
long lOffset = 0;
long lLen = 0;
long numberSections = 0;
long numberParagraphs = 0;
long numberTextItems = 0;
long numberNewLines = 0;
DKString strDoc;
DKString strSection;
DKString strText;
DKString strDID;
DKString strXNAME;
DKString strDataName;
DKPid pid;
while (pCur->isValid())
 {
   item = pCur->fetchNext();
   if (item != 0)
   {
    pid = item->getPid();
    // Process the DKDDO
    for (i = 1; i <= item->dataCount(); i++)
    {
     anyObj = item->getData(i);
     strDataName = item->getDataName(i);
     if (strDataName == "")
     {
      strDID = pid.getId();
     }
     if (strXNAME == "")
     {
      strXNAME = p->getObjectType();
     }
     switch (anyObj.typeCode())
     {
      ...
     }
    }
    // Get Match Highlighting Information
    pMInfo = dsTS.getMatches(pCur,strDID,strXNAME,FALSE);
    strDID = "";
    strXNAME = "";
    if (pMInfo != 0)
    {
     strDoc = pMInfo->getDocumentName();
     numberSections = pMInfo->numberOfSections();
     // loop thru document sections
```

```
                for (j = 1; j <= numberSections; j++)
                {
                 pMSect = pMInfo->getSection(j);
                 strSection = pMSect->getSectionName();
                 numberParagraphs = pMSect->numberOfParagraphs();
                 // loop thru section paragraphs
                 for (k = 1; k <= numberParagraphs; k++)
                 {
                  pMPara = pMSect->getParagraph(k);
                  lCCSID = pMPara->getCCSID();
                  lLang = pMPara->getLanguageId();
                  numberTextItems = pMPara->numberOfTextItems();
                  // loop thru paragraph text items
                  for (m = 1; m <= numberTextItems; m++)
                  {
                   pMText = pMPara->getTextItem(m);
                   strText = pMText->getText();
                   // if match found in text item get offset and
                   // length of match in text item
                   if (pMText->isMatch() == TRUE)
                   {
                    lOffset = pMText->getOffset();
                    lLen = pMText->getLength();
                   }
                   numberNewLines = pMText->numberOfNewLines();
                  }
                 }
                }
                delete pMInfo;
               }
               ...
              delete item;
              }
          }
          delete pCur;
          dsTS.disconnect();
```

Result set cursor

The `dkResultSetCursor` is a datastore cursor which manages a virtual collection of DDO objects. This means that the collection does not materialize until you fetch an element from it. The collection is a resulting set of a query submitted to the datastore.

Note: When you are finished using the cursor, call the destroy method. This will automatically close the cursor, and prevent memory leaks.

Open and close the result set cursor to re-execute the query

When the result set cursor is created, it is in an open state. To re-execute the query you should close the cursor and reopen it.

```
DKString cmd = "SEARCH=(INDEX_CLASS=GP2DLS4);";
cmd +=   "OPTION=(CONTENT=YES;";
cmd +=   "TYPE_QUERY=DYNAMIC;" ;
cmd +=   "TYPE_FILTER=FOLDERDOC)";
...

dkResultSetCursor* pCur = dsDL.execute(cmd);
// re-execute the query
pCur–>close();
pCur–>open();
```

Set position and get position in a result set cursor

The result set cursor allows you to set cursor position, and get the current position. In the following example, a query is created and executed. Inside the while loop, the cursor position is set to the first (or next) valid position. Then, a DDO is fetched from the cursor position. Finally, the cursor position is retrieved, and assigned to the variable `i`. A null value is returned from `fetchObject` if the cursor is passing the last item in the result.

```
DKString cmd = "SEARCH=(INDEX_CLASS=GP2DLS4);";
cmd +=   "OPTION=(CONTENT=YES;";
cmd +=   "TYPE_QUERY=DYNAMIC;" ;
cmd +=   "TYPE_FILTER=FOLDERDOC)";
pCur = 0;
DKDDO *item = 0;
long i = 0;
...

dkResultSetCursor* pCur = dsDL.execute(cmd);
while (pCur->isValid()) {
       pCur->setToNext();
       item = pCur->fetchObject();
       if (item != 0) {
           i = pCur->getPosition();
           delete item;
       }
}
delete pCur;
```

Another way to do this is:

```
DKAny a;
pCur = dsDL.execute(cmd);
while (pCur->isValid()) {
       pCur->setPosition(DK_NEXT,a);
       item = pCur->fetchObject();
       if (item != 0) {
           i = pCur->getPosition();
           delete item;
       }
}
delete pCur;
```

You can use the RELATIVE positioning. In the example below we skip every other item in the result set cursor.

```
DKAny a;
long increment = 2;
pCur = dsDL.execute(cmd);
a = increment;
while (pCur.isValid()) {
       pCur->setPosition(DK_RELATIVE,a);
       item = pCur->fetchObject();
       if (item != 0) {
           i = pCur->getPosition();
           delete item;
      }
}
delete pCur;
```

Creating a collection from a result set cursor

The result set cursor allows you to supply a collection, to be filled in with a specified number of items from the result set cursor. Below all items are fetched from the result set cursor into the sequential collection. A zero in the first parameter of

fetchNextN indicates that all items of the cursor will be put into the collection. The first parameter specifies how many items to put into the collection. If the fItems is true, as least one item was returned.

```
DKSequentialCollection seqColl;
DKBoolean fItems = FALSE;
long how_many = 0;
fItems = pCur->fetchNextN(how_many,seqColl);
```

Retrieving a document or folder

In order to retrieve a document from DKDatastoreDL, you need to know its index-class name and item-id. You also need to associate the DDO to a datastore and establish a connection:

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVR2","userId1","passwd1");
DKPid pid;
// set the index-class name it belongs to
pid.setObjectType("GRANDPA");
// set the item-id
pid.setId("LN#U5K6ARLGM3DB4");
// create a DDO with pid and associated to dsDL
DKDDO* ddo = new DKDDO(&dsDL, pid);
// retrieve it
ddo->retrieve();
```

In case the DDO is a result of a parametric query with the query option CONTENT=NO, the DDO is empty (does not have the attribute values), but all information required to retrieve it is already set. You simply need to call:

```
ddo->retrieve();
```

After a call to retrieve, the DDO will have all of its attribute values set to the values of the persistent data stored in the datastore. If the document has parts, the DKPARTS attribute is set to a DKParts object. However, the content of each part in this collection is not retrieved yet. Since the size of a part can be big, it is not desirable to retrieve all of them into memory at once. It is better to retrieve each desired part explicitly. See “Retrieving parts” below for an example of how to retrieve a part.

Retrieving parts

Once you have retrieved a DDO, you can get its parts stored in DKPARTS attribute:

```
DKAny any = ddo->getDataByName(DKPARTS);
DKParts* parts = (DKParts*) any.value();
```

Note that the above code assumes that the DKPARTS attribute exists; an exception will be thrown if the attribute does not exist.

See “Retrieving a folder” on page 60 for another example of extracting attribute value by getting the data_id first and testing it for zero.

To retrieve each part, you need to create an iterator to iterate over the collection and retrieve each part one by one.

See also “Creating and using a DKPARTS attribute” on page 30, for more information.

```
// create iterator and process the part collection member one by one
if (parts != NULL) {
    DKAny* element;
    DKBlobDL* blob;
    dkIterator* iter = parts->createIterator();
    while (iter->more()) {
           element = iter->next();
           blob = (DKBlobDL*) element->value();
           if (blob != NULL) {
               // retrieve the blob's content
               blob->retrieve();
               // other processing, as needed
               blob->open();
            }
    }
    delete iter;
}
```

Note that similar to the DDO results of a parametric query, each part XDO inside `DKParts` collection is empty (does not have its content set) but it has all the information needed for its retrieval; this part is ready to be retrieved. You only need to call:

```
blob->retrieve();
```

This will bring its content and the balance of other information into memory.

See also “Creating and using a DKPARTS attribute” on page 30.

Retrieving a folder

Retrieving a folder DDO is the same as retrieving a document DDO. After retrieve, the folder DDO will have all of its attributes set, including a special attribute `DKFOLDER`. This attribute value is set to `DKFolder` object, a collection of DDO member of this folder. Like parts in `DKParts`, these member DDOs are relatively empty; they only contain enough information to retrieve them. You can retrieve the part at any time.

```
// get DKFOLDER data-id
data_id = ddo->dataId(DKFOLDER);
// folder not found
if (data_id == 0) {
    DKException exc(" folder data-item not found");
    DKTHROW exc;
}
// get the folder collection
any = ddo->getData(data_id);
DKFolder* fCol = (DKFolder*) any.value();
// create iterator and process the DDO collection member one by one
if (fCol != NULL) {
    DKAny* element;
    DKDDO* item;
    dkIterator* iter = fCol->createIterator();
    while (iter->more()) {
           element = iter->next();
           item = (DKDDO*) element->value();
           if (item != NULL) {
               // retrieve the member DDO
               item->retrieve();
               // other processing
               ...
```

```
            }
        }
        delete iter;
}
```

See also “Creating and using DKFOLDER attribute” on page 31.

Creating, updating, and deleting documents or folders

This section describes the processes involved in creating, updating, and deleting documents and folders.

Creating a new document

To create a new document and save its persistent data in the datastore, you need to create a DDO with all its attributes and other information set, except its item-id. The item-id will be assigned and returned by the datastore. Some of the previous examples can be combined to provide the complete steps:

```
// step 1: create a datastore and connect to it
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVR2","userId1","passwd1");

// step 2: create a document (or folder) DDO
//         and set all its attributes and other required information
//         See the section on "Using DDO"
DKPid pid;
// set the index-class name it belongs to
pid.setObjectType("GRANDPA");
// create a DDO with pid and associated to dsDL
DKDDO* ddo = new DKDDO(&dsDL,pid);

// step 2.a: add attributes according to index-class GRANDPA
DKAny any;
DKBoolean yes = TRUE;
DKBoolean no = FALSE;
// add a new attribute named "Title"
unsigned short data_id = cddo->addData("Title");
// add type property VSTRING
any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_PROPERTY_TYPE, any);
// add nullable property: non-nullable
any = no;
cddo->addDataProperty(data_id, DK_PROPERTY_NULLABLE, any);

// add a new attribute named "Subject"
data_id = cddo->addData("Subject");

any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_PROPERTY_TYPE, any);
any = yes;
cddo->addDataProperty(data_id, DK_PROPERTY_NULLABLE, any);

// add some more attributes as necessary
//  ...

// step 2.b: add DKPARTS attribute
// create a new XDO blob
DKParts* parts = new DKParts;
DKBlobDL* blob = new DKBlobDL(&dsDL);

DKPidXDODL pidXDO;                              // create Pid for this XDO object
```

```
pidXDO.setPartId(5);                              // set part number to 5
blob->setPid(&pidXDO);                           // set the Pid for the XDO blob
blob->setContentClass(DK_CC_GIF);                // set content class type GIF
blob->setRepType(DK_REP_NULL);                   // set rep type for the part
blob->setContentFromClientFile("choice.gif");    // set the blob's content

DKAny any = (dkDataObjectBase*) blob;
parts->addElement(any);                          // add the blob to the parts collection

...                                               // create and add some more blobs
...                                               // to the collection as necessary

// create DKPARTS attribute and sets it to refer to the DKParts object
// add attribute "DKParts"
unsigned short data_id = ddo->addData(DKPARTS);
any = DK_COLLECTION_XDO;
// add type property
ddo->addDataProperty(data_id,DK_PROPERTY_TYPE,any);
any = (DKBoolean) TRUE;
// add nullable property
ddo->addDataProperty(data_id,DK_PROPERTY_NULLABLE,any);
any = (dkCollection*) parts;
// sets the attribute value
ddo->setData(data_id, any);

// step 2.c: sets the item type : document
any = DK_DOCUMENT;
ddo->addProperty(DK_PROPERTY_ITEM_TYPE, any);

// step 3: make it persistent
// a document is created in the datastore
ddo->add();
```

After the last step, you will have a new document with the above information created in the datastore. When a document DDO is added to a datastore, all its attributes are being added, including all parts inside the `DKParts` collection. The same rule applies to adding a folder DDO; the `DKFolder` collection members are added to the datastore as a Digital Library folder.

Recall that a Digital Library folder actually contains a table-of-contents of its members, which are existing documents and folders. Therefore, all folder members must be created first in the datastore before you can add a folder DDO.

You can add the same document to a different datastore of the same type, provided that you adjust the information accordingly. To add this document to the server `LIBSRVRN`, which has an index-class `GRANDPA2` with the same structure as `GRANDPA`, you could do the following steps:

```
// create datastore and connect to LIBSRVRN
DKDatastoreDL dsN;
dsN.connect("LIBSRVRN","userIdN","passwdN");
// update the Pid
pid = ddo->getPid();
pid.setObjectType("GRANDPA2");              // set the new index-class
pid.setId("");                              // blank the item-id
pid.setDatastoreName("LIBSRVRN");           // set the new datastore name
ddo->setPid(pid);                           // update the Pid
ddo->setDatastore(&dsN);                    // re-associate it with dsN
ddo->add();                                 // add it
```

Updating a document or a folder

Updating a document is quite simple. First, you need to have the item-id and object-type set. Then, update the desired attributes; for example, set another value for the attribute or add more parts to DKParts collection. Finally, call the update method to have the update reflected in the datastore, for example:

```
// update the value of attribute Title
DKAny any = DKString("Guess who is behind all this");
unsigned short data_id = ddo->getDataByName("Title");
ddo->setData(data_id, any);
ddo->update();
```

After the call to update, the value of the attribute ″Title″ in the datastore will be updated. The parts in this document will not be updated unless their content has changed. The associated datastore and its connection must be valid when you call the update method.

Updating a folder DDO requires similar steps. You need to update some attribute values, or add or remove some elements from DKFolder, then call the update method.

Updating parts

The collection of parts in a document is represented as a DKParts object. DKParts is a subclass of DKSequentialCollection, but in addition to inheriting the sequential collection functions, it has two additional members to add and remove a part from the collection and have it reflected in the datastore immediately. The document must already exist in the datastore.

Add and remove member

Given a document DDO, you could use the following code fragments to add a part to this document immediately:

```
// a document DDO
DKDDO* ddo;
// a new part to be added
DKBlobDL* newPart;
// ddo and newPart are
// initialized somewhere along the line
...
...
// get DKParts
DKAny any = ddo->getDataByName(DKPARTS);
DKParts* parts = (DKParts*) any.value();
// assume none of these values are NULL
parts->addMember(ddo, newPart);
```

The last statement gives the same effect as:

```
// add the part to datastore, into the
// specified document
newPart->add();

DKAny any = (dkDataObjectBase*) newpart;
// add newPart into this collection
parts->addElement(any);
```

The newPart must have its Pid set to make it work.

Similarly, to remove newPart from the collection and the datastore, you would do the following:

```
parts->removeMember(ddo, newPart);
```

which is equivalent to:

```
DKSequentialIterator* iter = (DKSequentialIterator*)
parts->createIterator();
...
DKAny* pany = iter->at();
newPart = (DKBlobDL*) pany->value();
// assume newPart is not NULL
parts->removeElement(iter);          // remove the part pointed by iter
newPart->del();                      // delete the part from this document
                                     // the copy in memory unchanged
```

To compare with the same method in DKFolder, the removeMember() method in DKParts actually deletes the persistent copy of the part in the datastore. In DKFolder, this method only removes the item from the folder table-of-contents; it does not actually delete the item.

See “Updating folders”.

Differences with update() on a document DDO

Add and remove member methods on DKParts provide conveniences for adding and removing a part in the collection and the datastore. Compared to the update() method in a document DDO, the add and remove members are faster. The update() method on a DDO will update all of the attributes in the DDO including DKParts and all of its members that changed. So, the steps would be:

```
...
DKAny any = ddo->getDataByName(DKPARTS);
// get DKParts, assume it exists
DKParts* parts = (DKParts*) any.value();
// assume it is not NULL
any = (dkDataObjectBase*) newpart;
parts->addElement(any);
// updates the whole ddo
ddo->update();
...
```

Updating folders

The collection of documents and folders in a Digital Library folder is represented as a DKFolder object. In the datastore, a folder holds a table of contents referring to its object members, instead of keeping all actual objects.

DKFolder is a subclass of DKSequentialCollection, but in addition to inheriting the sequential collection functions, it has two additional members to add and remove a member from the collection, be it a document or a folder, and have it reflected in the datastore immediately. The removed or added document or folder must already exist in the datastore.

Add and remove a member

Given a folder DDO, you could use the following code fragments to add another document or folder DDO to it:

```
// a folder DDO
DKDDO* folderDDO;
// a new DDO to be added as a member of this folder
DKDDO* newMember;
...      // folderDDO and newMember are
...      // initialized somewhere along the line
DKAny any = folderDDO->getDataByName(DKFOLDER);
// get DKFolder, assume it exists
DKFolder* folder = (DKFolder*) any.value();
// assume non NULL
folder->addMember(folderDDO, newMember);
```

The `newMember` and `folderDDO` must exist in the datastore to allow this to work.

The last statement gives the same effect as:

```
any = (dkDataObjectBase*) newMember;
// add newMember to this collection
folder->addElement(newMember);
// reflects it in the datastore
folderDDO->update();
```

Similarly, to remove `newMember` from the collection and the datastore, you would do the following:

```
folder->removeMember(folderDDO, newMember);
```

This is equivalent to the following, which has more steps:

```
DKSequentialIterator* iter = (DKSequentialIterator*)
folder->createIterator();
...
newMember = (DKDDO*) iter->at();
// remove newMember from this collection
folder->removeElementAt(iter);
// reflects it in the datastore
folderDDO->update();
...
// if no longer needed
delete newMember;
```

Note that removing a member from the folder only removes that member from the folder table of contents. It does not delete the member itself from memory or the datastore.

Differences with update() on a folder DDO

Add and remove member methods on `DKFolder` provide conveniences for adding and removing a document or folder in the collection and in the datastore. Compared to `update()` method in a folder DDO, add and remove members are faster. The `update()` method on a DDO will update all of the attributes in the DDO, including `DKFolder` and all of its members (that is, the folder table of contents).

Add and remove member operations only involve adding or removing one particular member to or from the folder table of contents.

Deleting a document or a folder

You only need to call `del()` method in the DDO to delete the persistent data from the datastore:

```
ddo->del();
```

The DDO must have its item-id and object-type set, and a valid connection to a datastore. The above statement can be used to delete a folder also. Only the persistent data is deleted; the in-memory copy of the DDO does not change. Therefore, you can add this DDO back to the same or different datastore later.

See “Creating a new document” on page 61; see also “Deleting a DDO” on page 32, to understand the difference between these two operations.

Queryable collection

A *queryable collection* is a collection that can be queried further, thereby providing a set of more refined results. A concrete implementation of a queryable collection is the `DKResults` class. `DKResults` is a collection of DDO, the results of query evaluation.

Getting the result of a query

The following example illustrates how to submit a parametric query and get results:

```
// establish a connection
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVR2","userId1","passwd1");
// create a query object
DKString query1 =  "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> NULL));";
DKParametricQuery* pq =(DKParametricQuery*)
dsDL.createQuery(query1,DK_PARAMETRIC_QL_TYPE, NULL);
pq->execute();
DKAny any = pq->result();
DKResult* rs = (DKResults*) any.value();
```

The results is in `rs` object, which is an instance of `DKResults`, a subclass of `dkQueryableCollection`. You can use the previous code examples to process the collection and get the DDO out.

See “Collections and iterators” on page 21 for more information.

Evaluating a new query

Given a result from the above query, you can submit another query to this result, as to refine the current result set to get a smaller set. Its concept is similar to using the current result as a scope for the new query. Here are the steps to do it:

```
DKString query2 = "SEARCH=(INDEX_CLASS=GRANDPA, COND=(Subject == 'Mystery'));";
         any = rs->evaluate(query2,DK_PARAMETRIC_QL_TYPE, NULL);
         ...
```

`any` will hold a `DKResults` object containing the refined results. The totals of both queries would be equivalent to:

```
"SEARCH=(INDEX_CLASS=GRANDPA, COND=(Title <> NULL AND Subject == 'Mystery'));"
```

You can repeat this step over and over to reach a satisfactory end result. Once you have started with a parametric query, all the subsequent queries must be of the same type, otherwise a null or unpredictable result will probably return.

The same procedure will work for text queries also. The following is an example for text queries:

```
DKDatastoreTS dsTS;
dsTS.connect("barney","6011",DK_CTYP_TCPIP);

DKString tquery1 = "SEARCH=(COND=(IBM)); OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery*  tq = (DKTextQuery*) dsTS.createQuery(tquery1,DK_TEXT_QL_TYPE, NULL);
tq->execute();
any = tq->result();
DKResults* trs = (DKResults*) any.value();

DKString tquery2 = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
any = trs->evaluate(tquery2,DK_TEXT_QL_TYPE, NULL);
```

any will hold a DKResults object containing the results of the totals of both queries:

```
"SEARCH=(COND=(IBM AND Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
```

Using queryable collection instead of combined query

Evaluating a queryable collection bears some similarities to some of the other features in this class library. One common feature is Combined Query. The combined query provides the flexibility to submit a combination of parametric and text queries, with or without scopes. However, all of these queries must be submitted at once, not one by one (like evaluating a queryable collection). See “Combined query” for more information.

The result of a combined query is a DKResults, so theoretically you can evaluate another parametric query against it.

Programming tips

Evaluating queryable collection with subsequent queries provide a flexibility to refine the results of a previous query, step by step, until you get a satisfactory final result. This is useful for browsing the content of datastore dynamically, formulating the next query depending upon the previous results. However, if you know the total query in advance, it would be more efficient to submit the entire query once, instead of in incremental fashion.

Combined query

As suggested by its name, a combined query allows you to execute a combination of parametric and text queries, with or without a scope. The final result is intersections between the scopes and the results of each query. Therefore, if you are not careful in formulating the query and including scopes, these collections may not intersect at all, and you would end up with an empty result.

If there is at least one parametric and one text query, the resulting DDO returned in DKResults will have an attribute DKRANK, which signifies the highest rank of the matching part belonging to this document.

Important: For each query in a combined query, you must use a dedicated connection to the search engine; you cannot route multiple queries through the same connection.

Combined parametric and text query

To execute a combined query with one parametric and one text query (without any scope), you need to create the combined query object, the parametric and text queries, and pass the latter two queries as input parameters to be executed by the combined query. Here is an example:

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVR2","userId1","passwd1");

DKDatastoreTS dsTS;
// TM is a local alias for the Text Search Engine server
dsTS.connect("TM","",' ');
// create a parametric query
DKString pquery =  "SEARCH=(INDEX_CLASS=GRANDPA,COND=(DLSEARCH_Date > 1994));";
DKParametricQuery* pq =
       (DKParametricQuery*) dsDL.createQuery(pquery,DK_PARAMETRIC_QL_TYPE, NULL);

// create a text query
DKString tquery = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery* tq =
       (DKTextQuery*) dsTS.createQuery(tquery,DK_TEXT_QL_TYPE, NULL);

// create a combined query
DKCombinedQuery* cq = new DKCombinedQuery();

// package the queries in DKNVPair as input parameters
DKNVPair par[3];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
// to signal the end of parameter list
par[2].setName(DK_PARM_END);

// execute the combined query
cq->execute(par);

// get the results
DKAny any = cq->result();
DKResults* res = (DKResults*) any.value();
if (res != NULL) {
    // process the results
    ...
}
```

The last if statement is necessary to make sure the `DKResults` object is not null before calling its method.

See examples under “Collections and iterators” on page 21, and “Querying the datastore” on page 49.

Using a scope

If you happen to have a `DKResults` object you want to use as the scope, you can modify the above code slightly to insert the scope as an additional parameter:

```
DKResults* scope;    // assume that this is the scope
                     // initialized somewhere as a result of
                     // some parametric query
DKResults* tscope    // assume that this is the scope
                     // initialized somewhere as a result of
                     // some text query


...
```

```
// package the query in DKNVPair as input parameters
DKNVPair par[4];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].set(DK_SCOPE_DL, scope);
par[3].set(DK_SCOPE_TS, tscope);
par[4].setName(DK_PARM_END);
// execute the combined query
cq—>execute(par);
...
```

The results of a combined query can also be used as a scope of another combined query, and sometimes the results are queryable.

Ranking

If the combined query contains at least one text query, then the DDO in the result will have the attribute `DKRANKS`. This is a transient attribute; it will not be stored in the datastore but will be computed each time by the text search engine. The value of the rank corresponds to the highest rank of the part in the document which satisfies the text query conditions.

Programming tips

The same observation as in queryable collection also applies here. If you have several parametric queries and scopes, it is more efficient to execute the total query as a single parametric query. This is also true for text queries. The most efficient execution would be if you have one total parametric query and one total text query, without scope. See also “Queryable collection” on page 66, for more information.

The query option `MAX_RESULTS=nn` limits the number of results returned to the caller. Usually, this option is more applicable to text queries, since the result is sorted by rank in descending order. Thus, if this option is set to `10`, for example, it means that the caller only wants the ten highest matching rank of the results.

The meaning of this option is different for parametric queries. Since there is no notion of rank, the caller will get the first ten results. These results will be intersected with the result from the text query. Therefore, when combining a parametric query and text query, it is advisable not to specify this option for the parametric query. Otherwise, the result would be less meaningful.

Text Search Engine

Text Search Engine is the product name of the text search engine. Text Search Engine allows you to specify boolean, proximity, GTR (Global Text Retrieval), hybrid and free text queries. The query results returned from Text Search Engine contain the Digital Library item ID, part number and ranking information. This information can be used to create a Digital Library XDO that can be used to retrieve the text document contents in Digital Library.

Programming tips

To prevent the timing out of a Text Search Engine connection, it is best to perform a connect, execute your queries and then disconnect, as opposed to leaving a connection open for long period of time.

The search options CCSID and LANG work together; if one is specified, you should also specify the other. The default CCSID and LANG are specified by the datastore text search options (DK_OPT_TS_CCSID and DK_OPT_TS_LANG).

You can specify more than one search option for a query term. The search options are separated by commas. An example of multiple search terms is specified in “GTR query” on page 71.

If you wish to specify both the SC and the MC search option, the SC option must occur first. An example could be ($SC=?,MC=*$ U?I*).

For detailed descriptions of the SC (single required character) and the MC (a sequence of optional characters) search options, refer to the *Application Programming Reference*.

Boolean query

A boolean query is made up of words and phrases, separated by a boolean operator. A phrase is a sequence of words enclosed in single quotes, to be searched as a literal. In the example below we are searching for all text documents with the word WWW or the phrase Web site in the TMINDEX text search index:

```
DKString cmd  = "SEARCH=(COND=(WWW OR 'Web site'));";
         cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Free text query

A free text query is made up of words, phrases or sentences enclosed in braces {}. The words do not need to be adjacent to each other. In the example below we are searching for all text documents with the free text Web site in the TMINDEX text search index:

```
DKString cmd  = "SEARCH=(COND=({Web site}));";
         cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Hybrid query

A hybrid query is made up of a boolean query, followed by a free text query. In the example below we are searching for all text documents with the words IBM and UNIX, as well as the free text Web site in the TMINDEX text search index:

```
DKString cmd  = "SEARCH=(COND=(IBM AND UNIX {Web site}));";
         cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Proximity query

A proximity query relates to a sequence of search arguments found in the same document, paragraph or sentence. In the example below we are searching for all text documents with the phrase rational numbers and the word math in the same paragraph.

```
DKString cmd  = "SEARCH=(COND=($PARA$ {'rational numbers' math}));";
         cmd += "OPTION=(SEARCH_INDEX=TMINDEX)"
```

Note: this type of query requires at least two search arguments.

GTR query

A GTR (Global Text Retrieval) query is optimized for the double-byte character set (DBCS) languages like Japanese or Chinese, but also supports the single-byte character set (SBCS) languages.

Make sure that the text search datastore options DK_OPT_TS_CCSID (coded character set identifiers) and DK_OPT_TS_LANG (language identifiers) are set properly. The default for DK_OPT_TS_CCSID is DK_CCSID_00850. The default for DK_OPT_TS_LANG is DK_LANG_ENU. These values are used as the global defaults for the text query. You can also enter specific CCSID and LANG information, as shown in the following example; see "Programming tips" on page 69 for further information.

The example below illustrates searching of a GTR index for all text documents with the phrase IBM marketing. *All double-byte characters should be enclosed in single quotation marks (')*. We have specified the character code set and language of the IBM marketing term. Also, we have specified the match keyword to indicate the degree of similarity for the IBM marketing search term:

```
DKString cmd = "SEARCH=(COND=($CCSID=850, LANG=6011,MATCH=1$ ";
         cmd += "'IBM marketing'));";
         cmd += "OPTION=(SEARCH_INDEX=TMGTRX)";
```

Loading data to be indexed by Text Search Engine

In order to load data into Digital Library to be indexed by Text Search Engine, a Digital Library index class must be created, as well as a Text Search Engine index.

Before creating a Text Search Engine index, the Text Search Engine server must be running. Make sure your environment is setup properly by running the Digital Library samples TListCatalogDL and TListCatalogTS, after the samples have been updated with your server, user ID, etcetera.

To create parts in Digital Library that are indexed by Text Search Engine, refer to the section "Using XDO" on page 32.

Once the data is loaded into Digital Library, the documents are placed in the document queue. Use a method in the DKDatastoreDL called wakeUpService, which takes a search engine name (SM). Once this is done, you launch the Digital Library Text Search Engine administration panel, double-click on the Text Search Engine server, then double-click on the Text Search Engine index and single click the INDEX button on the notebook page. This action will index the documents on the document query. Once the indexing is complete, you can perform queries against Text Search Engine .

Environment settings

On AIX

Your PATH directories

- set NLS path for the message file:

  ```
  export NLSPATH=${NLSPATH}:/usr/lpp/frn/msg/En_US/%N
  ```

- set PATH to:

```
export PATH=/usr/lpp/frn/lib
```

Your LIBPATH directories

- set LIBPATH to:

```
export LIBPATH=/usr/lpp/frn/lib
```

Your INCLUDE directories

- set INCLUDE to:

```
export INCLUDE=/usr/lpp/frn/INCLUDE
```

On Windows NT

Your PATH directories: x - drive

- set PATH to:

```
set PATH=x:\FRNROOT\DLL
```

Your INCLUDE directories: x - drive

- set INCLUDE to:

```
set INCLUDE=x:\FRNROOT\INCLUDE
```

Building C++ programs on AIX

Use the `-qalign=packed` compiler option so that the object will align properly. Refer to the sample makefiles in the samples directory for more information.

Building C++ programs on Windows NT

New project and makefile, Microsoft Developers Studio:

1. Create a new project. (console application)
2. Select the **Insert** menu choice and insert. Insert the .cpp file into the project.
3. Select the **Build** menu choice and settings, Microsoft Foundation Classes under the **General** tab.
4. Select **Use MFC in a Shared DLL**.
5. Select the **C/C++** tab, type in under additional include directories where Development Kit - OO API include directories are located.
6. Select the **Link** tab, type in under Object/library modules where Development Kit - OO API lib (dkmm.lib) is located. Use the dkmm.lib file only if you are building in release mode; use dkmmd.lib is you are building in debug mode.
7. You can now build your project and run it

Use existing makefile, Microsoft Developers Studio:

1. Open the workspace and select the sample makefile.
2. Select the **Build** menu item and view the settings.
3. Select the **C/C++** tab, type in under the additionally included directories where the Development Kit - OO API include directories are located.
4. Select the **Link** tab, type in under the Object/library modules where the Development Kit - OO API lib (dkmm.lib) is located. Use the dkmm.lib file only if you are building in release mode; use dkmmd.lib is you are building in debug mode.
5. You can now build your project and run it.

Setting console sub system for code page conversion on Windows NT

```
#include <DKConstant.h>
#include <DKEnvironment.hpp>

void main(int argc, char *argv[])
{
   // set sub system to console at the beginning of program this
   // will cause the code page that the error messages are returned
   // in by DKExceptions to be converted from the Windows Graphical
   // User Interface (ANSI format) to the Console (OEM format)
   // If this is not specified the default is DK_SS_WINDOWS
   DKEnvironment::setSubSystem(DK_SS_CONSOLE);
   ...
}
```

Understanding the workflow and workbasket functions

This section describes the workflow and workbasket functions.

Understanding the workflow service

The C++ APIs provide the workflow service for the Digital Library datastore, encompassing the workflow and workbasket functions that are available in the Folder Manager. A workbasket is a container that holds documents and folders that you want to process. A workflow is an ordered set of workbaskets that represent a specific business process. Folders and documents move between workbaskets within a workflow, allowing your applications to create simple business models and to route work through the process until completion.

The workflow model used in the Folder Manager follows these rules:

- A workbasket does not need to be located in a workflow
- A workbasket can be located in one or more workflows
- A workbasket can be located in the same workflow more than once
- A document or folder can only be stored in one workflow at a time; however, workbaskets can be located in multiple workflows
- A document or folder can be stored in a workbasket that is not located in a workflow

The DKWorkFlowServiceDL class represents the workflow service of the Digital Library datastore. This class provides the capability to start, change, remove, route, and complete a document or folder in a workflow. Additionally, the DKWorkFlowServiceDL class allows you to retrieve information about workbaskets and workflows in a Digital Library datastore. In Digital Library, the DKWorkFlowDL and DKWorkBasket classes are the object-oriented representations of a workflow item and a workbasket item, respectively.

Establishing a connection

You must establish a connection to a Digital Library datstore before you can use the workflow service. The Digital Library datastore provides connect and disconnect methods. The following connection example would connect to a Digital Library datastore named `LSMUFASA`, using the user ID `FRNADMIN` and the password `PASSWORD`. The complete sample of datastore connection, TListWorkFlowWFS.cpp, is available in the samples directory.

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
...                                  // do some work
dsDL.disconnect();
```

The connect method allows an application to connect to a Digital Library datastore. After a workflow service is created, subsequent methods of the workflow service can be issued.

List workflows

The workflow service provides a method that lists the workflows in the system. The following example demonstrates the retrieval of the list of workflows; this list is returned in a sequential collection of DKWorkFlowDL objects. The complete sample of this function, TListWorkFlowsWFS.cpp, is available in the samples directory.

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1 = (DKSequentialCollection *)wfDL.listWorkFlows();
if (wfList1 != NULL)
   {
      dkIterator * pIter1 = wfList1->createIterator();
      DKWorkFlowDL * pwf1;
      while (pIter->more())
      {
         pwf1 = (DKWorkFlowDL *)((void*)(*pIter1->next()));
         pwf1->retrieve();
         ...                         // do some work
         delete pwf1;
      }
   }
dsDL.disconnect();
```

List workbaskets

The workflow service provides a method that lists the workbaskets in the system. The following example demonstrates the retrieval of the list of workbaskets; this list is returned in a sequential collection of DKWorkBasketDL objects. The complete sample of this function, TListWorkBasketsWFS.cpp, is available in the samples directory.

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1=(DKSequentialCollection *)wfDL.listWorkBaskets();
if (wbList1 != NULL)
   {
      dkIterator * pIter1 = wbList1->createIterator();
      DKWorkBasketDL * pwb1;
      while (pIter->more())
      {
         pwb1 = (DKWorkBasketDL *)((void*)(*pIter1->next()));
         pwb1->retrieve();
         ...                         // do some work
         delete pwb1;
      }
   }
dsDL.disconnect();
```

List items in a workflow

The workflow service provides a method that lists the item IDs of the items in a workflow. The following example demonstrates the retrieval of the list of item IDs for the items in a particular workflow; this list is returned in a sequential collection of DKString objects. The complete sample of this function, TListItemsWFS.cpp, is available in the samples directory. The workflow name HI7MOPALUPFQ1U47 is fictitious; you must use a valid workflow ID instead of the workflow name in the example.

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
DKWorkFlowDL * wf = new DKWorkFlowDL(&wfDL, (char *)itemIDWF);
wf->retrieve;
DKSequentialCollection * pColDoc1 = (DKSequentialCollection *)wf->listItemIDs();
if (pColDoc1 != NULL)
   {
      dkIterator* pIterDoc1 = pColDoc1->createIterator();
      DKString DocID1;
      while (pIterDoc1->more() == TRUE)
      {
         DocID1 = (DKString)(*pIterDoc1->next()));
         ...                        // do some work
      }
   }
dsDL.disconnect();
```

Executing a workflow

The workflow service provides methods that allow you to execute a workflow. The following example demonstrates how to start an item in a workflow, how to route an item to a workbasket, and how to complete an item in a workflow. The complete sample of this function, TProcessWFS.cpp, is available in the samples directory. You must modify the sample program to reflect these changes:

- Use a valid item ID instead of EP8L8OR9MHH##QES
- Use a valid workflow ID instead of HI7MOPALUPFQ1U47
- Use a valid workbasket ID instead of E3PP1UZOZUFQ1U3M

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
DKString itemID   = DKString("EP8L8OR9MHH##QES");
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
DKString itemIDWB = DKString("E3PP1UZOZUFQ1U3M");
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
wfDL.startWorkFlowItem(itemID,                  // itemID
                       itemIDWF,                // itemIDWB
                       NULL,                    // default(the first workbasket)
                       TRUE,                    // overload
                       DK_WIP_DEFAULT_PRIORITY  // initial_priority
                      );
...                                        // do some work
wfDL.routeWipItem(itemID,                  // itemID
                  itemIDWF,                // itemIDWB
                  TRUE,                    // overload
                  DK_NO_PRIORITY_CHANGE    // initial_priority
                      );
...                                        // do some work
wfDL.completeWorkFlowItem(itemID);
dsDL.disconnect();
```

Creating a workflow

The workflow service allows you to create a new workflow. The typical workflow creation process follows these steps:

1. Create an instance of DKWorkFlowDL
2. Set the workflow name to `GOLF`
3. Set the workbasket sequence to NULL to indicate that this workflow contains no workbaskets
4. Set the privilege to **All Privileges**
5. Set the disposition to DK_WF_SAVE_HISTORY
6. Call the add method

The following example uses these steps to create a workflow. The complete sample of this function, TCreateDelWorkFlow.cpp, is available in the samples directory. If you connect to the datastore as a normal user (DK_SS_NORMAL), you will not get the workflow that is defined after you connect. Therefore, this sample uses DK_SS_CONFIG.

```
DKDatastoreDL dsDL;
DKAny input_option = DK_SS_CONFIG;
DKWorkFlowServiceDL wfDL(&dsDL);
dsDL.setOption(DK_OPT_DL_ACCESS, input_option);
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
DKWorkFlowDL * newwf = new DKWorkFlowDL(&wfDL);
newwf->setName("GOLF");
newwf->setWorkBasketSequence((dkCollection *)NULL);
newwf->setAccessList("All Privileges");
newwf->setHistoryDisposition(DK_WF_SAVE_HISTORY);
newwf->add();
...                                 // do some work
dsDL.disconnect();
```

Creating a workbasket

The workflow service allows you to create a new workbasket. The typical workbasket creation process follows these steps:

1. Create an instance of DKWorkBasketDL
2. Set the workbasket name to `Hot Items`
3. Set the privilege to **All Privileges**
4. Call the add method

The following example uses these steps to create a workflow. The complete sample of this function, TCreateDelWorkBasket.cpp, is available in the samples directory. If you connect to the datastore as a normal user (DK_SS_NORMAL), you will not get the workflow that is defined after you connect. Therefore, this sample uses DK_SS_CONFIG.

```
DKDatastoreDL dsDL;
DKAny input_option = DK_SS_CONFIG;
DKWorkFlowServiceDL wfDL(&dsDL);
dsDL.setOption(DK_OPT_DL_ACCESS, input_option);
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
DKWorkBasketDL * newwb = new DKWorkBasketDL(&wfDL);
newwb->setName("Hot Items");
newwb->setAccessList("All Privileges");
newwb->add();
...                                 // do some work
dsDL.disconnect();
```

Chapter 4. Using the Java application programming interface

The Java application programming interface (API) is a set of Java classes that provides access and manipulation of local or remote data stored in the Digital Library storage facilities.

This document describes the Java application programming interface, the Java implementation of Digital Library API multi-search facilities, and Internet connectivity. It uses the Java Class Library API derived from OMG/Object Query Services (OQS) and Dynamic Data Object (DDO) protocol, which is a part of OMG/Persistence Object Services.

The Java API supports:

- Multi-search and update across a combination of heterogeneous datastores
- A common-object model for Digital Library data access
- A flexible mechanism to use a combination of different search-engines; for example, Text Search Engine text search and QBIC image search
- Client/server implementation for Java application users

The C++ API and Java API chapters contain similar information placed within a consistent structure; this consistency between chapters unites the information contained in the C++ and Java API sets. The consistent structure also helps application developers as they coordinate the development of their applications, regardless of the programming language they choose.

General Handling Information

This section includes the information about handling large objects and exceptions; in the previous version of this book, the Text Search Engine section of this chapter contained the handling information.

Handling large objects

This section describes the ways in which Digital Library handles large objects.

What is MAXPIECE?

MAXPIECE is an environment variable for Digital Library to handle a large object. This variable defines, in megabytes, the largest object to be processed as a whole. When MAXPIECE is set, an object larger than it is stored as a sequence of objects whose size is equal to or less than MAXPIECE megabytes. If MAXPIECE is not set, the object is treated as a single large object.

How to set the MAXPIECE environment variable

In Windows NT, go to **Start** → **Setting** → **Control Panel** → **System** → **Environment** → **System Variables** or **User Variables**, enter `MAXPIECE` in the entry field and a number (for example, `4`) in the value field, then click **Set** → **OK**. This will set `MAXPIECE` to `4` megabytes.

In AIX, type: `export MAXPIECE=4` , then press the Enter key. This will set `MAXPIECE` to `4` megabytes.

Setting Java heap size

The current Java system has a limitation for the default initial and maximum heap size. The default initial heap size is 1048576 and the default maximum heap size is 16777216. If your Java application program uses objects bigger than the default maximum heap size, your program will fail during execution. To increase maximum heap size for your application, use the -mx option when you execute your Java application program. The following is an example to increase the default maximum heap size:

```
java -mx40000000 yourApplication
```

How to catch a DKException

A DKException, once caught, allows users to see any error messages, error codes, and error states that occurred while running. If an error is caught below the DKException name, an error is issued along with the location of where the exception was thrown. The error ID and exception ID are also given.

```
try {
     DKDatastoreDL dsDL = new DKDatastoreDL();
     dsDL.connect("LIBSRVRN","USER1","PASSWORD","");
     dsDL.disconnect();
}
catch (DKException exc) {
   System.out.println("Exception name " + exc.name());
   System.out.println("Exception message " + exc.getMessage());
   exc.printStackTrace();
}
```

Tracing Information

You can use any of the following environment variable settings to get tracing information.

For Text Queries Using Text Search Engine

- DKTMDSTREAMTRACE=<fileName> (for example, <.\tm.out> for Windows NT or <./tm.out> for AIX)

  This environment variable setting puts the Text Search Engine query datastream, in binary format, into the specified file.
- DKTMTRACE=<fileName> (for example, <.\tm.api> for Windows NT or <./tm.api> for AIX)

  This environment variable setting puts some of the Text Search Engine API calls used during a text query into the specified file.

For Parametric Queries Against Digital Library

- DKDLQRYTRACE=<fileName> (for example, <.\dlqry.out> for Windows NT or <./dlqry.out> for AIX)

  This environment variable setting puts the parametric query passed to the Folder Manager into the specified file.

Multi-Search facilities

Use the multi-search facilities to:

- Search within a given datastore, using one or a combination of supported query types:

Parametric query
Queries requiring an exact match on the condition specified in the query predicate and the data values stored in the datastore.

Text query
Queries on the content of text fields for approximate match with the given text search expression; for example, the existence (or nonexistence) of certain phrases or word-stems.

Each search type is supported by one or more search-engines.

- Search on the results of previous search.

Client/Server architecture

The Digital Library Java API provides a convenient application programming interface for the Java application users. The applications can be located at local or remote sites. Java APIs will reside on both the server and the client sides; both sides provide exactly the same interface. The client side of Java API communicates with the server side to access data in Digital Library through the network. Communication between the client and the server is performed by Java classes; it is not necessary to add any additional programs.

Java API classes consist of three groups: server, client and common. The client and server packages provide the same API, but have different implementations.

- The server classes are related mainly to Digital Library and connect directly with it
- The client classes are not directly connected to Digital Library; these client classes communicate with the server classes through the network by invoking the server classes to execute and retrieve the results
- The common classes are shared by both the client and server sides

Packaging for the Java environment

Java provides the concept of ″packaging″, which is a grouping of classes. Typically, packages are grouped by strong interrelationships. Class members that are not explicitly declared as public, private, or protected can only be seen by classes in the same package. The package environment provides a convenient means to associate the classes within the package group. The compiler creates a directory structure that matches the package names. In the Digital Library, both packages and directories of the packages will be defined.

Package hierarchy

The following syntax diagram shows the package structure:

```
COM --- ibm --- mm ---- sdk ---- server
                     |       |
                     |       |--- client
                     +--- .....
```

The Digital Library package hierarchy starts with `COM` followed by `ibm` to distinguish it from the products of other companies. The subdirectory `mm` under the `ibm` directory indicates that these classes are for handling multimedia data, which is the main form of content of Digital Library repositories. The subdirectory `sdk` under the `mm`

directory indicates that this is the software developer kit's package. Currently there are three packages: common, server, and client. The server package is implemented for access and manipulation of Digital Library information. The client package is implemented for communication with the server package, network control, and data transformation between the server and client sites. You must install one of these packages; the server package for local applications, or the client package for applications that access the remote server.

Differences from the C++ application programming interface

- Java does not support multiple inheritance; it replaces this with interfaces. Some of the classes in the Java API are therefore defined as an interface. The interface can simulate multiple inheritance.
- Java does not support operator overloading, so operators defined in the C++ API cannot be defined in the Java API. Therefore, the C++ operators that are necessary are supported as Java functions.
- Java system provides data types defined as Java objects, such as **String**. These data types are subclasses of the Java basic class Object. These objects can provide a mechanism to deal with dynamic data types, where the precise type of data is known only at run-time. Therefore, the Java class Object is used in place of the C++ class DKAny to represent a generic object.
- Java does not support header files. Java uses interfaces instead of header files. For example, the interface **DKConstant** replaces the header file where common and global constants are defined.
- Java Virtual Machine provides memory management in the form of an automatic garbage collector. This garbage collector keeps track of all objects and references to those objects in Java programs. Therefore, there is no destructor or memory de-allocator in the Java API.

Understanding Datastore DL

A DKDatastoreDL (datastore DL) represents and manages a connection to a Digital Library datastore; it also provides transaction support and executes datastore commands.

Establishing a connection

The datastore DL provides a method for connect as well as a method for disconnect. The following example shows how to connect to the Digital Library library server named LIBSRVRN, using the user ID USER1 and password PASSWORD. Normally, you would create a datastore DL, connect to it, do some work, then disconnect when done.

A complete sample, TConnectDL.java, is available in the samples directory.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","USER1","PASSWORD","");
....                        // do some works
dsDL.disconnect();
```

Setting and getting datastore options

The datastore provides some processing or informational options that you can set or get using its methods. The following example shows how to set and get the

option for establishing an administrative session. See the *Application Programming Reference* for the list of options and their descriptions.

```
Integer input_option = new Integer(DK_SS_CONFIG);
Integer output_option = null;
dsDL.setOption(DK_OPT_DL_ACCESS,input_option);
output_option = (Integer)dsDL.getOption(DK_OPT_DL_ACCESS);
```

The option usually takes an Integer value, but it could be of any object.

List servers

The datastore provides a method to list the library servers that it can connect to. The following example shows how to retrieve the list of servers. The list of servers are returned in a DKSequentialCollection of DKServerInfoDL objects. Once a DKServerInfoDL object is obtained you can pull out the server name and server type, and use the server name to establish a connection to it.

The complete sample, TListCatalogDL.java, is available in the samples directory.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKServerInfoDL pSV = null;
String strServerName = null;
String strServerType = null;
DKSequentialCollection pCol = (DKSequentialCollection) dsDL.listServers();
dkIterator pIter = pCol.createIterator();
while (pIter.more()) {
       pSV = (DKServerInfoDL) pIter.next();
       strServerName = pSV.serverName();
       strServerType = pSV.serverType();
       System.out.println(" Server Name : " + strServerName);
       System.out.println(" Server Type : " + strServerType);
      }
```

List schema and schema attributes

The datastore provides methods for listing the schema. In the Digital Library datastore, these are index classes and their attributes. The following example shows how to retrieve the list of index classes as well as the list of attributes. The list of index classes is returned in a DKSequentialCollection of Strings. The attributes are returned as a DKSequentialCollection of DKAttributeDef objects. Once a DKAttributeDef object is obtained, you can pull out information about the attribute, such as its name and type, and use the information to form a query.

For further details on these two methods, see the section in the *Application Programming Reference* on DKDatastoreDL.

The complete sample, TListCatalogDL.java, is available in the samples directory.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","USER1","PASSWORD","");
....

DKSequentialCollection pCol2 = null;
dkIterator pIter2 = null;
String strIndexClass = null;
DKAttributeDef aDef = null;
....

DKSequentialCollection pCol = (DKSequentialCollection) dsDL.listSchema();
dkIterator pIter = pCol.createIterator();
while (pIter.more()) {
      strIndexClass = (String) pIter.next();
```

```
      pCol2 = (DKSequentialCollection) dsDL.listSchemaAttributes(strIndexClass);
      pIter2 = pCol2.createIterator();
      while (pIter2.more()) {
        aDef = (DKAttributeDef) pIter2.next();
        System.out.println("  name = " + aDef.name);
        System.out.println("  type = " + aDef.type);
      }
}
```

Note that these two methods require a valid connection to the datastore.

Understanding datastore text search (Text Search Engine)

A DKDatastoreTS (datastore TS) represents the Text Search Engine. Text Search Engine does not actually store the data, it merely indexes the data stored in a Digital Library datastore to support a text search on them. The result of a text search is an item identifier (`item-id`) describing the location of the document in the Digital Library datastore. These identifiers can be used to retrieve the document from the Digital Library datastore.

The datastore TS does not support add, update, retrieve, and delete operations. You can perform queries against this datastore. Refer to “Loading data to be indexed by Text Search Engine” on page 126 for information on adding data to Digital Library that is indexed by Text Search Engine.

Establishing a connection

The datastore TS provides two methods for connect and a method for disconnect. The following is an example of the first connect method using the text search server TM. Normally, you would create a datastore TS, connect to it, do some work, then disconnect when done.

A complete sample, TConnectTS.java, is available in the samples directory.

```
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect("TM","","","");
....                        // do some works
dsTS.disconnect();
```

The following example shows the second connect method using the TS server hostname `apollo`, port number `7502`, and TCP/IP communication type:

```
dsTS.connect("apollo","7502",DK_CTYP_TCPIP);
```

The following example shows the first connect method using the text search server hostname `apollo`, port number `7502`, communication type `T` (TCP/IP):

```
dsTS.connect("apollo","","","PORT=7502;COMMTYPE=T");
```

The following example shows the second connect method using the text search server name `TM`:

```
dsTS.connect("TM","",'');
```

The following example shows the first connect method using the text search server `TM`, the Digital Library library server `LIBSRVR2`, user ID `USER1`, and password `PASSWORD`.

```
dsTS.connect("TM","","","LIBACCESS=(LIBSRVR2,USER1,PASSWORD)");
```

Note that the last parameter, also called *connect-string*, can be used to pass a sequence of parameters in one string.

Getting and setting datastore TS options

The datastore TS provides some options that you can set or get using its methods. The following example shows how to set and get the option for a TS character code set. See the *Application Programming Reference* for the list of options and their descriptions.

```
Integer input_option  = new Integer(DK_CCSID_00850);
Integer output_option = null;

dsTS.setOption(DK_OPT_TS_CCSID,input_option);
output_option = (Integer) dsTS.getOption(DK_OPT_TS_CCSID);
```

The option usually takes an integer value, but it could be of any type.

List servers

The datastore TS provides a method to list the text search servers that it can connect to. The following example shows how to retrieve the list of servers. The list of servers is returned in a DKSequentialCollection of DKServerInfoTS objects. Once a DKServerInfoTS object is obtained, you can pull out the server name and server location, and use the server name to establish a connection to it.

The complete sample, TListCatalogTS.java, is available in the samples directory.

```
DKServerInfoTS pSV = null;
String strServerName = null;
char   chServerLocation = ' ';
DKSequentialCollection pCol = (DKSequentialCollection) dsTS.listServers();
dkIterator pIter = pCol.createIterator();
while (pIter.more()) {
      pSV = (DKServerInfoTS)pIter.next();
      strServerName = pSV.serverName();
      chServerLocation = pSV.serverLocation();
      System.out.println(" Server Name     : " + strServerName);
      System.out.println(" Server Location : " + chServerLocation);
     }
```

List schema

The datastore provides methods for listing the schema. In the text search case, these are text search indexes. The following example shows how to retrieve the list of indexes. The list of indexes returned is in a DKSequentialCollection of DKIndexTS objects. Once a DKIndexTS object is obtained, you can pull out information about the index, such as its name and library ID, and use the name to form a query.

The complete sample, TListCatalogTS.java, is available in the samples directory.

```
String strIndexName = null;
String strLibId = null;
DKIndexTS pIndx = null;

...

DKSequentialCollection pCol  = (DKSequentialCollection) dsTS.listSchema();
dkIterator pIter = pCol.createIterator();
while (pIter.more()) {
      pIndx = (DKIndexTS)pIter.next();
```

```
      strIndexName = pIndx.indexName();
      strLibId = pIndx.libraryId();
      System.out.println(" Index Name : " + strIndexName);
      System.out.println(" Library Id : " + strLibId);
     }
```

Collections and iterators

dkCollection is an abstract class which provides the interface to collection functions. DKSequentialCollection provides the concrete implementation of those functions. Other collections are derived as a subclass of DKSequentialCollection. In essence, these collections contain objects as members.

Usually, collection members are objects of the same type. However, you can have members with different types in one collection.

Sequential collection

DKSequentialCollection provides methods for adding, retrieving, removing, and replacing its member. In addition, it also has a sort() function, which will be discussed later. The following example illustrates how to add a new member to a collection:

```
DKSequentialCollection sq = new
DKSequentialCollection();
String str = " first member ";
sq.addElement(str);         // add a new element at the last position
```

addElement() takes an Object as the parameter. The above statement works because String is a subclass of Object.

Sequential iterator

Iterators are provided to let you iterate over collection members. There are two types of iterators: base iterator dkIterator, which supports next(), more(), and reset() operations; and its subclass DKSequentialIterator, which has more methods. An iterator is created by calling the method createIterator() on the collection. This method creates a new iterator and returns it to you. Use the following code to iterate over a collection:

```
dkIterator iter = sq.createIterator();   // create an iterator for sq
Object member;
while(iter.more()) {                     // while there are more members
      member = iter.next();                 // get the current member and
                                            // advance iter to the next member
      System.out.println(member);           // display it, if you want to
      ....                                  // do other processing
     }
```

DKSequentialIterator provides additional methods to move the iterator bidirectionally. The above code could be rewritten as follows:

```
DKSequentialIterator iter =              // create a seq. iterator for sq
    (DKSequentialIterator) sq.createIterator();
Object member;
while(iter.more()) {
      member = iter.at();                   // get the current member
      ....                                  // do other processing
      iter.setToNext();                     // advance to the next position
     }
```

This code allows you to do some operations at the current member before moving along to the next member. Such an operation would be, for example, replacing a member with a new one, or removing it.

```
String st1 = "the new first member";
sq.replaceElementAt(st1, iter);   // replace current member with a new one
....                              // or
sq.removeElementAt(iter);         // remove the current member
....
```

Note that when you remove the current member, the iterator will be advanced to the next member. Remember to take this fact into account when removing a member inside a loop construct. That is, you need to do a check like the following:

```
....
if (removeCondition == true)
     sq.removeElementAt(iter);   // remove current member, do not advance iter
                                 // since it is advanced to the next after
                                 // the removal operation
else
     iter.setToNext();           // no removal, advance the iterator to the
....                             // next position
```

The above check is necessary to avoid skipping the next member after removing the current one.

Sorting the collection

The `sort()` method allows you to sort collection members based on a specified key in either ascending or descending order. You need to pass in a sort function object and the desired order. The interface for sort function objects is defined in `dkSort.java`; so you can write your own sort function for sorting your specific collection. The following example illustrates how to sort a collection of DDOs based on each DDO's item-id:

```
DKResults rs;
....                  // execute a query to fill DKResults with DDOs
....
DKSortDDOId sortId;   // the sort function; sort on item-id
rs.sort(sortId);      // by default, sort in ascending order
....
```

The definition of a function object `DKSortString` is provided in the sample directory as an example of creating a new sort function object.

Using DDO (Dynamic Data Object)

DKDDO can be regarded as a container of attributes. An attribute, also called data-item (these terms will be used interchangeably), has a name, value, and properties. Each attribute is identified by a data_id, which is a number starting from 1 up to the total number of attributes in the DDO. Since the number, name, value, and property of an attribute may vary, DKDDO provides flexible mechanisms to represent data originating from a variety of datastores and formats, be it items from different index classes in Digital Library, or rows from different tables in a relational database, etc. The DKDDO itself can have properties which apply to the whole DKDDO, instead of to only one particular attribute.

You need to associate a DKDDO with a datastore to be able to call the methods add, retrieve, update and delete to send its attributes into the datastore and retrieve them. This is done by calling the proper DKDDO constructor or by calling `setDatastore()` method.

Every DKDDO has a persistent object identifier (PID), which contains information for locating the data-items in the datastore. Recall that, in Digital Library, a DDO represents an item, which could be a document or a folder.

Creating a DKDDO

The simplest way to create a DKDDO is by calling its constructor which takes no parameter.

```
DKDDO addo = new DKDDO();
```

If you have some ideas on how many attributes the DDO has, you can pass this information to the constructor:

```
DKDDO bddo = new DKDDO(10);
```

The DKDDO will be constructed, leaving enough room to hold 10 attributes. This is more efficient than the previous statement, since `bddo` does not have to grow on the fly to accommodate more attributes.

You can create a DKDDO by supplying datastore and object type as input:

```
DKDatastoreDL dsDL = new DKDatastoreDL();   // create a digital library datastore
DKDDO cddo = new DKDDO(dsDL, "GRANDPA");    // create a DDO to hold an object type
                                            // GRANDPA in dsDL
```

Pid

All DDO must have a persistent object identifier or Pid. It contains information about the datastore name, datastore type, ID, and object type. The ID identifies the location of the DDO's persistent data in the datastore. For a Digital Library datastore, this ID is the item-id. The item-id is one of the most important pieces of information for the retrieve, update, and delete operations. For add, the item-id will be created and returned by the datastore.

To create a DDO of a known item for retrieval, you could do the following:

```
DKDatastoreDL dsDL = new DKDatastoreDL();   // create a digital library datastore
DKPid pid = new DKPid();
pid.setObjectType("GRANDPA");               // set the index-class name it belongs to
pid.setId("LN#U5K6ARLGM3DB4");              // set the item-id
DKDDO ddo = new DKDDO(dsDL,pid);            // create a DDO with pid and associated
                                            // to dsDL
```

Then you can connect to the datastore and call retrieve to retrieve this DDO. This topic will be discussed further in "Retrieving a document or folder" on page 116.

Adding data-items and properties

Suppose the index class `GRANDPA` has the following two attributes:

```
Attribute    data_id=1      2
---------------------------------
Name           Title       Subject
Type           String      String
Nullable       no          yes
```

then, you can represent the above information in a DKDDO as follows:

```
DKDatastoreDL dsDL = new DKDatastoreDL();   // create a digital library datastore
DKDDO cddo = new DKDDO(dsDL, "GRANDPA");    // create a DDO to hold an object type
                                            // GRANDPA in dsDL
Boolean yes   = new Boolean(true);
Boolean no    = new Boolean(false);
Short vstring = new Short(DK_VSTRING);
Object obj;

// add the first attribute
short data_id = cddo.addData("Title");      // add a new attribute named "Title"

// add a property named: "type", set to value : variable length string
cddo.addDataProperty(data_id, DK_PROPERTY_TYPE, vstring);

// add a property named: "nullable", set to value : boolean false
cddo.addDataProperty(data_id, DK_PROPERTY_NULLABLE, no);

// add the second attribute
data_id = cddo.addData("Subject");   // add a new attribute named "Subject"

// add a property named: "type", set to value : variable length string
cddo.addDataProperty(data_id, DK_PROPERTY_TYPE, vstring);

// add a property named: "nullable", set to value : boolean true
cddo.addDataProperty(data_id, DK_PROPERTY_NULLABLE, yes);
```

The above example illustrates the standard properties that an attribute should have, namely `type` and `nullable`. In practice, you can have as many additional properties as required by your application.

Adding properties to a DDO

So far, the above DKDDO has all the attribute information it needs to have. However, there is no information to indicate the kind of DKDDO it is; it must be either a document or a folder. This information is recorded under DKDDO properties, as opposed to attribute properties.

```
cddo.addProperty(DK_PROPERTY_ITEM_TYPE, new Short(DK_DOCUMENT));  // it is a document
```

Setting and getting data_item values

After you create a data_item and its properties, you can set its value:

```
// set Title value to the given string
//   assume we know the data_id for the data_item "Title" is 1
obj = new String("One dark and stormy night");
cddo.setData(1, obj);

// set Subject value to the given string; Subject exists
//   assume we do not know the data_id for the data_item "Subject"
//   find data_id for data_item named "Subject"
data_id = cddo.dataId("Subject");
obj = new String("Mystery");
cddo.setData(data_id, obj);
```

To get the value back, use the `getData()` method:

```
obj = cddo.getData(1);
System.out.println("Title   = " + obj);  // displays "One dark and stormy night"
System.out.println("Subject = " + cddo.getData(data_id));  // displays "Mystery"
```

Getting the properties

When processing a DKDDO, the first thing you want to know is its type—a document or a folder. The following code illustrates such a check:

```
short prop_id = cddo.propertyId(DK_PROPERTY_ITEM_TYPE);
if (prop_id > 0) {
    short type = ((Short) cddo.getProperty(prop_id)).shortValue();
    switch(type) {
       case DK_DOCUMENT:
         // process document
         ....
         break;
       case DK_FOLDER:
         // process folder
         ....
         break;
    }
}
```

To retrieve properties of an attribute, you need to have the data_id of the attribute:

```
data_id = cddo.dataId("Title");  // get data_id of Title
                                 // how many prop does it have
short number_of_data_prop = cddo.dataPropertyCount(data_id);
//  displays all data properties belonging to this attribute
//  notice that the loop index starts from 1,  1 <= i <= number_of_data_prop
for(short i = 1; i <= number_of_data_prop; i++) {
    System.out.println(i + "  Property Name = " + cddo.getDataPropertyName(data_id,i)
                       + "  value = " + cddo.getDataProperty(data_id,i));
    }
```

It is important to notice that data_id and property_id start from 1. If you specify zero you will get an exception.

Displaying the whole DDO

During the course of application development, a developer may need to display the content of a DKDDO for debugging purposes. The following code will do the task:

```
short number_of_attribute = cddo.dataCount();
short number_of_prop      = cddo.propertyCount();
short number_of_data_prop;
// list DDO properties
for (short k = 1; k <= number_of_prop; k++) {
     System.out.println( k + " Property Name = " + cddo.getPropertyName(k) +
                         ",\t   value = " + cddo.getProperty(k));
}
// list data-items and their properties
for (short i = 1; i <= number_of_attribute; i++) {
     System.out.println( i + " Attr. Name = " + cddo.getDataName(i) +
                         ",\t   value = " + cddo.getData(i));
     number_of_data_prop = cddo.dataPropertyCount(i);
     for (short j = 1; j <= number_of_data_prop; j++) {
          System.out.println( "\t" + j + " Data Prop. Name = " +
                              cddo.getDataPropertyName(i,j)    +
                              ",\t   value = "                 +
                              cddo.getDataProperty(i,j));
     }
}
```

DDO representing Digital Library information

A DDO associated with DKDatastoreDL has some specific information to represent the Digital Library document metaphor: document, folder, parts, item, item-id, rank, etc. The following subsections explain how this information is recorded.

DDO properties

As discussed before, the type of item, either a document or folder, is recorded as a property of the DDO, under the name DK_PROPERTY_ITEM_TYPE. To get the item-type of the DDO, you call:

```
Object obj = cddo.getPropertyByName(DK_PROPERTY_ITEM_TYPE);
if (obj != null) {
    short item_type = ((Short) obj).shortValue();
}
```

After the call, the item_type will be equal to DK_DOCUMENT for document, or DK_FOLDER if it is a folder. The if-statement is a check to make sure that the property exists. See “Adding properties to a DDO” on page 87 and “Getting the properties” on page 88.

Pid

The Pid records two important pieces of information specific to Digital Library. The object-type indicates the index-class the DDO belongs to. The ID contains the item-id of the associated item in the datastore. See “Pid” on page 86.

Representing Digital Library documents

A DDO representing a Digital Library document has a property DK_PROPERTY_ITEM_TYPE equal to DK_DOCUMENT, as described above. Its Pid contains index-class name as the object-type, and item-id in the Pid’s ID. The parts inside a document are represented as a DKParts object, which are collections of blobs, each of which is represented as a DKBlobDL object. A document DDO has a specific attribute with a reserved name DKPARTS, whose value is a DKParts object. To get to each part in a document, you need to pull out DKParts first, then create an iterator to iterate over each part as you would normally process a collection. If the document does not have a part at all, the DKPARTS will have a null value, but the DKPARTS attribute will always be present in a document DDO created by the datastore. Documents associated with a combination of parametric and text query (combined query) may have a transient attribute DKRANK, whose value is an Object containing an integer rank computed by the text search engine.

Sample code to create and process `DKParts` is given in “Creating and using DKPARTS attribute” on page 91, “Retrieving a document or folder” on page 116, and “Creating, updating, and deleting documents or folders” on page 117.

Representing Digital Library folders

A DDO representing a Digital Library folder has a property DK_PROPERTY_ITEM_TYPE equal to DK_FOLDER. Similar to a document DDO, its Pid contains index-class name as the object-type, and item-id in the Pid’s ID.

The folder table-of-contents inside a Digital Library folder is represented as a DKFolder object, which is a collection of DDOs, each of which represents an

item—either a document or another folder—belonging to this Digital Library folder. A folder DDO has a specific attribute with a reserved name DKFOLDER, whose value is a DKFolder object.

To get to each DDO member of the folder, you need to retrieve DKFolder first, then create an iterator to iterate over each item member as you would normally process a collection. If the folder does not have a member at all, the DKFOLDER will have a null value, but the DKFOLDER attribute will always be present in a folder DDO created by the datastore.

Sample code to create and process DKFolder is given in the sections “Creating and using DKFOLDER attribute” on page 92, “Retrieving a document or folder” on page 116, and “Creating, updating, and deleting documents or folders” on page 117.

DDO representing Text Search Engine information

A DDO associated with DKDatastoreTS has some specific information to represent text searching results. It does not have a property item-type like DKDatastoreDL. The format of its ID is also different. A DDO resulting from a text query corresponds to a text part inside a Digital Library item. It has a set of standard attributes, described below. As a convention, the attribute value is always an Object.

DKDLITEMID
: The Digital Library item-id where this text part belongs. This item-id is used to retrieve the whole item from Digital Library datastore.

DKPARTNO
: An integer part-number of this text part. It is used together with the item-id to retrieve this part from Digital Library datastore.

DKREPTYPE
: the Digital Library rep-type of this text part. This attribute, when used in conjunction with the item-id and part-number, retrieves the part from the Digital Library datastore.

DKRANK
: An integer rank signifies the relevance of this part to results set of the text query. A higher rank means a better match. See the *Text Search Engine: Application Programming Reference* manual for further information.

DKDSIZE
: An integer number of word-occurrences (only in the results of boolean queries). See the *Text Search Engine: Application Programming Reference* manual for further information.

DKRCNT
: An integer number of matches (only in the results of boolean queries). See the *Text Search Engine: Application Programming Reference* manual for further information.

The Pid for a Text Search DDO has the following information:

datastore type
: TS

datastore name
: The server name used to connect to the datastore

object type
: Text search index

id Text Search Engine document ID

Creating and using DKPARTS attribute

As mentioned earlier, the DKPARTS attribute in a DDO represents the collection of parts in a document. The value of this attribute is a DKParts object, which is a collection of XDOs. This attribute is set during DDO retrieve, or created and set by user, as shown below:

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKParts parts = new DKParts();  // create new DKParts, collection of parts

DKBlobDL blob  = new DKBlobDL(dsDL);  // create new XDO blob
DKPidXDODL pid = new DKPidXDODL();    // create Pid for this XDO object

pid.setPartId(5);                     // set part number to 5
pid.setId("LN#U5K6ARLGM3DB4"          // the item-id this part belongs to
blob.setPid(pid);                     // set the Pid for the XDO blob
blob.setContentClass(DK_CC_GIF);      // set content class type GIF
blob.setRepType(DK_REP_NULL);         // set rep type for the part
blob.setContentFromClientFile("choice.gif");  // set the blob's content
blob.setInstanceOpenHandler("xv");    // the viewer program on AIX

parts.addElement(blob);           // add the blob to the parts collection

....                              // create and add some more blobs to the
....                              // collection as necessary

DKDDO ddo = new DKDDO();          // create a ddo
....                              // sets some of its attributes

Object obj = new Short(DK_DOCUMENT);  // set the type to document DDO
ddo.addProperty(DK_PROPERTY_ITEM_TYPE, obj);

// create DKPARTS attribute and sets it to refer to the DKParts object
short data_id = ddo.addData(DKPARTS);  // add attribute "DKParts"
obj = new Short(DK_COLLECTION_XDO);    // add type property
ddo.addDataProperty(data_id,DK_PROPERTY_TYPE,obj);
obj = new Boolean(true);               // add nullable property
ddo.addDataProperty(data_id,DK_PROPERTY_NULLABLE,obj);
ddo.setData(data_id, parts);           // sets the attribute value
```

Once a DKParts is set to be an attribute value of a DDO, the DDO owns it.

Getting back the parts from a DDO is a lot simpler:

```
data_id = ddo.dataId(DKPARTS);  // get DKPARTS data-id
if (data_id == 0)               // parts not found
    throw new DKException(" parts data-item not found");

DKParts pCol = (DKParts) ddo.getData(data_id); // get the parts collection

// create iterator and process the part collection member one by one
if (pCol != null) {
    DKBlobDL blob;
    dkIterator iter = pCol.createIterator();
    while (iter.more()) {
       blob = (DKBlobDL) iter.next();
       if (blob != null) {
           blob.retrieve();   // retrieve the blob
           blob.open();       // display the blob using the viewer
           ....               // other processing
       }
    }
}
```

Creating and using DKFOLDER attribute

In a folder DDO, the DKFOLDER attribute represents a collection of folders and documents that belong to the folder. The value of this attribute is a DKFolder object, which is a collection of DDOs. Similar to DKPARTS, DKFOLDER is set during DDO retrieve, or created and set by user, as shown below:

```
DKDatastoreDL dsDL = new DKDatastoreDL();
...                                  // create a new DKFolder, collection of DDO
DKFolder    folder = new DKFolder();

DKDDO member = new DKDDO();          // create the first member of this folder
...                                  // sets member DDO attributes and properties
folder.addElement(member);           // add member to the folder collection
....                                 // create and add some more member DDO to the
....                                 // DDO collection as necessary
DKDDO ddo = new DKDDO();             // create a folder ddo
....                                 // sets some of its attributes

Object obj = new Short(DK_FOLDER);  // set the type to folder DDO
ddo.addProperty(DK_PROPERTY_ITEM_TYPE, obj);

// create DKFOLDER attribute and sets it to refer to the DKFolder object
short data_id = ddo.addData(DKFOLDER);  // add attribute "DKFolder"
obj = new Short(DK_COLLECTION_DDO);     // add type property
ddo.addDataProperty(data_id,DK_PROPERTY_TYPE,obj);
obj = new Boolean(true);                // add nullable property
ddo.addDataProperty(data_id,DK_PROPERTY_NULLABLE,obj);
ddo.setData(data_id, folder);           // sets the attribute value
```

Once a DKFolder is set to be an attribute of a DDO, the DDO owns it.

Getting back the folder from a DDO is also a lot simpler:

```
data_id = ddo.dataId(DKFOLDER);  // get DKFOLDER data-id
if (data_id == 0)                // folder not found
    throw new DKException(" folder data-item not found");

DKFolder fCol = (DKFolder) ddo.getData(data_id); // get the parts collection

// create iterator and process the DDO collection member one by one
if (fCol != null) {
    DKDDO item;
    dkIterator iter = fCol.createIterator();
    while (iter.more()) {
       item = (DKDDO) iter.next();
       if (item != null) {
           item.retrieve();      // process the member DDO
           ....                  // other processing
       }
    }
}
```

Using XDO

An XDO represents a single part in Digital Library. There are two types of XDO: DKBlobDL and DKClobDL. DKBlobDL is for binary objects and DKClobDL is for character objects. Both DKBlobDL and DKClobDL require datastore DKDatastoreDL as input to create the object instance.

XDO Pid

An XDO needs to have a Pid in order to store its data persistently. The itemId and partId of DKPidXDODL are required for XDO to locate the persistent data in a Digital Library datastore.

XDO data members

You should be aware of some important information about an XDO, such as RepType, ContentClass, AffiliatedType, AffiliatedData, SearchEngine, SearchIndex, and SearchInfo. If no values are set for these members, the object content will not be indexed by a search engine and the default values will be provided as follows:

```
RepType is  "FRN$NULL"
ContentClass is  DK_CC_UNKNOWN
AffiliatedType is DK_BASE
AffiliatedData is NULL
```

Tip: For the valid values of ContentClass, see the file DKConstant2.h shipped with Digital Library under the include directory.

Indexing XDOs by search engine

If you want the object content to be indexed by a search engine (for example, Text Search Engine), the values of **SearchEngine**, **SearchIndex** and **SearchInfo** are required. Be aware that the SearchIndex value is a combination of two names: search service name and search index name. For example, if, in the system administration program, you have the text search server named `TM` and you defined a search index named `TMINDEX` for it, then the correct value for the SearchIndex is `TM-TMINDEX`.

For the object that is to be indexed by Text Search Engine, the value of **SearchEngine** must be `SM`, and for the object that is to be indexed by QBIC, the value of **SearchEngine** must be `QBIC`. The **SearchIndex** of QBIC is a combination of three values: QBIC database name, QBIC catalog name, and QBIC server name. For example, if the QBIC database name is `SAMPLEDB`, the QBIC catalog name is `SAMPLECAT`, and the QBIC server name is `QBICSRV`, then the correct value for the **SearchIndex** would be `SAMPLEDB-SAMPLECAT-QBICSRV`.

Please refer to the files LoadSampleTSQBIC.java and LoadFolderTSQBIC.java included with the product, as working code examples to load data, or to create a folder and load data. These are located in the samples directory.

Important: When adding a part object to be indexed by a search engine, don't set the repType. Currently, the Text Search Engine search manager only works with the default repType `FRN$NULL`.

Programming tips

The combination of itemId, partId and repType is the key to identify an XDO. To handle a stand-alone XDO, you need to provide the itemId and partId. repType is optional since the system provides a default value for it.

During an `add()` operation, if you set partId to 0, the system assigns an available partId for it. You can retrieve the partId value after `add()` if you want to do some other operation with that object later.

You can use the following statement after `add()` to obtain the system assigned partId:

```
int partID = ((DKPidXDODL)(axdo.getPid())).getPartId();
```

Important: There are two situations where a valid partId is required and you cannot set partId to 0:

1. Adding a part to be indexed by search manager.
2. Adding a large object that will be divided into MAXPIECE size pieces.

XDO as a part of DDO instead of stand alone XDO

In Digital Library, an XDO represents a single part object when a DDO is a document that is a collection of part objects. You can manipulate the XDO as a component of the DDO or as a stand alone object. To handle it as a stand alone object, you need to know the existing itemId for the XDO. To handle as a part of the DDO, the DDO provides the itemId for the XDO.

XDO as a part of DDO

For the XDO as a part of DDO, refer to the file TLoadSample.java shipped with this product as the working code example. It is located in the samples directory.

The major statements to relate the XDO with DDO are:

```
//create DDO
DKPid pid = new DKPid();
pid.setObjectType(indexClassName);
DKDDO ddo = new DKDDO(dsDL, pid);
ddo.addProperty(DK_PROPERTY_ITEM_TYPE,
                new Short(DK_DOCUMENT));
...
...
DKParts parts = new DKParts();

//create XDO
DKBlobDL  axdo = new DKBlobDL(dsDL);
DKPidXDODL apid = new DKPidXDODL();
apid.setPartId(partId);
axdo.setPid(apid);
axdo.setContentClass(DK_CC_GIF);
axdo.setAffiliatedType(DK_BASE);
axdo.setContentFromClientFile(imageNames[i]);

//add XDO to the DKParts collection
parts.addElement(axdo);
...
...
//add DDO
dataId = ddo.addData(DKPARTS);
ddo.addDataProperty(dataId, DK_PROPERTY_TYPE,
     new Short(DK_COLLECTION_XDO));
ddo.setData(dataId, parts);
ddo.add();
```

Stand alone XDO

The following code examples are for a stand alone XDO.

Add an XDO from the buffer: Add an XDO from the buffer:

```
//*****************************************************************************
// txdoadd.java - Test dkblobdl for add()
// (this test case requires you to know the existing itemId in the library.)
//*****************************************************************************
import COM.ibm.mm.sdk.server.*;
import COM.ibm.mm.sdk.common.*;
import java.io.*;
import java.lang.*;

public class txdoadd
       implements DKConstant {

  public static void main(String[] args)
  {

    int    partId = 0;                          //let system decide the partId
    String itemId = "CPPIORH4JBIXWIY0";         //existing itemId
    String fileName = "g:\\test\\cheetah.gif"; //a Windows NT file for add
    try {
         ...                                    //required datastore
         DKDatastoreDL dsDL = new DKDatastoreDL();
         ...                                    //connection to datastore
         dsDL.connect("LIBSRVRH","FRNADMIN","PASSWORD","");
         DKBlobDL axdo = new DKBlobDL(dsDL);      //create XDO
         DKPidXDODL  apid = new DKPidXDODL();     //create Pid
         apid.setPartId(partId);                  //set partId
         apid.setId(itemId);                      //set itemId
         axdo.setPid(apid);                       //setPid to XDO
         axdo.setContentClass(DK_CC_GIF);         //set ContentClass
         axdo.setContentFromClientFile(fileName); //set file content to buffer area
         axdo.add();                              //add from buffer
         System.out.println("after add partId = " + ((DKPidXDODL)
         (axdo.getPid())).getPartId());           //display the partId after add
         dsDL.disconnect();                       //disconnect from datastore
        }
        catch (DKException exc)
        {
         System.out.println("Exception name " + exc.name());
         System.out.println("Exception message " + exc.getMessage());
         exc.printStackTrace();
        }
  } //end of main
} //end of class
```

Add an XDO from a file: Add an XDO from file:

```
//*****************************************************************************
// txdoaddf.java - Test DKBlobDL for add(fileName)
//*****************************************************************************
import COM.ibm.mm.sdk.server.*;
import COM.ibm.mm.sdk.common.*;
import java.io.*;
import java.lang.*;

public class txdoaddf
       implements DKConstant {

  public static void main(String[] args)
  {

    int    partId = 0;                           //partId 9 is not used yet
    String itemId = "CPPIORH4JBIXWIY0";          //existing itemId
    String fileName = "/u4/mmdb/test/choice.gif"; //an AIX file for add
    try {
         DKDatastoreDL dsDL = new DKDatastoreDL();  //required datastore
         dsDL.connect("LIBSRVRH","FRNADMIN","PASSWORD",""); //connect
         DKBlobDL axdo = new DKBlobDL(dsDL);        //create XDO
         DKPidXDODL  apid = new DKPidXDODL();       //create Pid
```

```
          apid.setPartId(partId);                    //set partId in Pid
          apid.setId(itemId);                        //set itemId in Pid
          axdo.setPid(apid);                         //setPid to XDO

          axdo.setRepType("ABCD");                   //set representation type
          axdo.setContentClass(DK_CC_GIF);           //set ContentClass
          axdo.add(fileName);                        //add from file
          dsDL.disconnect();                         //disconnect from datastore
         }
         catch (DKException exc)
         {
          System.out.println("Exception name " + exc.name());
          System.out.println("Exception message " + exc.getMessage());
          exc.printStackTrace();
         }
  } //end of main
} //end of class
```

Add an XDO to be indexed by Text Search Engine: Add an XDO to be indexed by Text Search Engine:

```
//*****************************************************************************
// txdosadd.java - Test dkblobdl for adding an search indexed part object
// (this test case requires you to know the existing itemId in the library.)
//*****************************************************************************
import COM.ibm.mm.sdk.server.*;
import COM.ibm.mm.sdk.common.*;
import java.io.*;
import java.lang.*;

public class txdosadd
       implements DKConstant {

  public static void main(String[] args)
  {

    int    partId = 0;                          //let system decide the partId
    String itemId = "CPPIORH4JBIXWIY0";         //existing itemId
    String fileName = "g:\\test\\cheetah.gif"; //file content will be indexed
    try {
         DKDatastoreDL dsDL = new DKDatastoreDL();//required datastore
         ...                                      //connection to datastore
         dsDL.connect("LIBSRVRH","FRNADMIN","PASSWORD","");
         DKBlobDL axdo = new DKBlobDL(dsDL);      //create XDO
         DKPidXDODL  apid = new DKPidXDODL();     //create Pid
         apid.setPartId(partId);                  //set partId
         apid.setId(itemId);                      //set itemId
         axdo.setPid(apid);                       //setPid to XDO
         axdo.setContentClass(DK_CC_ASCII);       //set ContentClass to text

         //---set searchEngine ----- (deprecated)
         axdo.setSearchEngine("SM");
         axdo.setSearchIndex("TM-TMINDEX");
         axdo.setSearchInfo("ENU");

         //---set searchEngine ----- (using extension object)
         DKSearchEngineInfoDL aSrchEx = new DKSearchEngineInfoDL();
         aSrchEx.setSearchEngine("SM");
         aSrchEx.setSearchIndex("TM-TMINDEX");
         aSrchEx.setSearchInfo("ENU");
         axdo->setExtension("DKSearchEngineInfoDL", (dkExtension)aSrchEx);

         ...                                    //set file content to buffer area
         axdo.setContentFromClientFile(fileName);
         axdo.add();                            //add from buffer
         ...                                    //display the partId after add
         System.out.println("after add partId = " + ((DKPidXDODL)
```

```
                                (axdo.getPid())).getPartId());

          System.out.println("about to call retrieve()");
          axdo.retrieve();                                  //retrieve object
          axdo.setInstanceOpenHandler("notepad", true);  //set open viewer
          System.out.println("about to do open()...");
          axdo.open();                               //open content with viewer
          dsDL.disconnect();                         //disconnect from datastore
         }
         catch (DKException exc)
         {
          System.out.println("Exception name " + exc.name());
          System.out.println("Exception message " + exc.getMessage());
          exc.printStackTrace();
         }
  } //end of main
} //end of class
```

Add an annotation object to an XDO: Add an annotation object to an XDO:

```
// To add an annotation object, insert the following statements before
// the add operation.

//------- set DKAnnotation------- (deprecated)
axdo.setAffiliatedType(DK_ANNOTATION);
DKAnnotation ann = new DKAnnotation();
ann.setPart(14);
ann.setPageNumber(1);
ann.setX((short)5);
ann.setY((short)5);
axdo.setAffiliatedData(ann);

//------ set DKAnnotationDL ------ (using extension object)
axdo->setAffiliatedType(DK_ANNOTATION);
DKAnnotationDL ann = new DKAnnotationDL();
ann.setPart(14);
ann.setPageNumber(1);
ann.setX((short)5);
ann.setY((short)5);
axdo->setExtension("DKAnnotationDL", (dkExtension)ann);
```

Retrieve, update, and delete an XDO: Retrieve, update, and delete an XDO:

```
// To retrieve, update or delete the object in datastore, you should provide
// the correct itemId, partId and repType to identify the object.

//*****************************************************************************
// txdorud.java - Test dkblobdl for retrieve/update/delete object.
//*****************************************************************************
import COM.ibm.mm.sdk.server.*;
import COM.ibm.mm.sdk.common.*;
import java.io.*;
import java.lang.*;

public class txdorud
       implements DKConstant {

  public static void main(String[] args)
  {

    int    partId = 17;                        //partId of object
    String itemId = "CPPIORH4JBIXWIY0";        //existing itemId
    String fileName = "g:\\test\\choice.gif";  //file content to update
    try {
         DKDatastoreDL dsDL = new DKDatastoreDL();//required datastore
         ...                                      //connection to datastore
         dsDL.connect("LIBSRVRH","FRNADMIN","PASSWORD","");
         DKBlobDL axdo = new DKBlobDL(dsDL);      //create XDO
```

```
          DKPidXDODL  apid = new DKPidXDODL();     //create Pid
          apid.setPartId(partId);                  //set partId
          apid.setId(itemId);                      //set itemId
          axdo.setPid(apid);                       //setPid to XDO
          axdo.retrieve();                         //retrieve the object
          axdo.setContentFromClientFile(fileName); //set file content to buffer area
          axdo.update();                           //update the object with buffer data
          axdo.retrieve("new.gif");                //retrieve content to a file
          axdo.del();                              //delete object from datastore
          dsDL.disconnect();                       //disconnect from datastore
         }
         catch (DKException exc)
         {
          System.out.println("Exception name " + exc.name());
          System.out.println("Exception message " + exc.getMessage());
          exc.printStackTrace();
         }
  } //end of main
} //end of class
```

Indexing an existing XDO using search engines: This example tests the setToBeIndexed method of the DKBlobDL API.

```
//*****************************************************************************
// indexPartxs.java - Test setToBeIndexed method of DKBlobDL
// invoke: indexPartxs <partId> <repType> <itemId>
// if itemId, partd or repType are not provided, default values will be used.
// (this test case requires you to know the existing itemId and partId; you
// also need to replace the library server, userid and password of your own)
//*****************************************************************************
import COM.ibm.mm.sdk.server.*;
import COM.ibm.mm.sdk.common.*;
import java.io.*;
import java.lang.*;

public class indexPartxs
       implements DKConstant
{
// Main method
public static void main(String args[])
  {
    int     partId = 3;
    String  itemId = "N2JJBERBQFK@WTVL";
    String repType = "FRN$NULL";
    if (args.length == 3)
    {
      partId = (short)Integer.parseInt(args[0], 10);
      repType = args[1];
      itemId = args[2];
      System.out.println("You enter: java indexPartxs " +
       + partId + " " + repType + " " + itemId);
    }
    if (args.length == 2)
    {
      partId = (short)Integer.parseInt(args[0], 10);
      repType = args[1];
      System.out.println("You enter: java indexPartxs " +
       + partId + " " + repType);
    }
    if (args.length == 1)
    {
      partId =(short)Integer.parseInt(args[0], 10);
      System.out.println("You enter: java indexPartxs " + partId );
      System.out.println("The supplied default repType = " + repType);
      System.out.println("The supplied default itemId = " + itemId);
    }
    if (args.length == 0)
```

```
    {
      System.out.println("invoke: java indexPartxs
                          <partId> <repType> <itemId>");
      System.out.println("No parameter, folldeowing defaults will be provided:");
      System.out.println("     default partId = " + partId);
      System.out.println("     default repType = " + repType);
      System.out.println("     default itemId = " + itemId);
    }

    try
    {
      DKDatastoreDL dsDL = new DKDatastoreDL();
      System.out.println("connecting to datastore");
      //replace following with your library server, userid, pasword
      dsDL.connect("LIBSRVRP","FRNADMIN","PASSWORD","");
      System.out.println("datastore connected");

      DKBlobDL axdo = new DKBlobDL(dsDL);
      DKPidXDODL  apid = new DKPidXDODL();
      apid.setPartId(partId);
      apid.setId(itemId);
      axdo.setPid(apid);
      System.out.println("itemid=" + apid.getId());
      System.out.println("partId=" + apid.getPartId());

      //---set searchEngine -----
      System.out.println("set search engine informations and setToBeIndexed()...");
      DKSearchEngineInfoDL aSrchEx = new DKSearchEngineInfoDL();
      aSrchEx.setSearchEngine("SM");
      aSrchEx.setSearchIndex("TM-TMINDEX");
      aSrchEx.setSearchInfo("ENU");
      axdo.setExtension("DKSearchEngineInfoDL", (dkExtension)aSrchEx);
      axdo.setToBeIndexed();
      System.out.println("setToBeIndexed() done...");

      dsDL.disconnect();
      System.out.println("datastore disconnected");
    }
    catch (DKException exc)
    {
     System.out.println("Exception name " + exc.name());
     System.out.println("Exception message " + exc.getMessage());
     exc.printStackTrace();
    }
  }
}
```

Invoking an XDO function: This example demonstrates how to test the DKBlobDL API.

```
//*****************************************************************************
// txdomisc.java - Test DKBlobDL
// (this test case requires you to know the existing ItemId and partId first)
// invoke: txdomisc <partId> <repType> <itemId>
// if itemId, partd or repType are not provided, default values will be used.
//*****************************************************************************
import COM.ibm.mm.sdk.server.*;
import COM.ibm.mm.sdk.common.*;
import java.io.*;
import java.lang.*;

public class txdomisc implements DKConstant
{
 public static void main(String args[])
 {

    int    partId = 5;
```

```
String itemId = "GAWCVGGVFUG428UJ";
String repType = "";
if (args.length == 3)
{
  partId = (short)Integer.parseInt(args[0], 10);
  repType = args[1];
  itemId = args[2];
  System.out.println("You enter: java txdomisc " +
   + partId + " " + repType + " " + itemId);
}
if (args.length == 2)
{
  partId = (short)Integer.parseInt(args[0], 10);
  repType = args[1];
  System.out.println("You enter: java txdomisc " +
   + partId + " " + repType);
}
if (args.length == 1)
{
  partId =(short)Integer.parseInt(args[0], 10);
  System.out.println("You enter: java txdomisc " + partId );
  System.out.println("The supplied default repType = " + repType);
  System.out.println("The supplied default itemId = " + itemId);
}
if (args.length == 0)
{
  System.out.println("invoke: java txdomisc   ");
  System.out.println("No parameter, folldeowing defaults will be provided:");
  System.out.println("     default partId = " + partId);
  System.out.println("     default repType = " + repType);
  System.out.println("     default itemId = " + itemId);
}

try
{
 DKDatastoreDL dsDL = new DKDatastoreDL();
 System.out.println("connecting to datastore");
 dsDL.connect("LIBSRVRH","FRNADMIN","PASSWORD","");
 System.out.println("datastore connected");

 DKBlobDL axdo = new DKBlobDL(dsDL);
 DKPidXDODL  apid = new DKPidXDODL();
 apid.setPartId(partId);
 apid.setId(itemId);
 axdo.setPid(apid);
 axdo.setRepType(repType);
 System.out.println("repType=" + axdo.getRepType());
 System.out.println("itemid=" + apid.getId());
 System.out.println("partId=" + apid.getPartId());

 //== before retrieve
 System.out.println("before retrieve:");
 System.out.println("  contentclass=" + axdo.getContentClass());
 System.out.print("  content length=" + axdo.length());
 System.out.println(" (the length of this object instance - in memory)");
 System.out.print("  getSize=" + axdo.getSize());
 System.out.println(" (get the object size without retrieving object)");
 System.out.println("  createdTimestamp=" + axdo.getCreatedTimestamp());
 System.out.println("  updatedTimestamp=" + axdo.getUpdatedTimestamp());
 axdo.retrieve();
 //== after retrieve
 System.out.println("after retrieve:");
 System.out.println("  contentclass=" + axdo.getContentClass());
 System.out.print("  content length=" + axdo.length());
 System.out.println(" (the length of this object instance - in memory)");
 System.out.print("  getSize=" + axdo.getSize());
 System.out.println(" (get the object size without retrieving object)");
```

```
     System.out.println("  createdTimestamp=" + axdo.getCreatedTimestamp());
     System.out.println("  updatedTimestamp=" + axdo.getUpdatedTimestamp());
     System.out.println("  affiliatedTyp=" + axdo.getAffiliatedType());
     if (axdo.getAffiliatedType() == DK_ANNOTATION)
     {
      System.out.println("    pageNumber=" +
                         axdo.getAffiliatedData().getPageNumber());
      System.out.println("    partIdr=" + axdo.getAffiliatedData().getPart());
      System.out.println("    X=" + axdo.getAffiliatedData().getX());
      System.out.println("    Y=" + axdo.getAffiliatedData().getY());
     }
     System.out.println("about to do open()...");
     axdo.setInstanceOpenHandler("notepad", true);
     int cc = axdo.getContentClass();
     if ( cc == DK_CC_GIF)
       axdo.setInstanceOpenHandler("lviewpro", true);
     else if (cc == DK_CC_ASCII)
       axdo.setInstanceOpenHandler("notepad", true);
     else if (cc == DK_CC_AVI)
       axdo.setInstanceOpenHandler("mplay32 ", true);
     axdo.open();
     dsDL.disconnect();
     System.out.println("datastore disconnected");
    }
    catch (DKException exc)
    {
     System.out.println("Exception name " + exc.name());
     System.out.println("Exception message " + exc.getMessage());
     exc.printStackTrace();
    }
  }
}
```

Add an XDO Media Object: For every media object added, an entry is created in the Digital Library FRN$MEDIA table. That entry contains the information about the media user data. The physical media object is stored in the VideoCharger server specified in the network table of Digital Library.

```
//*****************************************************************************
// txdoAddVS.java - Test DKBlobDL for add() of media object
// (this test case requires you to know the existing itemId in the library.)
// invoke: java txdoAddVS <fileName> <partId> <itemId>
//   or
// invoke: java txdoAddVS <fileName> <partId>  -- with default itemId
// note:
//  you can also code the itemId, partId and fileName in this program and
//  invoke as:  java txdoAddVS
//*****************************************************************************
import COM.ibm.mm.sdk.server.*;
import COM.ibm.mm.sdk.common.*;
import java.io.*;
import java.lang.*;

public class txdoAddVS
       implements DKConstant
{
// Main method
public static void main(String[] args)
{

    String  fileName = "/icing1.mpg1";        //an media object
    String itemId = "K1A04EWBVHJAV1D7";       //an known itemId
    int partId = 45;
    if (args.length == 3)
    {
      fileName = args[0];
      partId = (int)Integer.parseInt(args[1], 10);
```

```
      itemId = args[2];
      System.out.println("You enter: java txdoAddVS " +
      fileName + " " + partId + " " + itemId);
    }
    if (args.length == 2)
    {
      fileName = args[0];
      partId =(int)Integer.parseInt(args[1], 10);
      System.out.println("You enter: java txdoAddVS " +
      fileName + " " + partId );
      System.out.println("The supplied default itemId = " + itemId);
    }
    if (args.length == 1)
    {
      fileName = args[0];
      System.out.println("You enter: java txdoAddVS " + fileName);
      System.out.println("The supplied default partId = " + partId);
      System.out.println("The supplied default itemId = " + itemId);
    }
    if (args.length == 0)
    {
      System.out.println("invoke: java txdoAddVS <fileName> <partId> <itemId>");
      System.out.println("No parameter, following defaults will be provided:");
      System.out.println("     default fileName = " + fileName);
      System.out.println("     default partId = " + partId);
      System.out.println("     default itemId = " + itemId);
    }
    try
    {
      // *** connect to datastore
      DKDatastoreDL dsDL = new DKDatastoreDL();
      //replace following with your library server, userid, pasword
      System.out.println("connecting to datastore...");
      dsDL.connect("LIBXDB2","FRNADMIN","PASSWORD","");
      System.out.println("datastore connected");

      // *** create xdo and pid
      DKBlobDL axdo = new DKBlobDL(dsDL);
      DKPidXDODL  apid = new DKPidXDODL();
      apid.setPartId(partId);
      apid.setId(itemId);
      axdo.setPid(apid);
      // *** you must use the content class DK_CC_IBMVSS for a media object
      axdo.setContentClass(DK_CC_IBMVSS);
      System.out.println("contentClass=" + axdo.getContentClass());
      System.out.println("partId = "
      + ((DKPidXDODL) (axdo.getPid())).getPartId());

      // *** setup DKMediaStreamInfoDL
      DKMediaStreamInfoDL aVS = new DKMediaStreamInfoDL();
      aVS.setMediaFullFileName(fileName);
      //if fileName contain a list of media segments then use following
      //aVS.setMediaObjectOption(DK_VS_LIST_OF_OBJECT_SEGMENTS);
      aVS.setMediaObjectOption(DK_VS_SINGLE_OBJECT);
      aVS.setMediaHostName("vchargerla.stl.ibm.com");
      aVS.setMediaUserId("root");
      aVS.setMediaPassword("video2");

      //following are optional, if not set default value will be provided
      aVS.setMediaNumberOfUsers(2);
      aVS.setMediaAssetGroup("AG");
      // *** same as defined in VideoCharger server
      aVS.setMediaType("MPEG1");
      aVS.setMediaResolution("SIF");
      aVS.setMediaStandard("NTSC");
      aVS.setMediaFormat("SYSTEM");
```

```
      axdo.setExtension("DKMediaStreamInfoDL", (dkExtension)aVS);

      System.out.println("about to call add()");
      axdo.add();
      System.out.println("add successfully.....");

      System.out.println("after added check for status:");
      boolean flag2 = axdo.isCategoryOf(DK_MEDIA_OBJECT);
      if (flag2)
      {
       DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
                                   axdo.getExtension("DKMediaStreamInfoDL");
       System.out.println(" mediaformat=" + media.getMediaFormat());
       System.out.println(" mediaBitRate=" + media.getMediaBitRate());
       System.out.println(" mediastate(dynamic)=" +
                          axdo.retrieveObjectState(DK_MEDIA_OBJECT));
      }
      dsDL.disconnect();
      System.out.println("datastore disconnected");
    }
    catch (DKException exc)
    {
     System.out.println("Exception name " + exc.name());
     System.out.println("Exception message " + exc.getMessage());
     exc.printStackTrace();
    }
  }
}
```

Delete an XDO Media Object: This example shows how to delete an XDO media object.

```
//*************************************************************************
// txdodelvs.java - Test DKBlobDL for delete and using extension object
// (this test case requires you to know the existing partId and itemId)
// invoke: java txdodelvs <partId> <repType> <itemId>
// You can also code the partId, repType and itemId in this program and
// invoke as: java txdodelvs
//*************************************************************************
import COM.ibm.mm.sdk.server.*;
import COM.ibm.mm.sdk.common.*;
import java.io.*;
import java.lang.*;

public class txdodelvs implements DKConstant
{
 public static void main(String args[])
 {
    int  partId = 45;
    String repType = "";
    String itemId =  "K1A04EWBVHJAV1D7";
    if (args.length == 3)
    {
      partId = (short)Integer.parseInt(args[0], 10);
      repType = args[1];
      itemId = args[2];
      System.out.println("You enter: java txdodelvs " +
       + partId + " " + repType + " " + itemId);
    }
    if (args.length == 2)
    {
      partId = (short)Integer.parseInt(args[0], 10);
      repType = args[1];
      System.out.println("You enter: java txdodelvs " +
       + partId + " " + repType);
    }
    if (args.length == 1)
```

```
        {
          partId =(short)Integer.parseInt(args[0], 10);
          System.out.println("You enter: java txdodelvs " + partId );
          System.out.println("The supplied default repType = " + repType);
          System.out.println("The supplied default itemId = " + itemId);
        }
        if (args.length == 0)
        {
          System.out.println("invoke: java txdodelvs <partID> <repType> <itemId>");
          System.out.println("No parameter, folldeowing defaults will be provided:");
          System.out.println("     default partId = " + partId);
          System.out.println("     default repType = " + repType);
          System.out.println("     default itemId = " + itemId);
        }

        try
        {
          DKDatastoreDL dsDL = new DKDatastoreDL();
          System.out.println("connecting to datastore...");
          //replace following with your library server, userid, pasword
          dsDL.connect("LIBSRVRP","FRNADMIN","PASSWORD","");
          System.out.println("datastore connected");

          DKBlobDL axdo = new DKBlobDL(dsDL);
          DKPidXDODL  apid = new DKPidXDODL();
          apid.setPartId(partId);
          apid.setId(itemId);
          axdo.setPid(apid);
          axdo.setRepType(repType);
          boolean flag2 = axdo.isCategoryOf(DK_MEDIA_OBJECT);
          System.out.println("isMediaObject?=" + flag2);
          if (flag2)
          {
            DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
                                        axdo.getExtension("DKMediaStreamInfoDL");
            System.out.println(" mediaformat=" + media.getMediaFormat());
            System.out.println(" mediaBitRate=" + media.getMediaBitRate());
            System.out.println(" mediastate(dynamic)=" +
                               axdo.retrieveObjectState(DK_MEDIA_OBJECT));
            //set delete option for media object
            axdo.setOption(DK_OPT_DL_DELETE_OPTION,
              (Object)new Integer(DK_DELETE_NO_DROPITEM_MEDIA_AVAIL));
            System.out.println("The setted delete option =" +
              (Integer)(axdo.getOption(DK_OPT_DL_DELETE_OPTION)));
          }

          System.out.println("about to call del().. ");
          axdo.del();
          System.out.println("del successfully.....");
          flag2 = axdo.isCategoryOf(DK_MEDIA_OBJECT);
          System.out.println("after delete isMediaObject? = " + flag2);
          System.out.println("about to call dsDL.disconnect()");
          dsDL.disconnect();
          System.out.println("datastore disconnected");
        }
        catch (DKException exc)
        {
         System.out.println("Exception name " + exc.name());
         System.out.println("Exception message " + exc.getMessage());
         exc.printStackTrace();
        }
      }
    }
```

Retrieve an XDO Media Object: This example shows how to retrieve an XDO media object. The retrieved object contains only the media meta data, not the media object itself.

```
/*****************************************************************************
 * txdoretxs.java - test the xdo retrieve/open and using the extension object.
 * (this test case requires you to know the existing itemId and partId
 * in the library.)
 *****************************************************************************/
import COM.ibm.mm.sdk.server.*;
import COM.ibm.mm.sdk.common.*;
import java.io.*;
import java.lang.*;

public class txdoretxs implements DKConstant
{
// Main method
public static void main(String args[])
 {
    int    partId = 45;
    String itemId = "K1A04EWBVHJAV1D7";
    String repType = "";
    if (args.length == 3)
    {
      partId = (short)Integer.parseInt(args[0], 10);
      repType = args[1];
      itemId = args[2];
      System.out.println("You enter: java txdoretxs " +
       + partId + " " + repType + " " + itemId);
    }
    if (args.length == 2)
    {
      partId = (short)Integer.parseInt(args[0], 10);
      repType = args[1];
      System.out.println("You enter: java txdoretxs " +
       + partId + " " + repType);
    }
    if (args.length == 1)
    {
      partId =(short)Integer.parseInt(args[0], 10);
      System.out.println("You enter: java txdoretxs " + partId );
      System.out.println("The supplied default repType = " + repType);
      System.out.println("The supplied default itemId = " + itemId);
    }
    if (args.length == 0)
    {
      System.out.println("invoke: java txdoretxs <partId> <repType> <itemId>");
      System.out.println("No parameter, folldeowing defaults will be provided:");
      System.out.println("     default partId = " + partId);
      System.out.println("     default repType = " + repType);
      System.out.println("     default itemId = " + itemId);
    }

    try
    {
      DKDatastoreDL dsDL = new DKDatastoreDL();
      System.out.println("connecting to datastore...");
      //replace following with your library server, userid, pasword
      dsDL.connect("LIBXDB2","FRNADMIN","PASSWORD","");
      System.out.println("datastore connected");

      DKBlobDL axdo = new DKBlobDL(dsDL);
      DKPidXDODL  apid = new DKPidXDODL();
      apid.setPartId(partId);
      apid.setId(itemId);
      axdo.setPid(apid);
      axdo.setRepType(repType);
      System.out.println("repType=" + axdo.getRepType());
      System.out.println("objectType=" + axdo.getObjectType());
      System.out.println("itemid=" + apid.getId());
      System.out.println("partId=" + apid.getPartId());
```

```
                boolean flag = axdo.isCategoryOf(DK_INDEXED_OBJECT);
                boolean flag2 = axdo.isCategoryOf(DK_MEDIA_OBJECT);
                System.out.println("isIndexedObject?=" + flag);
                System.out.println("isMediaObject?=" + flag2);
                if (flag)
                {
                 DKSearchEngineInfoDL srch = (DKSearchEngineInfoDL)
                                             axdo.getExtension("DKSearchEngineInfoDL");
                 System.out.println("  serverName=" + srch.getServerName());
                 System.out.println("  textIndex=" + srch.getTextIndex());
                 System.out.println("  timeStamp=" + srch.getSearchTimestamp());
                 System.out.println("  searchIndex=" + axdo.getSearchIndex());
                 System.out.println("  indexedState=" +
                                    axdo.retrieveObjectState(DK_INDEXED_OBJECT));
                }
                if (flag2)
                {
                 DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
                                             axdo.getExtension("DKMediaStreamInfoDL");
                 System.out.println(" mediaformat=" + media.getMediaFormat());
                 System.out.println(" mediaBitRate=" + media.getMediaBitRate());
                 System.out.println(" mediastate(dynamic)=" +
                                    axdo.retrieveObjectState(DK_MEDIA_OBJECT));
                }

                System.out.println("before retrieve......");
                System.out.println(" lob length=" + axdo.length());
                System.out.println(" size=" + axdo.getSize());
                System.out.println(" createdTimestamp=" + axdo.getCreatedTimestamp());
                System.out.println(" updatedTimestamp=" + axdo.getUpdatedTimestamp());

                System.out.println("about to call retrieve()");
                axdo.retrieve();
                System.out.println("after retrieve......");
                System.out.println(" lob length=" + axdo.length());
                System.out.println(" size=" + axdo.getSize());
                System.out.println(" mimeType=" + axdo.mimeType());
                System.out.println(" createdTimestamp=" + axdo.getCreatedTimestamp());
                System.out.println(" updatedTimestamp=" + axdo.getUpdatedTimestamp());
                System.out.println("affiliatedTyp=" + axdo.getAffiliatedType());
                if (axdo.getAffiliatedType() == DK_ANNOTATION)
                {
                 DKAnnotationDL ann = (DKAnnotationDL)(axdo.getExtension("DKAnnotationDL"));
                 System.out.println("affil pageNumber=" + ann.getPageNumber());
                 System.out.println("affil X=" + ann.getX());
                 System.out.println("affil Y=" + ann.getY());
                }
                System.out.println("about to do open()...");
                axdo.setInstanceOpenHandler("notepad", true); //default use notepad in NT
                int cc = axdo.getContentClass();
                if ( cc == DK_CC_GIF)
                 axdo.setInstanceOpenHandler("lviewpro ", true);  //use lviewpro in NT
                else if (cc == DK_CC_AVI)
                 axdo.setInstanceOpenHandler("mplay32 ", true);   //use mplay32 in NT
                else if (cc == DK_CC_IBMVSS)
                 axdo.setInstanceOpenHandler("iscoview ", true);  //use iscoview in NT
                axdo.open();

                dsDL.disconnect();
                System.out.println("datastore disconnected");
               }
               catch (DKException exc)
               {
                System.out.println("Exception name " + exc.name());
                System.out.println("Exception message " + exc.getMessage());
```

```
            exc.printStackTrace();
          }
        }
      }
```

Querying the datastore

You can issue a query to a datastore and get back results in the form of a dkResultSetCursor or DKResults. You can create a query object to represent your query, then invoke the execute method or evaluate method of the query object. With the help of its datastores, the query object performs query processing tasks, such as preparing and executing a query, monitoring the status of a query execution, and storing the results.

There are three types of query objects, Parametric, Text and Combined. The Combined query is composed of Text and Parametric queries.

The datastore uses two ways to run a query: execute and evaluate. Execute returns dkResultSetCursor, evaluate returns DKResults. The dkResultSetCursor is used to handle large result sets as well as to perform delete and update operations on the current position of the result set cursor. You can use the fetchNextN method to fetch a group of objects into a collection.

The dkResultSetCursor can also be used to re-execute a query by calling the close() and open() methods. This is described in “Result set cursor” on page 114.

The DKResults contains all of the results from the query. You can iterate over the items in the collection either forwards or backwards. This collection is also queryable and can be used as a scope for another query.

See “Queryable collection” on page 121 and “Combined query” on page 123 for more information.

Differences between dkResultSetCursor and DKResults

The difference between a `dkResultSetCursor` and a `DKResults` collection is the following:

- The `dkResultSetCursor` works like a datastore cursor; it can be used for large result sets since the DKDDO it contains are fetched one at a time. It can also be used to re-execute a query to get the latest results.
- The `DKResults` contains the entire result set and supports a bidirectional iterator.

Parametric Query

This section explains the ways in which you use the parametric query function.

Formulating a parametric query

The following example is a query string representing a query against the index class GP2DLS2. The condition of the query is to search for all documents or folders with an attribute of `DLSEARCH_DocType <> null`. The maximum number of results returned is limited to five. The content is set to ″yes″ so that contents of the document or folder are returned.

The example specifies that Digital Library use dynamic SQL for this query and that folders and documents be searched. If the attribute name has more than one word or is in a DBCS language, it should be enclosed in single quotes. If the attribute value is in DBCS, it should be enclosed in double quotes.

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS2," +
                     "MAX_RESULTS=5," +
                     "COND=(DLSEARCH_DocType <> null));" +
                     "OPTION=(CONTENT=YES;"       +
                     "TYPE_QUERY=DYNAMIC;" +
                     "TYPE_FILTER=FOLDERDOC)";
```

Formulating a parametric query on multiple criteria

The parametric query allows you to specify more than one search criteria. The following example shows how to specify a query against two index classes.

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS1,MAX_RESULTS=3," +
                     "COND=(DLSEARCH_DocType <> null);" +
                     "INDEX_CLASS=GP2DLS1,MAX_RESULTS=8,"
                     "COND=('First name'==\"Robert\"));" +
                     "OPTION=(CONTENT=YES;" +
                     "TYPE_QUERY=DYNAMIC;" +
                     "TYPE_FILTER=FOLDERDOC)";
```

Executing a parametric query

The datastore provides a method to create a query object. Use the query object to execute the query and obtain the results. The following example shows how to create a parametric query object and execute the query. Once the query is executed, the results are returned in a `DKResults` collection.

A complete sample, SamplePQryDL.java, is available in the samples directory.

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS3," +
             "COND=('last name' == \"SUMMERS\"));" +
             "OPTION=(CONTENT=YES;"       +
                      "TYPE_QUERY=DYNAMIC;" +
                      "TYPE_FILTER=DOC)";
DKNVPair parms[] = null;
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","USERID","PASSWORD");
DKDDO item = null;

...

dkQuery pQry = dsDL.createQuery(cmd,DK_PARAMETRIC_QL_TYPE,parms);
pQry.execute(parms);
DKResults pResults = (DKResults)pQry.result();
dkIterator pIter = pResults.createIterator();
while (pIter.more() == true)
{
 item = (DKDDO)pIter.next();
 // Process DKDDO
}
dsDL.disconnect();
```

Executing a parametric query from the datastore

The datastore provides a method to execute a query. The following example shows how to execute a parametric query against the datastore. Once the query is executed, the results are returned in a `dkResultSetCursor` object.

A complete sample, `TExecuteDL.java`, is available in the samples directory.

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS4);" +
                      "OPTION=(CONTENT=YES;"       +
                      "TYPE_QUERY=DYNAMIC;" +
                      "TYPE_FILTER=FOLDERDOC)";
DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","USERID","PASSWORD");
...

dkResultSetCursor pCur = dsDL.execute(cmd,DK_PARAMETRIC_QL_TYPE,parms);
while (pCur.isValid()) {
       item = pCur.fetchNext();
       // Process DKDDO
      }
pCur.destroy();        //done using the cursor
dsDL.disconnect();
```

Evaluating a parametric query from the datastore

The datastore provides a method to evaluate a query. The following example shows how to evaluate a parametric query against the datastore. Once the query is executed, the results are returned in a `DKResults` collection.

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS5," +
                      "COND=((DLSEARCH_Date >= \"1995\") AND " +
                       "(DLSEARCH_Date <= \"1996\")));" +
                       "OPTION=(CONTENT=NO;"       +
                       "TYPE_QUERY=DYNAMIC;" +
                       "TYPE_FILTER=FOLDERDOC)";
DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","USERID","PASSWORD");


DKResults pResults = (DKResults)dsDL.evaluate(cmd,DK_PARAMETRIC_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more()) {
       item = (DKDDO)pIter.next();
       // Process DKDDO
      }
dsDL.disconnect();
```

Text Query

This section explains the ways in which you use the text query function.

Formulating a text query

The following example is a query string representing a query against the text index TMINDEX. The criteria for the query is to search for all text documents with the word ″UNIX″ or ″member″. The maximum number of results returned is five.

```
String cmd = "SEARCH=(COND=(UNIX OR member));" +
             "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";
```

Formulating a text query on multiple indexes

The text query allows a user to specify that a search is to be performed against more than one index. The following example shows how to specify a query against two indexes.

Important: If you specify more than one text search index in the query, the indexes must be the same type. For example, you can specify two precise indexes in the query, but you cannot specify a precise index and a linguistic index within the query.

```
String cmd = "SEARCH=(COND=(UNIX OR member));" +
             "OPTION=(SEARCH_INDEX=(TMINDEX,TMINDEX2);MAX_RESULTS=5)";
```

Executing a text query

The datastore provides a method to create a query object. Use the query object to execute the query and obtain the results. The following example shows how to create a text query object and execute a query. Once the query is executed, the results are returned in a `DKResults` collection.

A complete sample, `SampleTQryTS.java`, is available in the samples directory.

```
String cmd = "SEARCH=(COND=('UNIX operating' AND system));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";

DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreTS dsTS;
dsTS.connect("TM","", ' ');

...

dkQuery pQry pQry = dsTS.createQuery(cmd,DK_TEXT_QL_TYPE,parms);
pQry.execute(parms);
DKResults pResults = (DKResults)pQry.result();
dkIterator pIter = pResults.createIterator();
while (pIter.more()) {
       item = (DKDDO)pIter.next();
       // Process DKDDO
      }
dsTS.disconnect();
```

Executing a text query from the datastore

The datastore provides a method to execute a query. The following example shows how to execute a text query against the datastore. Once the query is executed, the results are returned in a `dkResultSetCursor` object.

A complete sample, `TExecuteTS.java`, is available in the samples directory.

```
String cmd = "SEARCH=(COND=({web site}));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";
DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreTS dsTS;
dsTS.connect("TM","", ' ');
...

dkResultSetCursor pCur = dsTS.execute(cmd,DK_TEXT_QL_TYPE,parms);
while (pCur.isValid()) {
       item = pCur.fetchNext();
       // Process DKDDO
      }
pCur.destroy();        //done using the cursor
dsTS.disconnect();
```

Evaluating a text query from the datastore

The datastore provides a method to evaluate a query. The following example shows how to evaluate a text query against the datastore. Once the query is executed, the results are returned in a `DKResults` collection.

```
String cmd = "SEARCH=(COND=($MC=*$ UN*));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";

DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreTS dsTS;
dsTS.connect("TM","", ' ');
...

DKResults pResults = (DKResults)dsTS.evaluate(cmd,DK_TEXT_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more()) {
       item = (DKDDO)pIter.next();
       // Process DKDDO
      }
dsTS.disconnect();
```

Getting match highlighting information for each text query result item

This code sample retrieves match highlighting information for each text query result item during a text query. The MATCH_INFO option is set to YES, indicating that the match highlighting information will be returned. The MATCH_DICT option specifies whether the highlighting information will be obtained using a dictionary. Because the MATCH_DICT option is set to NO, the dictionary will not be used to retrieve highlighting information. The match information is returned in the attribute DKMATCHESINFO in the DKDDO returned from a text query. The value of the attribute DKMATCHESINFO will be a DKMatchesInfoTS object.

Important: This process is time consuming because the document is retrieved from DL datastore and analyzed linguistically, and potential matches are determined. These processes will have an impact on the performance of a text query.

```
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;

dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
String cmd = "SEARCH=(COND=('UNIX operating' AND system));" +
             "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5;
                      MATCH_INFO=YES;MATCH_DICT=NO)";

...

pCur = dsTS.execute(cmd,DK_TEXT_QL_TYPE,parms);
DKDDO item = null;
DKMatchesInfoTS pMInfo = null;
DKMatchesDocSectionTS pMSect = null;
DKMatchesParagraphTS pMPara = null;
DKMatchesTextItemTS  pMText = null;
int i = 0;
int j = 0;
int k = 0;
int m = 0;
int lCCSID = 0;
int lLang = 0;
int lOffset = 0;
int lLen = 0;
int numberSections = 0;
int numberParagraphs = 0;
int numberTextItems = 0;
int numberNewLines = 0;
String strDoc = "";
String strSection = "";
String strText = "";
```

```
Object anyObj = null;
while (pCur.isValid())
{
 item = pCur.fetchNext();
 if (item != null)
 {
    // Process the DKDDO
    for (i = 1; i <= item.dataCount(); i++)
    {
     anyObj = item.getData(i);
     if (anyObj instanceof String)
     {
      ...
     }
     else if (anyObj instanceof Integer)
     {
      ...
     }
     else if (anyObj instanceof Short)
     {
      ...
     }
     else if (anyObj instanceof DKMatchesInfoTS)
     {
      pMInfo = (DKMatchesInfoTS)anyObj;
      // process the Match Hightlighting information
      if (pMInfo != null)
      {
       strDoc = pMInfo.getDocumentName();
      numberSections = pMInfo.numberOfSections();
       // loop thru document sections
       for (j = 1; j <= numberSections; j++)
       {
         pMSect = pMInfo.getSection(j);
         strSection = pMSect.getSectionName();
         numberParagraphs = pMSect.numberOfParagraphs();
         // loop thru section paragraphs
        for (k = 1; k <= numberParagraphs; k++)
        {
          pMPara = pMSect.getParagraph(k);
          lCCSID = pMPara.getCCSID();
          lLang = pMPara.getLanguageId();
          numberTextItems = pMPara.numberOfTextItems();
          // loop thru paragraph text items
         for (m = 1; m <= numberTextItems; m++)
         {
           pMText = pMPara.getTextItem(m);
           strText = pMText.getText();
           // if match found in text item get offset and
           // length of match in text item
           if (pMText.isMatch() == true)
          {
           lOffset = pMText.getOffset();
            lLen = pMText.getLength();
           }
           numberNewLines = pMText.numberOfNewLines();
          }
         }
        }
       }
      }
     }
    }
}
dsTS.disconnect();
```

Getting match highlighting information for a particular text query result item

This code sample retrieves the match highlighting information for a specific item returned from a text query. The match information contains the text of the document and the highlighting information for every match of the corresponding query. The `resultSetCursor` passed into this routine must be in an open state.

Important: This process is time consuming because the document is retrieved from DL datastore and analyzed linguistically, and potential matches are determined. These processes will have an impact on the performance of a text query.

```
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;

dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
String cmd = "SEARCH=(COND=('UNIX operating' AND system));" +
             "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";

...

pCur = dsTS.execute(cmd);
DKDDO item = null;
Object anyObj = null;
DKMatchesInfoTS pMInfo = null;
DKMatchesDocSectionTS pMSect = null;
DKMatchesParagraphTS pMPara = null;
DKMatchesTextItemTS  pMText = null;
int i = 0;
int j = 0;
int k = 0;
int m = 0;
int lCCSID = 0;
int lLang = 0;
int lOffset = 0;
int lLen = 0;
int numberSections = 0;
int numberParagraphs = 0;
int numberTextItems = 0;
int numberNewLines = 0;
String strDoc;
String strSection;
String strText;
String strDID = "";
String strXNAME = "";
String strDataName = "";
DKPid pid = null;
while (pCur.isValid())
{
 item = pCur.fetchNext();
 if (item != null)
 {
  pid = item.getPid();
  // Process the DKDDO
  for (i = 1; i <= item.dataCount(); i++)
  {
   anyObj = item.getData(i);
   strDataName = item.getDataName(i);
   if (strDID.equals(""))
   {
    strDID = pid.getId();
   }
   if (strXNAME.equals(""))
   {
    strXNAME = p.getObjectType();
```

```
    }
    ...
   }
   // Get Match Highlighting Information
   pMInfo = dsTS.getMatches(pCur,strDID,strXNAME,false);
   strDID = "";
   strXNAME = "";
   if (pMInfo != null)
   {
    strDoc = pMInfo.getDocumentName();
    numberSections = pMInfo.numberOfSections();
    // loop thru document sections
    for (j = 1; j <= numberSections; j++)
    {
     pMSect = pMInfo.getSection(j);
     strSection = pMSect.getSectionName();
     numberParagraphs = pMSect.numberOfParagraphs();
     // loop thru section paragraphs
     for (k = 1; k <= numberParagraphs; k++)
     {
      pMPara = pMSect.getParagraph(k);
      lCCSID = pMPara.getCCSID();
      lLang = pMPara.getLanguageId();
      numberTextItems = pMPara.numberOfTextItems();
      // loop thru paragraph text items
      for (m = 1; m <= numberTextItems; m++)
      {
       pMText = pMPara.getTextItem(m);
       strText = pMText.getText();
       // if match found in text item get offset and
       // length of match in text item
       if (pMText.isMatch() == true)
       {
        lOffset = pMText.getOffset();
        lLen = pMText.getLength();
       }
       numberNewLines = pMText.numberOfNewLines();
      }
     }
    }
   }
  }
 }
}
dsTS.disconnect();
```

Result set cursor

The `dkResultSetCursor` is a datastore cursor that manages a virtual collection of DDO objects. This means that the collection does not materialize until you fetch an element from it. The collection is the resulting set of a query submitted to the datastore.

Important: When you are finished using the cursor, call the destroy method. This method closes the cursor, preventing memory leaks.

Open and close the result set cursor to re-execute the query

When the result set cursor is created, it is in an open state. To re-execute the query, close and reopen the cursor.

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS4);" +
                      "OPTION=(CONTENT=YES;"       +
                               "TYPE_QUERY=DYNAMIC;" +
                               "TYPE_FILTER=FOLDERDOC)";
```

```
DKNVPair parms[] = null;
...

dkResultSetCursor pCur = dsDL.execute(cmd,DK_PARAMETRIC_QL_TYPE,parms);

pCur.close();
pCur.open();          //re-execute the query
```

Set position and get position in a result set cursor

The result set cursor allows the user to set the cursor position and get the current position. In the following example, a query is created and executed. Inside the WHILE loop, the cursor position is set to the first (or next) valid position. Then a DDO is fetched from the current position. A null is returned from `fetchObject` if the cursor is past the last item in the result.

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS4);" +
                      "OPTION=(CONTENT=YES;"       +
                               "TYPE_QUERY=DYNAMIC;" +
                               "TYPE_FILTER=FOLDERDOC)";
DKNVPair parms[] = null;
DKDDO item = null;
int i = 0;
...

dkResultSetCursor pCur = dsDL.execute(cmd,DK_PARAMETRIC_QL_TYPE,parms);
while (pCur.isValid()) {
       pCur.setToNext();
       item = pCur.fetchObject();
       if (item != null) {
           i = pCur.getPosition();
         }
       }
```

Another way to do this is:

```
Object a = null;
pCur = dsDL.execute(cmd,DK_PARAMETRIC_QL_TYPE,parms);
while (pCur.isValid()) {
       pCur.setPosition(DK_NEXT,a);
       item = pCur.fetchObject();
       if (item != null) {
           i = pCur.getPosition();
          }
       }
```

You can use the RELATIVE positioning. In the following example, every other item in the result set cursor is skipped.

```
Object a = null;
pCur = dsDL.execute(cmd,DK_PARAMETRIC_QL_TYPE,parms);
a = new Integer(2);
while (pCur.isValid()) {
       pCur.setPosition(DK_RELATIVE,a);   // move cursor 2 positions forward
       item = pCur.fetchObject();         // from the current podition
       if (item != null) {                // (relative)
           i = pCur.getPosition();
       }
     }
```

Creating a collection from a result set cursor

The result set cursor allows you to supply a collection to be filled in with a specified number of items from the result set cursor. In the following example, all items from the result set cursor are fetched into the sequential collection. A zero in the first

parameter of fetchNextN indicates that all items of the cursor will be put into the collection. The first parameter specifies how many items to put into the collection. If fItems is true, at least one item was returned.

```
DKSequentialCollection seqColl = new DKSequentialCollection();
boolean fItems = false;
int how_many = 0;
fItems = pCur.fetchNextN(how_many,seqColl);
```

Retrieving a document or folder

To retrieve a document from DKDatastoreDL, you need to know its index-class name and item-id. You also need to associate the DDO to a datastore and establish a connection.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVR2","userId1","passwd1");

DKPid pid = new DKPid();
pid.setObjectType("GRANDPA");          // set the index-class name it belongs to
pid.setId("LN#U5K6ARLGM3DB4");         // set the item-id
DKDDO ddo = new DKDDO(dsDL,pid);       // create a DDO with pid and associated to dsDL
ddo.retrieve();                        // retrieve it
```

In case the DDO is a result of a parametric query with the query option CONTENT=NO, the DDO is empty (does not have the attribute values). However, all information required to retrieve it is already set. To retrieve the DDO, call

```
ddo.retrieve();
```

After a call to retrieve, the DDO will have all of its attribute value set to the value of persistent data stored in the datastore. If the document has parts, the DKPARTS attribute is set to a DKParts object. However, the content of each part in this collection is not retrieved yet. Since a part may be big, it is not desirable to retrieve all of them into memory at once. It is better to retrieve the part you want explicitly. See “Retrieving parts” for an example of how to retrieve a part.

Retrieving parts

Once you have retrieved a DDO, you can get its parts stored in DKPARTS attribute,

```
DKParts parts = (DKParts) ddo.getDataByName(DKPARTS);
```

The above code assumes that the DKPARTS attribute exists. It throws an exception if the attribute does not exist. See also “Retrieving a folder” on page 117 for another example of extracting an attribute value by getting the data_id first and testing it for zero.

To retrieve each part, you need to create an iterator to iterate over the collection and retrieve each part one by one. See also “Creating and using DKPARTS attribute” on page 91.

```
// create iterator and process the part collection member one by one
if (parts != null) {
    DKBlobDL blob;
    dkIterator iter = parts.createIterator();
    while (iter.more()) {
           blob = (DKBlobDL) iter.next();
           if (blob != null) {
               blob.retrieve();   // retrieve the blob's content
               blob.open();       // other processing, as needed
```

```
            ....
          }
       }
}
```

Similar to DDO results of a parametric query, each part XDO inside the DKParts collection is empty (does not have its content set). However, it has all the information needed for its retrieval; this part is ready to be retrieved. To bring its content and related information into memory, call

```
blob.retrieve();
```

Retrieving a folder

Retrieving a folder DDO is the same as retrieving a document DDO. After retrieve, the folder DDO will have all of its attributes set, including a special attribute DKFOLDER. This attribute value is set to a DKFolder object, a collection of DDO members of this folder. Like parts in DKParts, these member DDO are relatively empty, they only contain enough information to retrieve them. You can retrieve it whenever you need.

```
data_id = ddo.dataId(DKFOLDER);  // get DKFOLDER data-id
if (data_id == 0)                // folder not found
    throw new DKException(" folder data-item not found");

DKFolder fCol = (DKFolder) ddo.getData(data_id); // get the folder collection

// create iterator and process the DDO collection member one by one
if (fCol != null) {
    DKDDO item;
    dkIterator iter = fCol.createIterator();
    while (iter.more()) {
           item = (DKDDO)  iter.next();
           if (item != null) {
               item.retrieve();  // retrieve the member DDO
               ....              // other processing
           }
        }
 }
```

See also “Creating and using DKFOLDER attribute” on page 92.

Creating, updating, and deleting documents or folders

This section describes the processes involved in creating, updating, and deleting documents and folders.

Creating a new document

To create a new document and save its persistent data in the datastore, you need to create a DDO with all its attributes and other information set, except its item-id. The item-id is assigned and returned by the datastore. Some of the previous examples can be combined to provide complete steps:

```
// step 1: create a datastore and connect to it
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVR2","userId1","passwd1");

// step 2: create a document (or folder) DDO
//         and set all its attributes and other required information
//         See the section on "Using DDO"
```

```
DKPid pid = new DKPid();
pid.setObjectType("GRANDPA");    // set the index-class name it belongs to
DKDDO ddo = new DKDDO(dsDL,pid); // create a DDO with pid and
...                              // associate it to dsDL

// step 2.a: add attributes according to index-class GRANDPA
Object obj, vstr;
Boolean yes = new Boolean(true);
Boolean  no = new Boolean(false);

short data_id = cddo.addData("Title");  // add new attribute "Title"
vstr = new Short(DK_VSTRING);
...                                     // add type property VSTRING
cddo.addDataProperty(data_id,DK_PROPERTY_TYPE,vstr);
...                                     // add nullable property
cddo.addDataProperty(data_id,DK_PROPERTY_NULLABLE,no);

data_id = cddo.addData("Subject");  // add new attribute "Subject"
cddo.addDataProperty(data_id,DK_PROPERTY_TYPE,vstr);
cddo.addDataProperty(data_id,DK_PROPERTY_NULLABLE,yes);

// add some more attributes as necessary
//  ....

// step 2.b: add DKPARTS attribute
DKParts parts = new DKParts();  // create a new DKParts, collection of parts
DKBlobDL blob = new DKBlobDL(dsDL);   // create a new XDO blob
DKPidXDODL pidXDO = new DKPidXDODL(); // create Pid for this XDO object

pidXDO.setPartId(5);                          // set part number to 5
blob.setPid(pidXDO);                          // set the Pid for the XDO blob
blob.setContentClass(DK_CC_GIF);              // set content class type GIF
blob.setRepType(DK_REP_NULL);                 // set rep type for the part
blob.setContentFromClientFile("choice.gif");  // set the blob's content
blob.setInstanceOpenHandler("xv");            // the viewer program on AIX

parts.addElement(blob);               // add the blob to the parts collection

....                                          // create and add some more blobs to
....                                          // the collection as necessary

// create DKPARTS attribute and sets it to refer to the DKParts object
short data_id = ddo.addData(DKPARTS);  // add attribute "DKParts"
obj = new Short(DK_COLLECTION_XDO);    // add type property
ddo.addDataProperty(data_id,DK_PROPERTY_TYPE,obj);
ddo.addDataProperty(data_id,DK_PROPERTY_NULLABLE,yes); // add nullable property
ddo.setData(data_id, parts);           // sets the attribute value

// step 2.c: sets the item type : document
obj = new Short(DK_DOCUMENT);
ddo.addProperty(DK_PROPERTY_ITEM_TYPE, obj);

// step 3: make it persistent; add it to the datastore
ddo.add();                             // document created in datastore
```

After the last step, you will have a new document with the above information created in the datastore. When a document DDO is added to a datastore, all its attributes are added, including all parts inside DKParts collection. The same rule applies to adding a folder DDO, the DKFolder collection members are added to the datastore as component of a Digital Library folder. Recall that a Digital Library folder actually contains a table-of-contents of its members, which are existing documents and folders. Therefore, all folder members must be created first in the datastore before you can add a folder DDO.

You can add the same document to a different datastore of the same type, provided that you adjust the information accordingly. To add this document to the server ″LIBSRVRN″, which has an index-class ″GRANDPA2″ with the same structure as ″GRANDPA″, do the following steps:

```
// create datastore and connect to LIBSRVRN
DKDatastoreDL dsN = new DKDatastoreDL();
dsN.connect("LIBSRVRN","userIdN","passwdN");

// update the Pid
pid = ddo.getPid();
pid.setObjectType("GRANDPA2");      // set the new index-class
pid.setId("");                      // blank the item-id
pid.setDatastoreName("LIBSRVRN");   // set the new datastore name
ddo.setPid(pid);                    // update the Pid
ddo.setDatastore(dsN);              // re-associate it with dsN
ddo.add();                          // add it
```

Updating a document or a folder

Updating a document is quite simple, you need to have the item-id and object-type set. Then, update the desired attributes. For example, set another value for the attribute or add more parts to DKParts collection. Finally, call the update method to have the update reflected in the datastore, for example:

```
// update the value of attribute Title
String newTitle = "Guess who is behind all this";
short data_id = ddo.getDataByName("Title");
ddo.setData(data_id, newTitle);
ddo.update();
```

After the call to update, the value of attribute ″Title″ in the datastore is updated. The parts in this document are not updated unless their content changed. The associated datastore and its connection must be valid when you call the update method.

Updating a folder DDO takes similar steps. You need to update some attribute values, or add or remove some elements from DKFolder, then call the update method.

Updating parts

The collection of parts in a document is represented as a DKParts object. DKParts is a subclass of DKSequentialCollection. In addition to inheriting the sequential collection functions, it has two additional members to add and remove a part from the collection and have it reflected in the datastore immediately. The document must already exists in the datastore.

Add and remove a member: Given a document DDO, you could use the following code fragments to add a part to this document immediately.

```
DKDDO ddo = new DKDDO();            // a document DDO
DKBlobDL newPart = new DKBlobDL();  // a new part to be added
....                                // ddo and newPart are
....                                // initialized somewhere along the line
DKParts parts = (DKParts) ddo.getDataByName(DKPARTS); // get DKParts
parts.addMember(ddo, newPart        // assume none of these values are NULL
```

The last statement has the same effect as:

```
newPart.add();              // add the part to datastore, into the
                            // specified document
parts.addElement(newPart);  // add newPart into this collection
```

The newPart must have its Pid set to make it work.

Similarly, to remove newPart from the collection and the datastore, enter

```
parts.removeMember(ddo, newPart);
```

which is equivalent to:

```
dkSequentialIterator iter = (dkSequentialIterator) parts.createIterator();
....
newPart = (DKBlobDL) iter.at();     // assume newPart is not NULL
parts.removeElement(iter);          // remove the part pointed by iter
newPart.del();                      // delete the part from this document
                                    // the copy in memory unchanged
```

To compare with the same method in DKFolder, the removeMember() method in DKParts actually deletes the persistent copy of the part in the datastore. In DKFolder, this method only removes the item from the folder table-of-contents; it does not actually delete the item. See “Updating folders”.

Differences with update() on a document DDO: Add and remove member methods on DKParts provide conveniences for adding and removing a part in the collection and in the datastore. Compared to the update() method in a document DDO, add and remove member are faster, too. The update() method on a DDO updates all of the attributes in the DDO, including DKParts and all of its member that changed. The steps are:

```
....
// get DKParts, assume it exists and not null
DKParts parts = (DKParts) ddo.getDataByName(DKPARTS);
parts.addElement(newPart);      // add a new part to parts
ddo.update();                   // updates the whole ddo
....
```

As seen in the above example, these latter steps are longer than the previous one; although the update DDO has its own good use, too.

Updating folders

The collection of documents and folders in a Digital Library folder is represented as a DKFolder object. In the datastore, a folder holds a table of contents referring to its object members instead of keeping all actual objects.

DKFolder is a subclass of DKSequentialCollection. In addition to inheriting the sequential collection functions, it has two additional members to add and remove a member (be it a document or a folder) from the collection and have it reflected in the datastore immediately. The removed or added document or folder must already exists in the datastore.

Add and remove a member: Given a folder DDO, use the following code fragments to add another document or folder DDO to it:

```
DKDDO folderDDO = new DKDDO();  // a folder DDO
DKDDO newMember = new DKDDO();  // new DDO to be added as a member of this folder
....                            // folderDDO and newMember are
....                            // initialized somewhere along the line
// get DKFolder, assume it exists, and the value not null
DKFolder folder = (DKFolder) folderDDO.getDataByName(DKFOLDER);
folder.addMember(folderDDO, newMember);
```

The newMember and folderDDO must exist in the datastore for it work.

The last statement has the same effect as:

```
folder.addElement(newMember);  // add newMember to this collection
folderDDO.update();            // reflects it in the datastore
```

Similarly, to remove `newMember` from the collection and the datastore:

```
folder.removeMember(folderDDO, newMember);
```

which is equivalent to a longer steps:

```
dkSequentialIterator iter = (dkSequentialIterator) folder.createIterator();
....
newMember = (DKDDO) iter.at();
folder.removeElementAt(iter);   // remove newMember from this collection
folderDDO.update();             // reflects it in the datastore
```

Important: Removing a member from the folder only removes that member from the folder table of contents. This removal method does not delete the member from memory or from the datastore.

Differences with update() on a folder DDO: Add and remove member methods on DKFolder provide conveniences for adding and removing a document or folder in the collection and in the datastore. Compared to the `update()` method in a folder DDO, add and remove members are faster. The `update()` method on a DDO updates all of the attributes in the DDO, including DKFolder and all of its members, whereas add and remove member operations only involve adding/removing one particular member to/from the folder table of contents. However, the method update DDO has its own good use, too.

Deleting a document or a folder

You only need to call the `del()` method in the DDO to delete the persistent data from the datastore.

```
ddo.del();
```

The DDO must have its item-id and object-type set, and a valid connection to a datastore. The above statement can be used to delete a folder, too. Only the persistent data is deleted, the in-memory copy of the DDO does not change. Therefore, you can add this DDO back to the same or different datastore later. See “Creating a new document” on page 117.

Queryable collection

A queryable collection is a collection that can be queried further, thus providing a smaller set, more refined results. A concrete implementation of a queryable collection is DKResults class. DKResults is a collection of DDO, the results of query evaluation.

Getting the result of a query

The following example illustrates how to submit a parametric query and get results:

```
// establish a connection
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVR2","userId1","passwd1");

// create a query object
String query1 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> null));";
```

```
DKParametricQuery pq =
    (DKParametricQuery) dsDL.createQuery(query1,DK_PARAMETRIC_QL_TYPE, null);
pq.execute();
DKResult rs = (DKResults) pq.result();
```

The results are in an `rs` object, which is an instance of DKResults, a subclass of dkQueryableCollection. You can use previous code examples to process the collection and get the DDO out. See “Collections and iterators” on page 84.

Evaluating a new query

Given a result from the above query, you can submit another query to this result to refine the current result to get a smaller set. Its concept is similar to using the current result as a scope for the new query. Here are the steps to do it:

```
String query2 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Subject == 'Mystery'));";
Object obj = rs.evaluate(query2,DK_PARAMETRIC_QL_TYPE, null);
....
```

`obj` is a DKResults object containing the refined results. The totals of both queries would be equivalent to:

```
"SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> null AND Subject == 'Mystery'));"
```

You can repeat this step over and over again to get satisfactory results. Once you started with a parametric query, the subsequent queries must be of the same type, otherwise it will probably return a null result.

The same procedure works for text queries also. The following is an example for text queries:

```
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect("barney","6011",DK_CTYP_TCPIP);

String tquery1 = "SEARCH=(COND=(IBM));OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery tq =
    (DKTextQuery) dsTS.createQuery(tquery1,DK_TEXT_QL_TYPE, null);
tq.execute();
DKResults trs = (DKResults) tq.result();

String tquery2 = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
Object obj = trs.evaluate(tquery2,DK_TEXT_QL_TYPE, null);
```

`obj` is a DKResults object containing the results of the totals of both queries:

```
   "SEARCH=(COND=(IBM AND Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
```

Using queryable collection instead of combined query

Evaluating a queryable collection bears some similarities with some other features in this class library. One such feature is the combined query. The combined query provides the flexibility to submit a combination of parametric and text queries, with or without scopes. However, all of these queries must be submitted at once, not in piece-meal like evaluating a queryable collection.

See “Combined query” on page 123. The result of a combined query is a DKResults, so theoretically you can evaluate another parametric query against it; although in the current implementation, it may not always work.

Programming tips

Evaluating a queryable collection with subsequent queries provide a flexibility to refine the results of a previous query, step by step, until you get a satisfactory final result. This is quite useful for browsing the content of a datastore dynamically and formulating the next query that depends on the previous results. However, if you know the total query in advance, it would be more efficient to submit the total query once instead of in pieces.

Combined query

As suggested by its name, a combined query allows you to execute a combination of parametric and text queries, with or without a scope. The final result is an intersection between the scopes and the results of each query. Therefore, if you are not careful in formulating the query and including scopes, these collections may not intersect at all and you will end up with an empty result. If there is at least one parametric and one text query, the resulting DDO returned in DKResults has an attribute DKRANK, which signifies the highest rank of the matching part belonging to this document.

Important: For each query in a combined query, you must use a dedicated connection to the search engine; you cannot route multiple queries through the same connection.

Combined parametric queries and text queries

To execute a combined query with one parametric, one text query, without any scope, you need to create the combined query object, the parametric and text queries, and pass the latter two queries as input parameters to be executed by the combined query. For example:

```
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVR2","userId1","passwd1");

DKDatastoreTS dsTS;
dsTS.connect("TM","",' ');  // TM is a local alias for
...                         // the Text Search Engine server

...                         // create a parametric query
String pquery =  "SEARCH=(INDEX_CLASS=GRANDPA,COND=(DLSEARCH_Date > 1994));";
DKParametricQuery pq =
    (DKParametricQuery) dsDL.createQuery(pquery,DK_PARAMETRIC_QL_TYPE, null);

// create a text query
String tquery = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery tq =
    (DKTextQuery) dsTS.createQuery(tquery,DK_TEXT_QL_TYPE, null);

// create a combined query
DKCombinedQuery cq = new DKCombinedQuery();

// package the queries in DKNVPair as input parameters
DKNVPair par[] = new DKNVPair[3];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].setName(DK_PARM_END);    // to signal the end of parameter list

// execute the combined query
cq.execute(par);
```

```
// get the results
DKResults res = (DKResults) cq.result();
if (res != null) {
    // process the results
    //   see examples under "Collection and Iterators" and "Query"
    ....
}
```

The last if-statement is necessary to make sure the DKResults object is not null before calling its method.

Using a scope

If you have a DKResults object that you want to use as the scope, you can modify the above code slightly to insert the scope as an additional parameter:

```
DKResults scope;               // assume that this is the scope
                               //  initialized somewhere as a result of
                               //  some parametric query
DKResults tscope;              // assume that this is the scope
                               //  initialized somewhere as a result of
                               //  some text query
....
// package the query in DKNVPair as input parameters
DKNVPair par[] = new DKNVPair[4];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].set(DK_SCOPE_DL, scope);
par[3].set(DK_SCOPE_TS, tscope);
par[4].setName(DK_PARM_END);

// execute the combined query
cq.execute(par);
....
```

The results of a combined query can also be used as a scope of another combined query, and sometimes the results are queryable.

Ranking

If the combined query contains at least one text query, then the DDO in the result has the attribute DKRANKS. This is a transient attribute, it is not stored in the datastore but is computed each time by the text search engine. The value of the rank corresponds to the highest rank of the part in the document that satisfies the text query conditions.

Programming tips

The same observation as in queryable collection also applies here. If you have several parametric queries and scopes, it is more efficient to execute the total query as a single parametric query. This is also true for text queries. The most efficient execution would be if you have one total parametric query and one total text query, without scope. See also “Queryable collection” on page 121.

The query option `"MAX_RESULTS=nn"` limits the number of results returned to the caller. Usually, this option is more applicable to text queries, since the result is sorted by rank in descending order. Thus, if this option is set to ten, for example, it means that the caller only wants the ten highest matching rank of the results. The meaning of this option is different for parametric queries. Since there is no notion of rank, the caller will get the first ten of the results. This result is intersected with the

result from text query. Therefore, when combining a parametric query and text query, it is advisable not to specify this option for the parametric query. Otherwise, the result is less meaningful.

Text Search Engine

Text Search Engine is the product name of the text search engine. Text Search Engine allows you to specify boolean, proximity, GTR, hybrid and free text queries. The query results returned from the text search contain the Digital Library item ID, part number, and ranking information. This information can be used to create a DL XDO that can be used to retrieve the text document contents in Digital Library.

Programming Tip

To prevent the timing out of a Text Search Engine connection, perform a connect, execute your queries and disconnect, rather than leaving a connection open for a long period of time.

The search options CCSID and LANG go together. If one is specified, the other must also be specified. The default CCSID and LANG are specified by the datastore TS options, DK_OPT_TS_CCSID and DK_OPT_TS_LANG. Refer to the *Application Programming Reference* for the list of the datastore options and their descriptions.

You can specify more than one search option for a query term. The search options are separated by commas. An example of multiple search terms is given in “GTR query” on page 126.

If both the SC (single required character) and the MC (sequence of optional characters) search options are specified, the SC search option must be specified first. For example, `$SC=?,MC=*$ U?I*`. For a detailed description of the SC and MC search options, refer to the *Application Programming Reference*.

Boolean query

A boolean query is made up of words and phrases, separated by a boolean operator. A phrase is a sequence of words enclosed in single quotes, to be searched as a literal. In the example below we are searching for all text documents with the word `WWW` or the phrase `Web site` in the `TMINDEX` text search index:

```
String cmd = "SEARCH=(COND=(www OR 'web site'));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Free text query

A free text query is made up of words, phrases or sentences enclosed in braces {}. The words do not need to be adjacent to each other. In the example below we are searching for all text documents with the free text `Web site` in the `TMINDEX` text search index:

```
String cmd = "SEARCH=(COND=({web site}));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Hybrid query

A hybrid query is made up of a boolean query, followed by a free text query. In the example below we are searching for all text documents with the words IBM and UNIX, as well as the free text Web site in the TMINDEX text search index:

```
String cmd = "SEARCH=(COND=(IBM AND UNIX {web site}));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Proximity query

A proximity query relates to a sequence of search arguments found in the same document, paragraph or sentence. In the example below we are searching for all text documents with the phrase rational numbers and the word math in the same paragraph.

```
String cmd = "SEARCH=(COND=($PARA$ {'rational numbers' math}));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Tip: This type of query requires at least two search arguments.

GTR query

A GTR (Global Text Retrieval) query is optimized for double-byte character set (DBCS) languages like Japanese or Chinese, but also supports single-byte character set (SBCS) languages. The following example shows a search for a GTR index for all text documents with the phrase IBM marketing. All double-byte characters should be enclosed in single quotes. The phrase to be searched for should be in the specified character code set and language. The match keyword is set to indicate the degree of similarity for the phrase.

Make sure that the text search datastore options DK_OPT_TS_CCSID (coded character set identifiers) and DK_OPT_TS_LANG (language identifiers) are set properly. The default for DK_OPT_TS_CCSID is DK_CCSID_00850. The default for DK_OPT_TS_LANG is DK_LANG_ENU. These values are used as the global defaults for the text query. You can also enter specific CCSID and LANG information as shown in the following example. You must specify both CCSID and LANG; one value cannot be specified with the other.

```
String cmd = "SEARCH=(COND=($CCSID=850,LANG=6011,MATCH=1$ " +
             "'IBM marketing'));" +
             "OPTION=(SEARCH_INDEX=TMGTRX)";
```

Loading data to be indexed by Text Search Engine

To load data into Digital Library to be indexed by Text Search Engine, a Digital Library index and a text search index must be created.

Before creating a text search index, the text search server must be running. Make sure that your environment is set up properly by running the Digital Library samples TListCatalogDL and TListCatalogTS after the samples have been updated with your server, user ID, etcetera.

To create parts in Digital Library that are indexed by Text Search Engine, refer to “Using XDO” on page 92.

Once the data is loaded into Digital Library, the documents are placed on the document queue. This is done by a method on the DKDatastoreDL called

wakeUpService, which takes a search engine name (`SM`). Once this is done, the user goes into the Digital Library text search administration panel, double-clicks the text search server, double-clicks the text search index, and single-clicks the INDEX button on the explicit notebook page. For more information on text search administration, refer to the Digital Library *System Administration Guide*. This indexes the documents on the document query. After the indexing is complete, you can perform queries against Text Search Engine.

For a complete example refer the sample TLoadSample.java in the samples directory.

Environment settings and component names

- Java
  - Environment setup on Windows NT and AIX (CLASSPATH, LD_LIBRARY_PATH, PATH)
  - Misc
- Package names (to be imported when using this class library)
  - COM.ibm.mm.sdk.common
  - COM.ibm.mm.sdk.server
- Package names (to be imported when using this class library for client/server application)
  - COM.ibm.mm.sdk.common
  - COM.ibm.mm.sdk.client
- Library
  - dksdk.jar
  - DKConstant.txt: constants used in this class library
- Shared objects for AIX
  - libdkdsdl.so
  - libdkdsts.so
  - libdkxdodl.so
  - libdkdsqbic.so
  - libdkdswfs.so
- DLLs for Windows NT
  - dkdsdl.dll
  - dkdsts.dll
  - dkxdodl.dll
  - dkdsqbic.dll
  - dkdswfs.dll

On AIX

Use the `-qalign=packed` compiler option so that the object will align properly. Refer to the sample makefiles in the samples directory for more information.

- set PATH to:

  ```
  export PATH=path/usr/lpp/frn/lib

  where path = your PATH directories.
  ```

- set LIBPATH to:

```
export LIBPATH=path/usr/lpp/frn/lib

where path = your LIBPATH directories.
```

- set LD_LIBRARY_PATH to:

```
export LD_LIBRARY_PATH=path/usr/lpp/frn/lib

where path = your LD_LIBRARY_PATH directories.
```

- set CLASSPATH to:

```
export CLASSPATH=path/usr/lpp/frn/lib/dksdk.jar

where path = your INCLUDE directories.
```

On Windows NT

- set PATH to:

```
set PATH=x:\FRNROOT\DLL

where x = your drive
```

- set CLASSPATH to:

```
set CLASSPATH=x:\FRNROOT\LIB\dksdk.jar

where x = your drive
```

Client/Server in Java

The Digital Library Java API provides a convenient interface to the Java application users. The applications can be located at a local or remote site. Java package resides on both the server side and the client side. Both the client and server packages provide you with the same API.

The client side of Java package communicates with the server side of Java package to access data in Digital Library through the network. Communication between client and server is done by the Java communication package. You do not need to add any additional program for the communication. Java applications can run on the client or server side using the same Java API.

The number of client components is exactly the same as the number of server components. Their public methods also correspond to one another. The client components package name is `COM.ibm.mm.sdk.client`, while the server components package is `COM.ibm.mm.sdk.server`. Thus, when the application program is run on the client side, it needs to import the client package, not the server package.

One additional item to consider in the client side is the remote exception. Although the same API is provided on the client and server, the client package has an additional exception item since it communicates with the server package.

Digital Library datastore connect and disconnect in the client side

The following is a simple Java example to run in the client side. We have already seen this example in the earlier chapter running in the server side. The only difference between this example and the earlier one is the package name to be imported. Also, one additional thing to consider is the remote exception. Since the

client classes communicate with server classes, some exceptions may be triggered by network communication errors. Therefore, the application program needs to catch the remote exception in the client package also.

Connect using the Digital Library library server name, user ID, and password. You can connect to Digital Library by creating a DKDatastoreDL. This example connects to the Digital Library library server `LIBSRVRN` using the user ID `FRNADMIN` and the password `PASSWORD`, then disconnects from the library server.

```
import COM.ibm.mm.sdk.common.*;
import COM.ibm.mm.sdk.client.*;
import java.io.*;

public class TConnectDL implements DKConstant
{
   // Main method
   public static void main(String argv[])
   {
      try {
          DKDatastoreDL dsDL = new DKDatastoreDL();
           dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
           dsDL.disconnect();
      }
      catch (DKException exc) {
          System.out.println("Exception name " + exc.name());
          System.out.println("Exception message " + exc.getMessage());
          exc.printStackTrace();
      }
      catch (java.rmi.RemoteException exc) {
          System.out.println("Exception message " + exc.getMessage());
          exc.printStackTrace();
      }
   }
}
```

Setting up the environment

Since the client components in the Java API need to communicate with the server components to access and manipulate the data in Digital Library through the network, both the server and client side should be ready for the client/server execution. On the server side, the daemon should be ready to receive the request from the client using a specified port number. On the client side, the program needs to know the server name and port number. To communicate between client and server, the port number of the client and server should be same.

To start the daemon on the server side, the script file called `regist.cmd` on Windows NT and `regist` on AIX is provided. Before the daemon is started, define the correct port number and server name. The following is the description in the script file for Windows NT:

```
set remotePort=<port#>
start rmiregistry %remotePort%
java -Djava.rmi.server.codebase=http://<servername>/<~userDirectory>/<dlDirectory>/
                        COM.ibm.mm.sdk.remote.DKRemoteMainImp %remotePort% &
echo "Regist is over"
```

In the above script file, define <port#>, <servername>, <~userDirectory>, and <dlDirectory>. For example,

```
set remotePort=1919
start rmiregistry %remotePort%
java -Djava.rmi.server.codebase=http://ibmserver/~john/dl/
                       COM.ibm.mm.sdk.remote.DKRemoteMainImp %remotePort% &
echo "Regist is over"
```

After you update this script file, enter `regist` to execute the daemon on the server side.

For AIX, the script file looks like the following:

```
export remotePort=<port#>
rmiregistry $remotePort &
java -Djava.rmi.server.codebase=http://<servername>/<~userDirectory>/<dlDirectory>/
                       COM.ibm.mm.sdk.remote.DKRemoteMainImp $remotePort &
echo "Regist is over"
```

In the above script file, define <port#>, <servername>, <~userDirectory>, and <dlDirectory>. For example,

```
export remotePort=1919
rmiregistry $remotePort &
java -Djava.rmi.server.codebase=http://ibmsever/~john/dl/
                       COM.ibm.mm.sdk.remote.DKRemoteMainImp $remotePort &
echo "Regist is over"
```

After you update this script file, enter `regist` to execute the daemon on the server side.

To connect from client to server, two ways are provided to define the server name and port number on the client side. The first method is to define the `client.ini` file externally in the directory where the client program is running on the client side. The file is formatted as shown:

```
RemoteHost=<hostname>
RemotePort=<port#>
```

The hostname and port number can be defined to communicate with the server in the above example.

```
RemoteHost=ibmserver
RemotePort=1919
```

The second method is to set the server name and port number internally in the client program running on the client side. The statements to set up the server name and port number are:

```
DKRMIConstant.setRemoteHost("your-hostname");
DKRMIConstant.setRemotePort("your-port#");
```

The DKRMIConstant class is defined in the `COM.ibm.mm.sdk.common` package. So, the above example program, including these two statements in the client, is:

```
import COM.ibm.mm.sdk.common.*;
import COM.ibm.mm.sdk.client.*;
import java.io.*;

public class TConnectDL implements DKConstant
{
   // Main method
   public static void main(String argv[])
   {
      DKRMIConstant.setRemoteHost("ibmserver");
      DKRMIConstant.setRemotePort("1919");
      try {
            DKDatastoreDL dsDL = new DKDatastoreDL();
            dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
            dsDL.disconnect();
          }
    catch (DKException exc) {
           System.out.println("Exception name " + exc.name());
           System.out.println("Exception message " + exc.getMessage());
```

```
            exc.printStackTrace();
          }
      catch (java.rmi.RemoteException exc) {
            System.out.println("Exception message " + exc.getMessage());
            exc.printStackTrace();
          }
        }
}
```

Once you select the correct import statement, either client or server, the above program can be run on either the client side and the server side because the two DKRMI statement do not affect the server side.

Programming tips

The client package and server package should not be imported together in the same program, since these two packages provide exactly the same classes. The Java compiler will be confused if both packages are imported in your program. Therefore, you should import only one of these two packages. If your application is on the client side, import the client package. Otherwise, import the server package.

The client package of the Java API is useful for Web applications. Unlike the server package of Java API, the client one is created with pure Java programs. (The server package includes C-native programs.) Therefore, if your application is Web-oriented, you should use the client package, since the server package cannot totally move across the network.

Since the application on the client side needs to catch the additional exception java.rmi.RemoteException, always attach this exception in the application whether the application runs on the server side or on the client side.

Understanding the workflow and workbasket functions

This section describes the workflow and workbasket functions.

Understanding the workflow service

The Java APIs provide the workflow service for the Digital Library datastore, encompassing the workflow and workbasket functions that are available in the Folder Manager. A workbasket is a container that holds documents and folders that you want to process. A workflow is an ordered set of workbaskets that represent a specific business process. Folders and documents move between workbaskets within a workflow, allowing your applications to create simple business models and to route work through the process until completion.

The workflow model used in the Folder Manager follows these rules:

- A workbasket does not need to be located in a workflow
- A workbasket can be located in one or more workflows
- A workbasket can be located in the same workflow more than once
- A document or folder can only be stored in one workflow at a time; however, workbaskets can be located in multiple workflows
- A document or folder can be stored in a workbasket that is not located in a workflow

The DKWorkFlowServiceDL class represents the workflow service of the Digital Library datastore. This class provides the capability to start, change, remove, route, and complete a document or folder in a workflow. Additionally, the DKWorkFlowServiceDL class allows you to retrieve information about workbaskets and workflows in a Digital Library datastore. In Digital Library, the DKWorkFlowDL and DKWorkBasket classes are the object-oriented representations of a workflow item and a workbasket item, respectively.

Establishing a connection

You must establish a connection to a Digital Library datstore before you can use the workflow service. The Digital Library datastore provides connect and disconnect methods. The following connection example would connect to a Digital Library datastore named `LSMUFASA`, using the user ID `FRNADMIN` and the password `PASSWORD`. The complete sample of datastore connection, TListWorkFlowWFS.java, is available in the samples directory.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
...                                  // do some work
dsDL.disconnect();
```

The connect method allows an application to connect to a Digital Library datastore. After a workflow service is created, subsequent methods of the workflow service may be issued.

List workflows

The workflow service provides a method that lists the workflows in the system. The following example demonstrates the retrieval of the list of workflows; this list is returned in a sequential collection of DKWorkFlowDL objects. The complete sample of this function, TListWorkFlowsWFS.java, is available in the samples directory.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
DKSequentialCollection wfList = (DKSequentialCollection)wfDL.listWorkFlows();
if (wfList != null)
   {
      dkIterator pIter = wfList.createIterator();
      DKWorkFlowDL pwf1;
      while (pIter.more())
      {
         pwf1 = (DKWorkFlowDL)pIter.next();
         pwf1->retrieve();
         ...                         // do some work
      }
   }
dsDL.disconnect();
```

List workbaskets

The workflow service provides a method that lists the workbaskets in the system. The following example demonstrates the retrieval of the list of workbaskets; this list is returned in a sequential collection of DKWorkBasketDL objects. The complete sample of this function, TListWorkBasketsWFS.java, is available in the samples directory.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
DKSequentialCollection wbList = (DKSequentialCollection)wfDL.listWorkBaskets();
if (wbList != null)
   {
      dkIterator pIter = wbList.createIterator();
      DKWorkBasketDL pwb1;
      while (pIter1.more())
      {
         pwb1 = (DKWorkFlowDL)pIter1.next();
         pwb1->retrieve();
         ...                        // do some work
      }
   }
dsDL.disconnect();
```

List items in a workflow

The workflow service provides a method that lists the item IDs of the items in a workflow. The following example demonstrates the retrieval of the list of item IDs for the items in a particular workflow; this list is returned in a sequential collection of DKString objects. The complete sample of this function, TListItemsWFS.java, is available in the samples directory.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
DKSequentialCollection wfList = (DKSequentialCollection)wfDL.listWorkFlows();
if (wfList != null)
   {
      dkIterator pIter = wfList.createIterator();
      while (pIter.more())
      {
       DKWorkFlowDL pwf1 = (DKWorkFlowDL)pIter.next();
       System.out.println("Workflow ID = " + pwf1.getID());
       DKSequentialCollection itemList = (DKSequentialCollection)pwf1.listItemIDs();
       if (itemList != null)
       {
         dkIterator iter1 = itemList.createIterator();
         String itemid;
         System.out.println("There are " + itemList.cardinality()
                            + " folders/documents defined in the workflow");
         while (iter1.more())
         {
           itemid = (String)iter1.next();
           System.out.println(" ------- Item ID = " + itemid);
         }
       }
      }
   }
dsDL.disconnect();
```

Executing a workflow

The workflow service provides methods that allow you to execute a workflow. The following example demonstrates how to start an item in a workflow, how to route an item to a workbasket, and how to complete an item in a workflow. The complete sample of this function, TProcessWFS.java, is available in the samples directory. You must modify the sample program to reflect these changes:

- Use a valid item ID instead of `EP8L8OR9MHH##QES`
- Use a valid workflow ID instead of `HI7MOPALUPFQ1U47`

- Use a valid workbasket ID instead of E3PP1UZOZUFQ1U3M

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
DKString itemID   = new String("EP8L8OR9MHH##QES");
DKString itemIDWF = new String("HI7MOPALUPFQ1U47");
DKString itemIDWB = new String("E3PP1UZOZUFQ1U3M");
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
wfDL.startWorkFlowItem(itemID,                  // itemID
                       itemIDWF,                // itemIDWB
                       NULL,                    // default (the first workbasket)
                       TRUE,                    // overload
                       DK_WIP_DEFAULT_PRIORITY  // initial_priority
                       );
...                                          // do some work
wfDL.routeWipItem(itemID,                    // itemID
                  itemIDWF,                  // itemIDWB
                  TRUE,                      // overload
                  DK_NO_PRIORITY_CHANGE      // initial_priority
                  );
...                                          // do some work
wfDL.completeWorkFlowItem(itemID);
dsDL.disconnect();
```

Creating a workflow

The workflow service allows you to create a new workflow. The typical workflow creation process follows these steps:

1. Create an instance of DKWorkFlowDL
2. Set the workflow name to GOLF
3. Set the workbasket sequence to NULL to indicate that this workflow contains no workbaskets
4. Set the privilege to **All Privileges**
5. Set the disposition to DK_WF_SAVE_HISTORY
6. Call the add method

The following example uses these steps to create a workflow. The complete sample of this function, TCreateDelWorkFlow.java, is available in the samples directory. If you connect to the datastore as a normal user (DK_SS_NORMAL), you will not get the workflow that is defined after you connect. Therefore, this sample uses DK_SS_CONFIG.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
Object input_option = new Integer(DK_SS_CONFIG);
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
dsDL.setOption(DK_OPT_DL_ACCESS, input_option);
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
DKWorkFlowDL newwf = new DKWorkFlowDL(wfDL);
newwf.setName("GOLF");
newwf.setWorkBasketSequence((dkCollection *)NULL);
newwf.setAccessList("All Privileges");
newwf.setHistoryDisposition(DK_WF_SAVE_HISTORY);
newwf.add();
...                                    // do some work
dsDL.disconnect();
```

Creating a workbasket

The workflow service allows you to create a new workbasket. The typical workbasket creation process follows these steps:

1. Create an instance of DKWorkBasketDL

2. Set the workbasket name to `Hot Items`
3. Set the privilege to **All Privileges**
4. Call the add method

The following example uses these steps to create a workflow. The complete sample of this function, TCreateDelWorkBasket.java, is available in the samples directory. If you connect to the datastore as a normal user (DK_SS_NORMAL), you will not get the workflow that is defined after you connect. Therefore, this sample uses DK_SS_CONFIG.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
Object input_option = new Integer(DK_SS_CONFIG);
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
dsDL.setOption(DK_OPT_DL_ACCESS, input_option);
dsDL.connect("LSMUFASA", "FRNADMIN", "PASSWORD");
DKWorkBasketDL newwb = new DKWorkBasketDL(wfDL);
newwb.setName("Hot Items");
newwb.setAccessList("All Privileges");
newwb.add();
...                                    // do some work
dsDL.disconnect();
```

Chapter 5. Using the Sample Java Applets and Servlet

The Java Applets are built for Web administrators to quickly create a Web site to connect, query, and access Digital Library. The Java Applets, also known as the Internet Application Toolkit, are built using the Digital Library Java API, Java Development Toolkit 1.1 using Remote Method Invocation (RMI) for remote access, and the new Delegation Event Model.

The Internet Application Toolkit consists of three applets and one servlet: Connect, Query and View applets, and the Retrieve servlet. All three applets work in conjunction to give easy access to Digital Library. The Retrieve servlet provides flexibility on the data types that can be returned and displayed through a web browser.

The Internet Application Toolkit has three variations to demonstrate the flexibility of the Java API and to provide ease of use.

DTApp

Use this application to browse the data in accessible Digital Library servers.

To run:

On AIX, enter `. DTApp.ksh`

On Windows NT or Windows 95, click the **Start** button on the Windows task bar. Select **Programs → Digital Library → Internet Application Toolikt**

dtappsvr.html

Use this sample file as an example of how the Internet Application Toolkit can be run locally through the Web browser.

To run, access dtappsvr.html through a Web browser that fully supports JDK 1.1, such as the HotJava Browser 1.0 FCS. The client toolkit must be installed on the machine on which dtappsvr.html is being invoked.

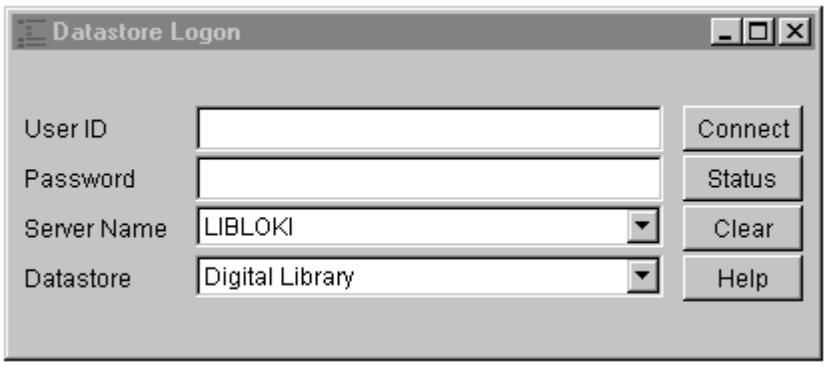
dtappclt.html

Use this sample file as an example of how to access the Internet Application Toolkit from a machine that does not have any Digital Library component installed.

To run dtappclt.html, you must set up a Web site and other configurations that are explained in detail in the following sections.

Connect applet

The Connect applet provides a simple interface to connect to different servers and check on the status of the connection. It has fields for the user ID and password, and drop-down lists for the accessible servers and types of datastore for connecting to the desired Digital Library server.



The Connect applet has the following parameters:

Country
Country language and formats desired to be used, supersedes LanguageCode and RegionCode. For example, Japan.

LanguageCode
Language code desired, must have the corresponding region code. For example, ja.

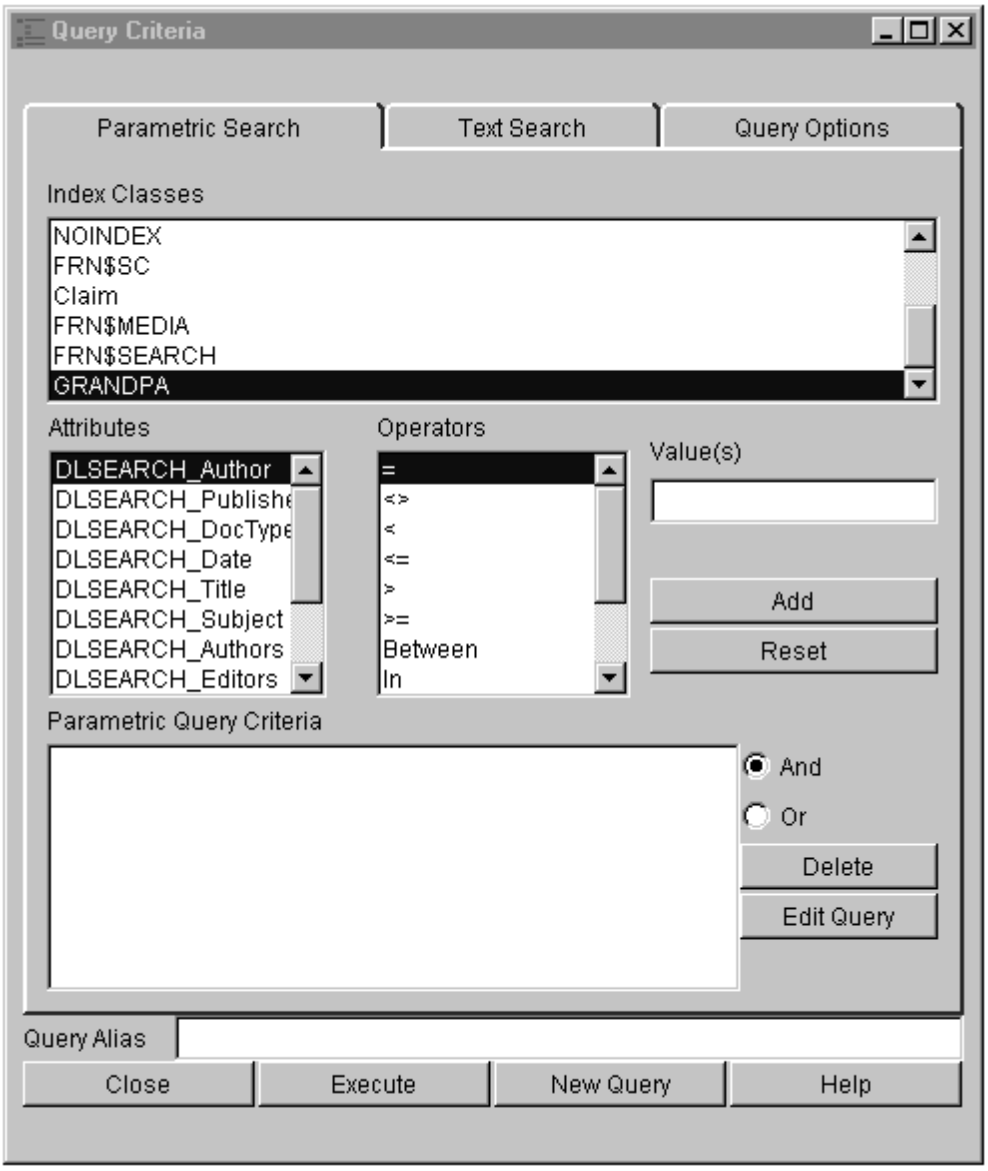
RegionCode
Region code desired, must have the corresponding language code. For example, JP.

RemoteHost
Host name for the RMI server, which is used only for remote client access so that the remote client knows which RMI server to contact.

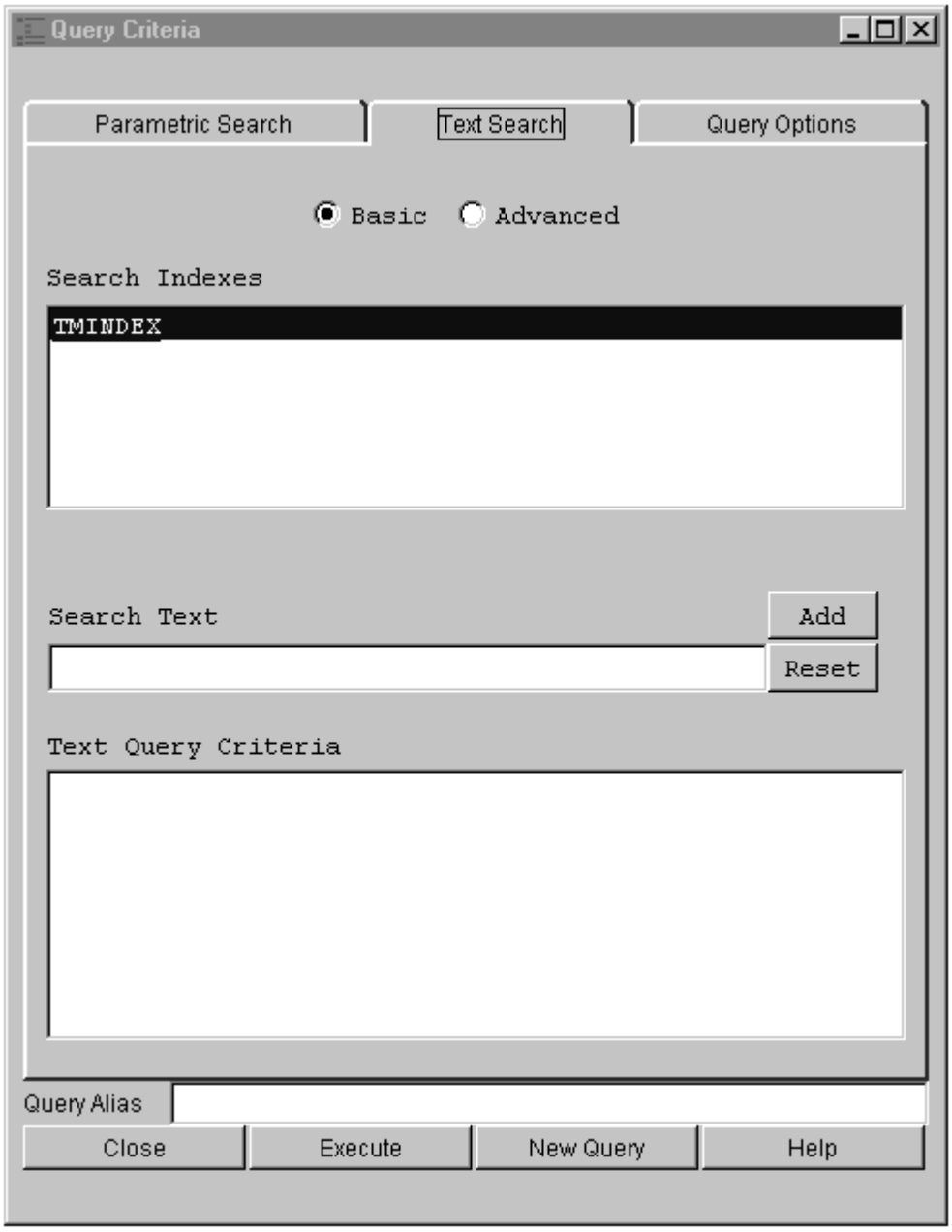
RemotePort
Port number for the RMI server, which is used only for remote client access so that the remote client knows which RMI port to use.

Query applet

The Query applet lists the index classes and their attributes of the current connected Digital Library and provides a GUI to guide you to visually construct your query criteria. You can search for attribute values or text strings, or combine them to enrich the search capability. While executing the query, the result list is displayed in the View applet simultaneously. The three tab pages give you easy choices for constructing the query.



The Parametric Search page displays the list of available index classes and the attributes associated with each index class. By clicking on the attributes and operators and entering the value to search for, a parametric search string is constructed and shown in the lower section of the page. You can also change the query by either selecting the current criteria and resetting them, or by clicking the **Edit Query** push button and modifying the parametric search string directly. Modifications through the **Edit Query** push button will disassociate the search string from further use by the graphical user interface until a new query is done.



The Text Search page displays the list of text search index classes and provides fields in which to enter the search string. Your can enter either a simple Boolean search, or click on the advanced functions and create a more complicated text search. Modifying the constructed query can also be done through the **Edit Query** button.

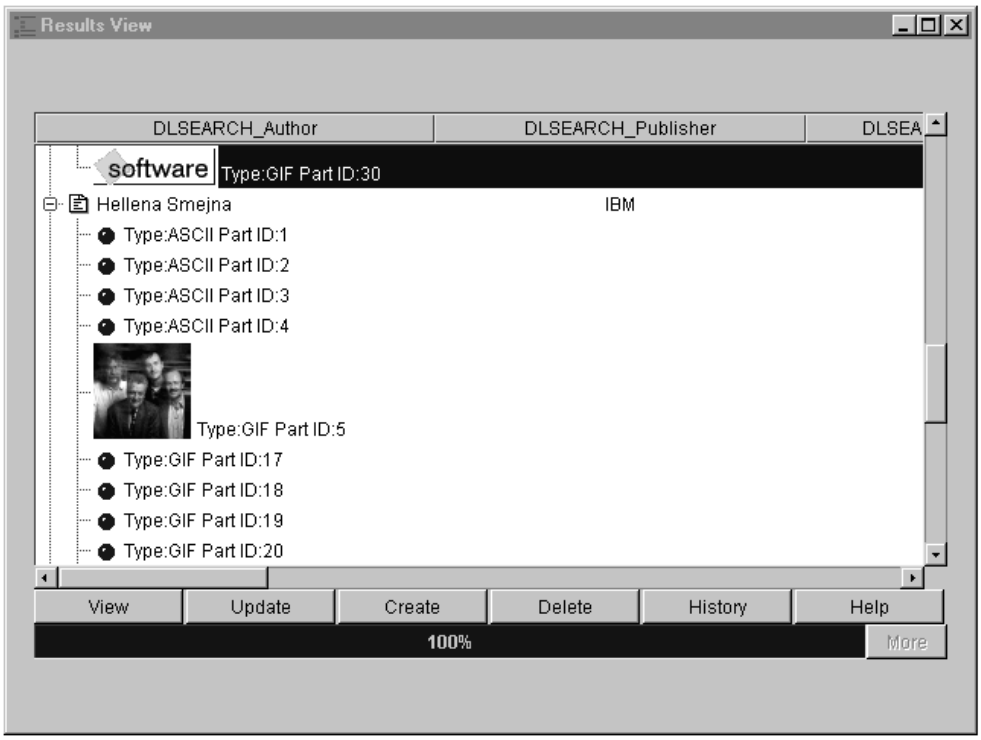
Combining the Parametric Search page and the Text Search Engine page allows you to do a combined search. If there is a search string entered in both the Parametric Search page and the Text Search Engine page, the application does a combined search and displays the intersection of the two search results.

The last page, Query Options, allows you to set generic query options for all the search strings.

The Query applet does not have any parameters.

View applet

The View applet displays the query result list in a tree structure. Click the left mouse button to expand or collapse an item from the result list. You can also create, retrieve, update, and delete data objects if you have the privilege to do so. Select an item from the result list and click the right mouse button to display the popup menu that shows the actions that you can perform on that data object.



You can view the individual parts of an item separately with the View applet, which displays GIF and JPEG files, or in conjunction with the Retrieve servlet, which displays the part through a Web browser. To view the item in its entirety, select a Doc item and click the right mouse button to display the popup menu. Click the menu item for the Dynamic Page Builder site that has been set up with the corresponding Digital Library connections and data. All the parts of the item are brought together and displayed as one part through the Web.

The View applet has the following parameters:

cgisource
: The URL source that contains the Retrieve servlet. For example, http://gandalf.stl.ibm.com:8080/servlet/DTRetrieve?

indexClassList
: The list of index classes, their attributes, and the thumbnail part. For example, (`GRANDPA`)(`1 2 3`)(`1`) means show index class `GRANDPA`'s first three attributes and the first image part is the thumbnail (0 means don't show thumbnail).

advance
The flag for showing the advanced features—create, update, and delete. The default value is true.

Retrieve servlet

The Retrieve servlet is used by the View applet to retrieve and display various types of parts from Digital Library. The servlet is located in the frnroot\lib\servlet directory, which contains four files:

DTRetrieve.java
The source code for the servlet.

You can modify the source as needed and use it to better understand how the servlet works. As new mime types are created or needed, you can modify DTRetrieve to handle the new types.

DTRetrieve.class
The executable for the servlet.

Servlets.properties
The file where you put initialization parameter values.

DTRetrieve has three initialization parameters:

datastore
The name of the library server to connect to.

Restriction: Currently, the Digital Library client for AIX is single-threaded and the servlet servers operate by spawning new threads for each request. Two or more requests arriving at the Retrieve servlet at the same time cause exceptions and the servlet server might stop.

userid The user ID to connect with.

password
The password to connect with.

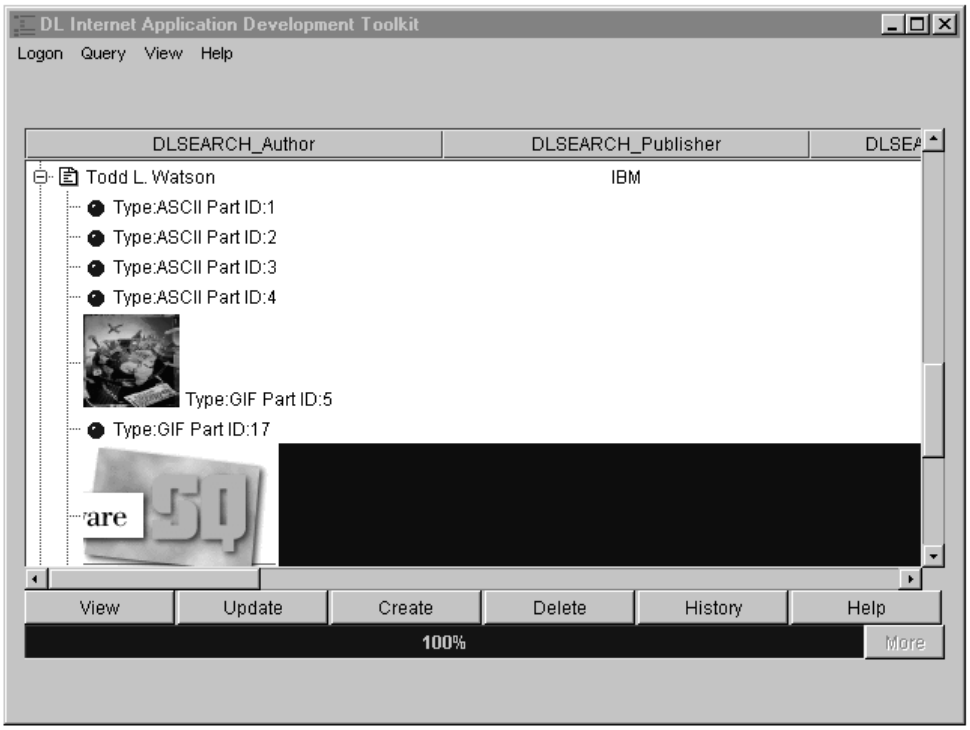
Classes.zip
The servlet class library files.

For the servlet to run, you must either have a Web server that is capable of running servlets, or run `srun.exe`, which is part of the Java Servlet Development Package available from JavaSoft at http://www.javasoft.com. If your Web server supports servlets, follow the documentation for that Web server on how to set up the servlet, for example, where to place the servlet. If srun.exe is to be used, enter `srun.exe -s servlets.properties` in the servlet directory. For more information on srun.exe, enter `srun -?` to see the options available.

For more details on servlets, go to http://www.javasoft.com and search for the Java Servlet Development Kit (JSDK).

Java application on a Digital Library client

DTApp.bat is the file that executes the applets as a Java application. Enter `dtapp` at a command prompt to start DTApp and retrieve the list of library servers that it can connect to.



You can set two view options under the view menu:

Set Web Browser
Allows you to enter the Web browser to be used when viewing parts that are not GIF or JPEG files. A sample of what might be entered in this option is `c:\netscape\program\netscape.exe`.

Set Web Server
Allows you to enter the Web server address to be used. For example, `http://webserver/servlet/DTRetrieve?`.

When retrieving non-GIF/JPEG files, ensure that both of the above options are set correctly and that srun is running correctly for the servlet. See “Retrieve servlet” on page 142 for details on using srun.exe and the Retrieve servlet.

Local applet access on a Digital Library client

A sample HTML page, dtappsvr.html, incorporates three applets—DTConnectApplet, DTQueryApplet, and DTTreeViewApplet—and allows logon, query, and view of data within Digital Library. To access this HTML page:

1. Go to the frnroot\lib\servlet directory and modify the servlets.properties file to have the correct datastore name, user ID, and password to connect to the correct library server.

   If your Web server is not servlet-enabled, you can download the Java Servlet Development Kit from http://www.javasoft.com and use `srun.exe` to test the servlets. The solaris package works for AIX also. To run srun, enter `srun -s servlets.properties` from the directory frnroot\lib\servlet. This starts a servlet server with the initial arguments set in servlets.properties. If your Web server is

servlet-enabled, refer to the Web server documentation or the Web server administrator for information on how to set up servlets for this particular Web server.

If no default initialization arguments are passed in by servlets.properties, the current DTRetrieve.class is compiled with server = LIBSRVR2, userid = FRNADMIN, and password = password. See “Retrieve servlet” on page 142 for more information on srun.exe and the Retrieve servlet.

2. Go to a browser and invoke the HTML page by entering the address, `file:/`*`frnroot`*`/dtappsvr.html`, where *frnroot* is the full path for dtappsvr.html. For example, c:\frnroot. This starts the three applets.

   The connection window opens when the applets have been properly loaded. Connect with the correct user ID/password/server combination, make a query against the index class, and view the items sent back.

   Restriction: You must use a Web browser that is enabled for JDK 1.1. Currently, the HotJava Browser 1.0 FCS is the only fully JDK 1.1 enabled Web browser. Within the HotJava Browser, make sure that the security level for unsigned applets is medium.

Remote applet access

A sample HTML page, dtappclt.html, incorporates three applets—DTConnectApplet, DTQueryApplet and DTTreeViewApplet—and allows you to remotely logon, query, and view your data. To access this HTML page:

1. Make sure that FRNROOT is set to an HTML-accessible directory for the Web server residing on your machine. If not, consult either your Web administrator or your Web server documentation to have this set up.
2. Go to the frnroot\lib\servlet directory and modify the servlets.properties file to set the correct datastore name, user ID, and password to connect to the correct library server.

   If your Web server is not servlet-enabled, you can download the Java Servlet Development Kit from http://www.javasoft.com and use `srun.exe` to test the servlets. The solaris package works for AIX also. To run srun, enter `srun -s servlets.properties` from the directory frnroot\lib\servlet. This starts a servlet server with the initial arguments set in servlets.properties. If your Web server is servlet-enabled, refer to the Web server documentation or the Web server administrator for information on how to set up servlets for this particular Web server.

   If no default initialization arguments are passed in by servlets.properties, the current DTRetrieve.class is compiled with server = LIBSRVR2, userid = FRNADMIN, and password = password.
3. Edit dtappclt.html to remove the comment tags from RemoteHost and RemotePort, after the definition of DTConnectApplect.class. Set RemoteHost to the host name of the Web server. Set RemotePort to the port number that the RMI server is using.

   Tip: Alternatively, CLIENT.INI, which is located in the FRNROOT directory, can be modified with the Web server’s host name where it states HOSTNAME and a port number where it states PORTNUMBER. The port number should be some unique port number, which will be used for RMI (Remote Method Invocation) communication. The client.ini file should be placed in the directory of the client browser’s invocation directory.

4. Modify regist.cmd on Windows 95/NT and regist on AIX, which is located in frnroot\lib directory, to use the same port number as the RemotePort parameter in dtappclt.html, or the same PortNumber in client.ini, if client.ini is used.
5. Go to the frnroot\lib directory and enter `regist` to initialize the RMI server.
6. Go to a browser and invoke the Web page dtappctl.html from your Web server's alias, http://webserver/frnroot/dtappctl.html. This brings up the Web page and starts the three applets.

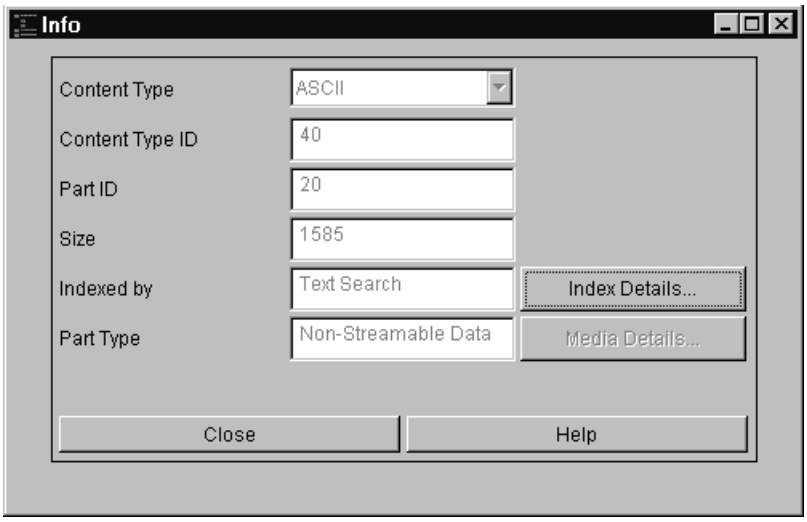
   The connection window opens when the applets have been properly loaded and have started up. Connect with the correct userid/password/server combination, make a query against the index class, and view the items sent back.

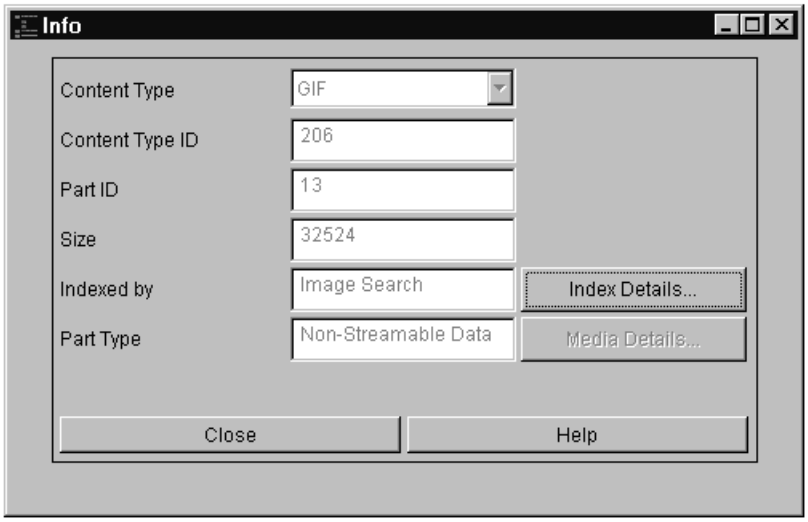
   Restrictions:

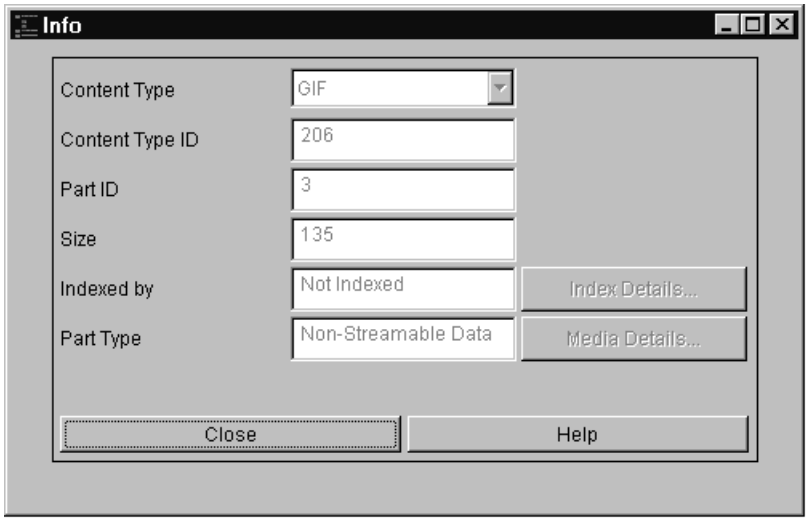
   - You must use a Web browser that is enabled for JDK 1.1. Currently, the HotJava Browser 1.0 FCS is the only fully JDK 1.1 enabled Web browser. Within the HotJava Browser, make sure that the security level for unsigned applets is medium.
   - The Digital Library client for AIX is currently single-threaded, so any one process on AIX can have only one connection at a time. This means that there can be only one remote connection at a time through one RMI process.

Display part information

The info function displays the information of the selected part. A Digital Library part must have a part ID and can also have a pre-defined type to describe its content. The following screen captures show the parts indexed, respectively, by a text search engine, an image search engine, and nothing. To view the index details of an indexed part, click the **Index Details** push button in this window.

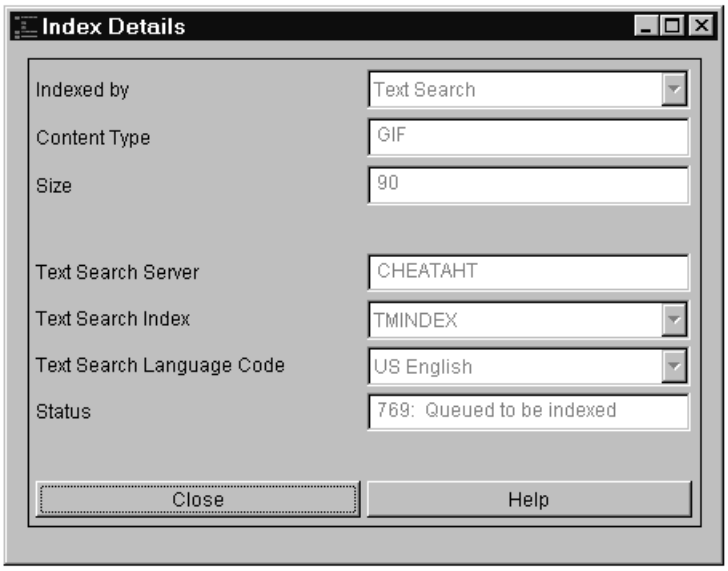


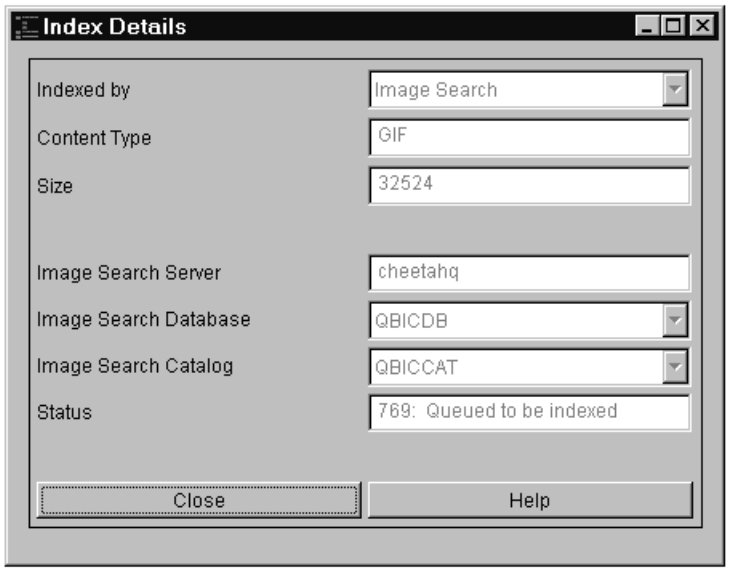




The Index Details window shows the search engine type, server, and index information of the selected part. The information presented in the screen captures below is example text for the parts indexed by either a Text Search Engine server

or a QBIC server.

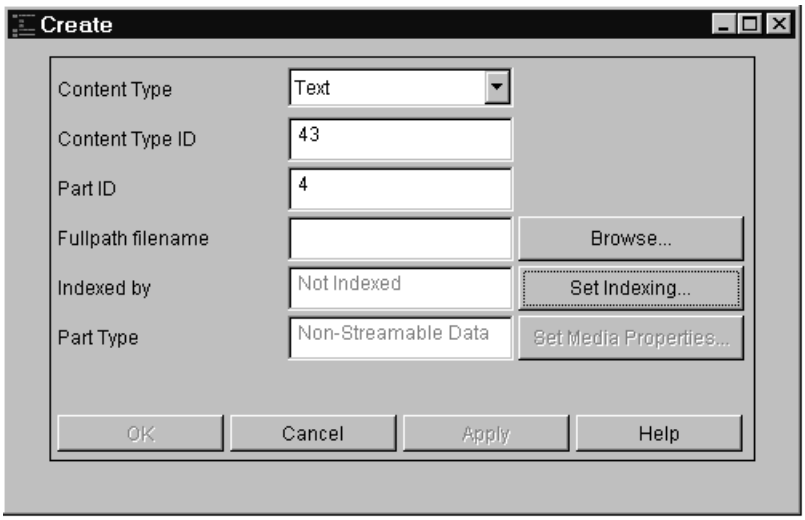


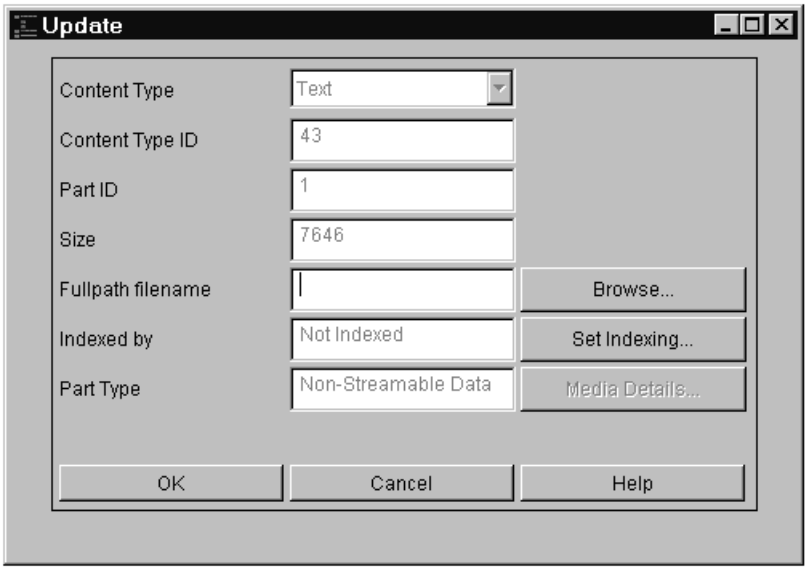


Index a part

This function allows you to index a part while loading, updating, or indexing an existing part. If you set the indexing information when you create a part, the creation will include both create and index processes. When you update a part (replacing a part content with a file, for example), if this part has not been previously indexed, you can set the indexing information and the update function will invoke the index process. If any search engine has already indexed the part to be updated, the update function will automatically invoke the index process using the existing indexing information. To index a part while the part is being loaded or updated, press the **Set Indexing** push button located in the Create and Update

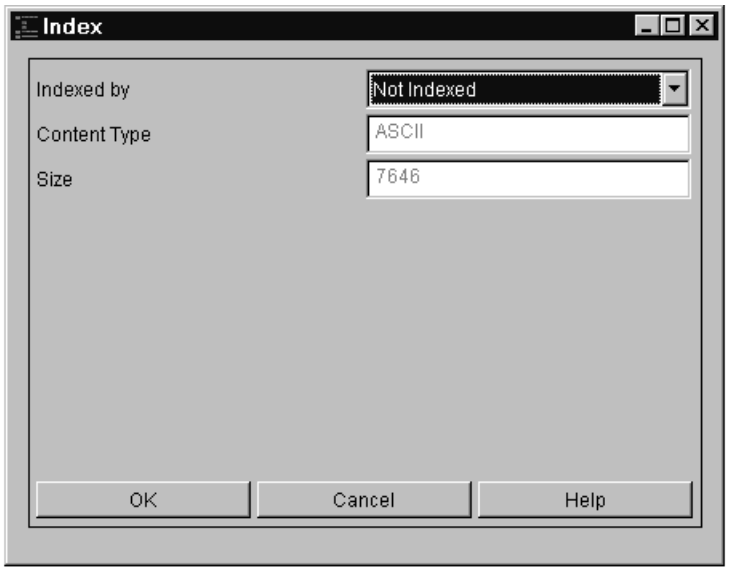
window.



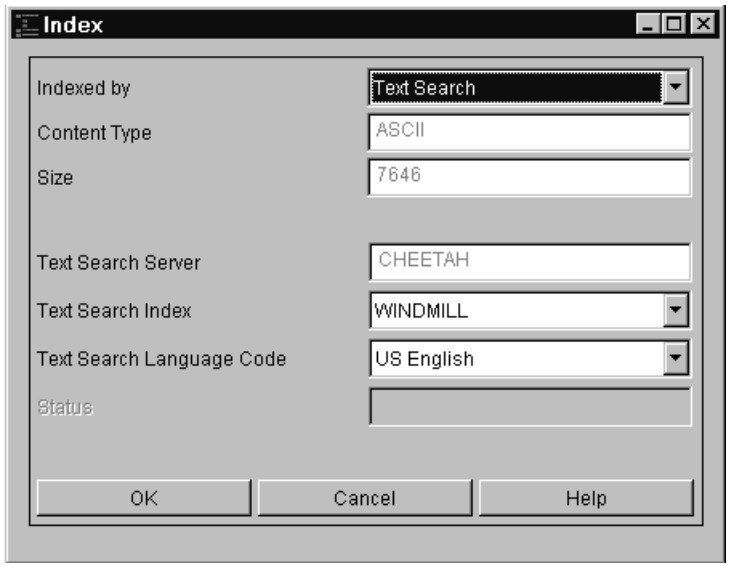


To index an existing part, select a part from the query result tree and then select **Index** from the pop-up memu. The Index window, shown below, requires input data

for indexing.

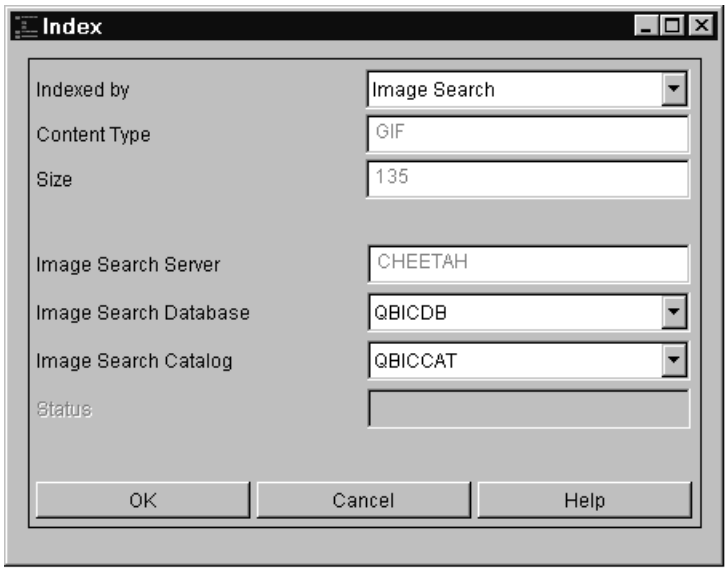


Selecting the search engine type from the **Indexed by** menu will update this window with the necessary input information. The windows are listed below (you must select the search server catalog information).

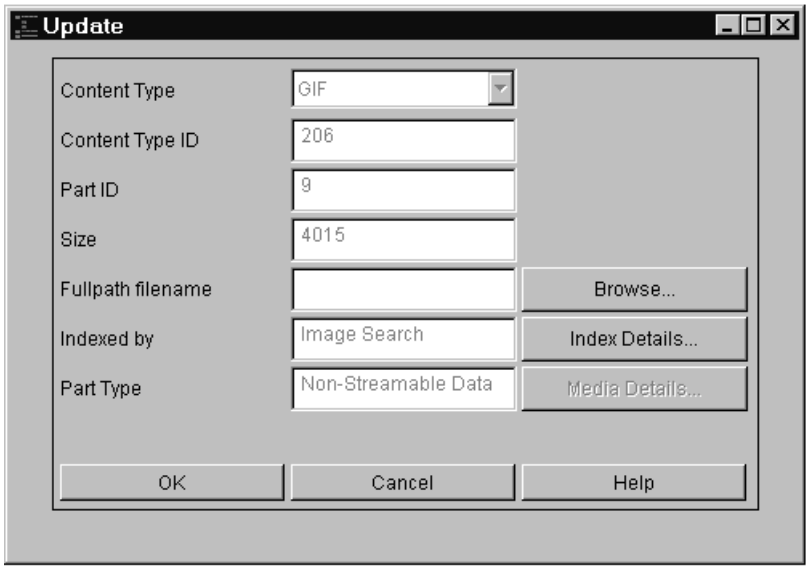


If the search server is not connected when you select a search type, the logon window will automatically pop up. At this point, you must select a server to connect to. You need to know which search server is associated with the currently connected Digital Library server. If the parts have already been indexed by any search engine, those parts cannot be indexed again. In this situation, selecting the **Index** menu choice will open the Index Details window. This rule applies to the

update fuction as well.



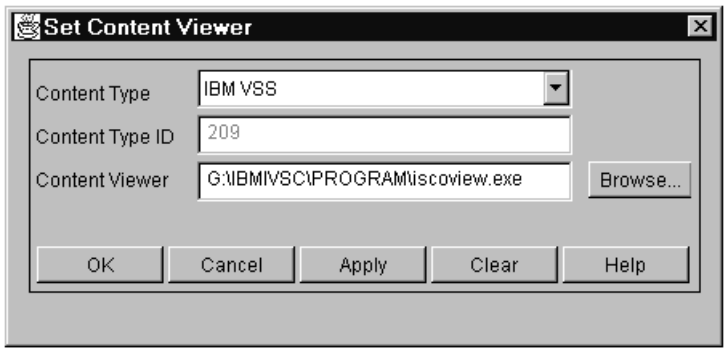
When you update an indexed part, the update function will automatically invoke the indexing process using existing indexing information. The Update window will look slightly different from the window depicted earlier; the new look is shown below.



Set content viewer

The Set Content Viewer window allows you to specify a viewer for displaying the content of a particular type of object. Each type can have only one viewer. The settings will be saved into a file; therefore, the settings will always be available throughout different sessions. After you have set a viewer, you can invoke the viewer from the query result tree by selecting a part and clicking the **View** push

button.



Content Type
The choices for the content type. Once a type is selected, its ID will be displayed in the **Content Type ID** field.

Content Viewer
The viewer application name with full path. Click the **Browse** push button to navigate the local file system. A file selection box will pop up if you click on this button.

OK Save the displayed settings and close the window

Apply Save the displayed settings

Clear Clear the displayed settings

Cancel
Close this window without committing any changes that have not been changed using the **Apply** or **Clear** push buttons

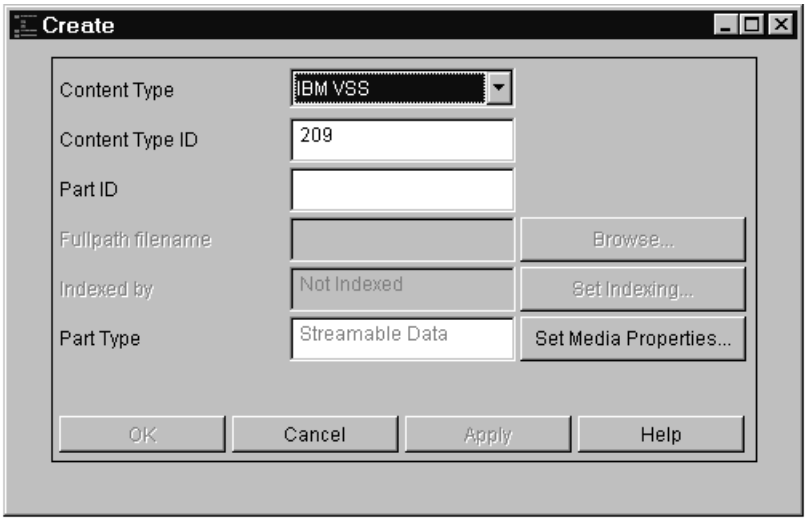
The settings are saved into the file named `dtcviewer.set`. For AIX, the default directory of this file is under $HOME; for Windows NT, the default directory is under %FRNROOT%.

Load video streams

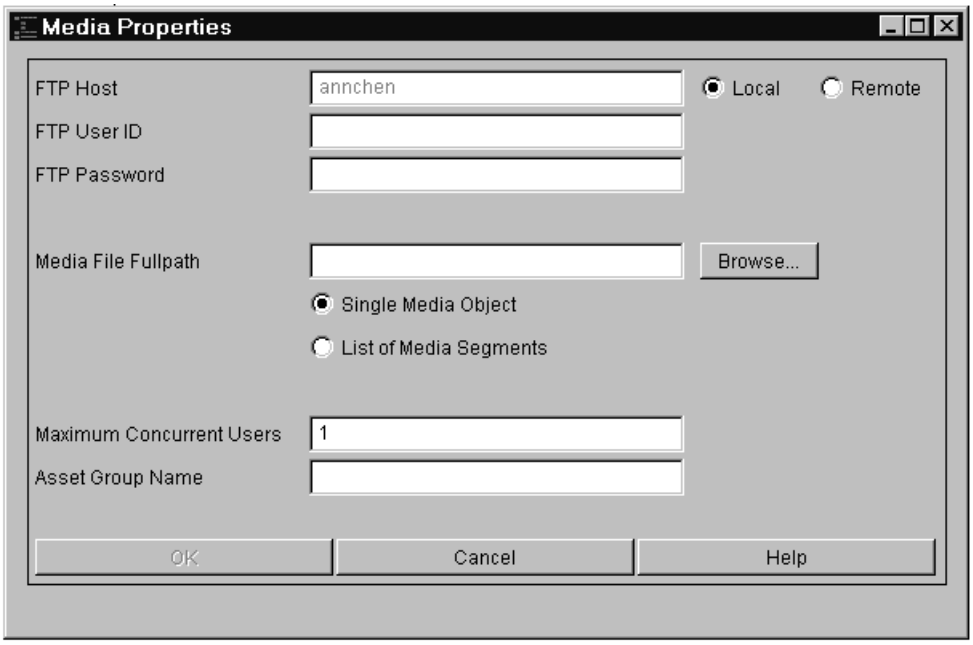
This function loads video streams into DL. Each video stream is a digitized file and must reside on a machine that is running an FTP server. The FTP server is necessary to transfer the video file to the VideoCharger server.

To load a video stream, first select **Create Part** from the query result tree menu. A Create window will pop up as shown in the screen capture below. From that window, you must select the **IBM VSS** content type for loading video stream parts. This selection will enable the **Set Media Properties** push button and will require

you to enter video asset properties.



The following screen capture shows the window that pops up when you press the **Set Media Properties** push button.



The required inputs are:

FTP Host
: The machine host name that has the video assets and an FTP server running

Local/Remote radio group
: Specifies whether the FTP host is local to the machine that this application runs from. Selecting **Remote** will disable the **Browse** push button.

FTP User ID/Password
: The user ID and password to log onto the host machine

Media File Fullpath
The video file full path and name. If this file is local, the user can click the **Browse** push button to navigate the local file system to find the video file.

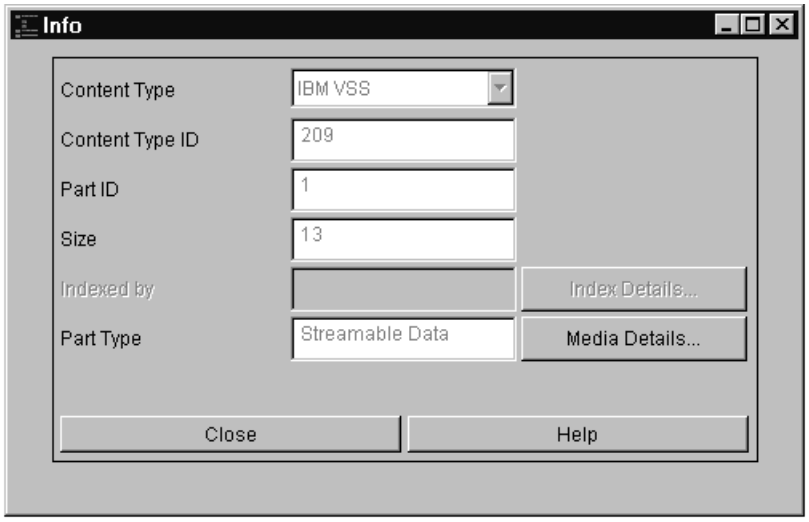
Single Media Object/List of Media Segments radio group
Indicates whether the file displayed in the Media File Fullpath field is the actual video stream file, or a control file which contains a list of fullpaths containing a set of video segments

Maximum Concurrent Users
Specifies maximun concurrent users for this video asset. This value is not enforced by the current version of VideoCharger (V2.0). Any positive, non-zero number will be accepted.

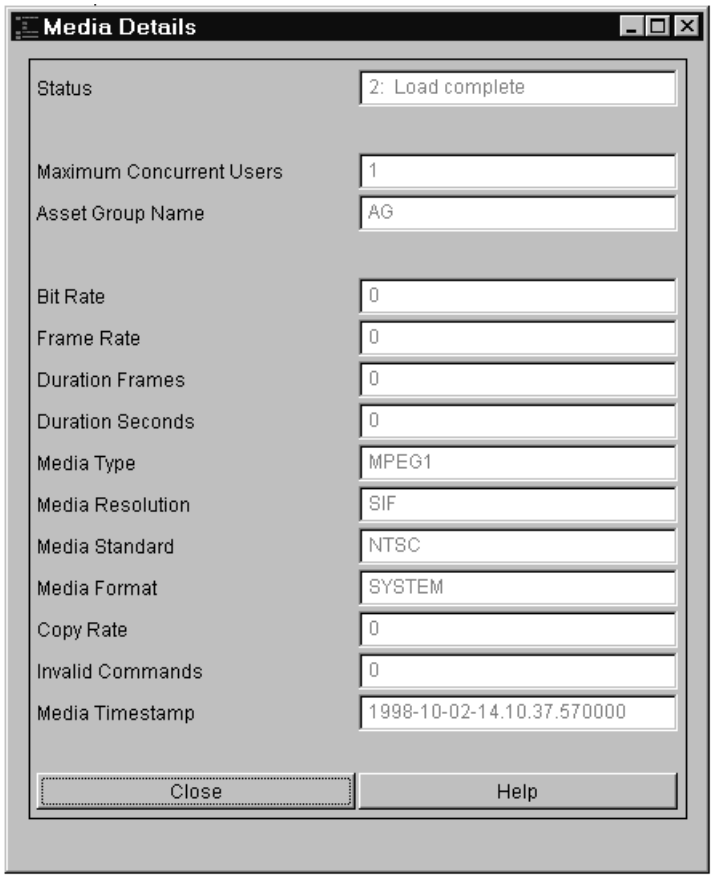
Asset Group Name
The storage entity name as defined in VideoCharger. Currently, VideoCharger has only one asset group. During VideoCharger installation, this default asset group name is `AG`. However, the installer may change this name; you must know the actual asset group name in order to load video parts.

Display video stream parts information

This function allows you to retrieve video parts information and loading status. From the query result tree, select a video part and **Info** pop-up menu item. The generic part information window displays the following information:



From the above window, click **Media Details** to display video stream part information:



The **Status** field displays the video loading states, including:

1. Load in process
2. Load complete
3. Load failed

This value will be updated automatically during the loading process.

Play video streams

To play video streams, you must have the VideoCharger Player installed. The VideoCharger Player is currerntly available for Windows NT only. To play a DL video stream, set the content viewer for video stream parts, then double-click on a video part from the query result tree.

Working in conjunction with Dynamic Page Builder

If Dynamic Page Builder is set up and configured, the applets also have a feature to view data that has been set up to be displayed by Dynamic Page Builder and Net.Data. In the View applet, when the list of items returned are also viewable from a Dynamic Page Builder setup site, click the right mouse button after selecting one of the document items. A popup menu appears with Dynamic Page as one of the menu items. Select that menu item to display a window that prompts you for the

part in this item that describes how the document should be brought together and for the URL of the Dynamic Page Builder setup. Make sure that frnpage.d2w for Dynamic Page Builder is set up correctly. See “Configuring the Dynamic Page Builder with Net.Data” on page 201 for more information on how to configure Dynamic Page Builder.

Chapter 6. Using the ActiveX application programming interface

The ActiveX application programming interface (API) is a set of ActiveX classes which provides access and manipulation of local or remote data stored in the Digital Library storage facilities. This section describes the design of the ActiveX application programming interface, the ActiveX implementation of Digital Library API multi-search facilities. It allows solution providers using any macro language or programming tool that supports Automation (for example, VBA, Lotus script, etc.) to create custom applications for doing multi-search and updates across multiple datastores. The objectives are to support:

- Multi-search and update across a combination of heterogeneous datastores
- A common-object model for Digital Library data access
- A flexible mechanism to use a combination of different search-engines; for example, Text Search Engine text search and QBIC image search

This ActiveX API specification follows the same structure as the C++ API specification wherever possible, to offer consistency and commonality between the ActiveX and C++ APIs. As ActiveX has its unique traits, it is necessary to define some parts of the ActiveX API differently than the C++ API. The ActiveX classes are basically automation servers that support the COM IDispatch interface (dual interfaces may be provided at a later release). A type library is shipped with the .DLL so that a programming tool can determine which interfaces an object supports, as well as the names of its members. Since automation servers are not meant to be subclassed, the ActiveX classes represent the leaf classes in the C++ class hierarchy and they include all the methods of their C++ counterparts, plus those of their parents. For methods that return non-leaf class objects, the return type is either replaced by the concrete equivalent or by the type Object. Visual Basic users can use the built-in TypeOf function to determine the real type. The differences between the two APIs will be described in subsequent sections, which use the same samples as those in the C++ sections but translated into Visual Basic. In general, the differences can be attributed to the following:

- Operators are either replaced by functions, or in cases where the functionality can be replaced by one or more existing functions, eliminated entirely.
- Since the datatypes have to be automation compatible, methods that take a DKAny object as argument now take a VARIANT instead. This will allow the methods to still perform run time type checking to ensure the data passed in is the correct type.

Multi-search facilities

The multi-search facilities can:

- Search within a given datastore, using one or a combination of supported query types:
  - Parametric query:
    - queries requiring an exact match on the condition specified in the query predicate and the data values stored in the datastore
  - Text query:
    - queries on the content of text fields for approximate match with the given text search expression; for example, the existence (or nonexistence) of certain phrases or word-stems.

- Each search type is supported by one or more search-engines
- Search on the results of previous search

Installation

The ActiveX API is installed as part of the Digital Library OO toolkit. For the first release, an additional step has to be run to register the classes before the automation classes can be used. To register the classes with the windows registry, open up a command prompt and go to the directory where Digital Library is installed (for example, D:\FRNROOT). Locate the file named DXInstall.exe and run it by entering `DXInstall`; you should get a message indicating the success or failure of the process. In the unlikely event that a problem occurs, go into the log directory and locate the DXInstall.log file; in it you will find information regarding which class(es) has(have) failed to register. At this point, you will have to manually edit the entries in the registry. A file named DXMM.reg, located in the FRNROOT directory, contains information on all the changes required to be made to the registry; this will help you in determining what entries need to be modified. An additional step is also necessary to unregister the classes during the uninstall process. Locate the file named DXUninstall.exe and follow the same procedure as install, and check the DXUninstall.log for potential problems. Again, DXMM.reg should help you in determining what entries need to be modified, should it be necessary.

In addition to the code segments described in this document, three sets of samples are provided on the Digital Library CD, one for each type of query. These are Visual Basic Forms, created with Visual Basic Version 5, and LotusScript source files. They incorporate relatively simple user interfaces and are intended as general guidelines only. You are encouraged to explore and use some of the more sophisticated features of Visual Basic, such as the tree and image list controls, or Lotus Approach's Forms and Picture Plus fields for displaying data.

Make sure that the Digital Library .DLL is loaded into memory when you run your developing application within the Visual Basic environment. If this .DLL is loaded more than once within Visual Basic, an exception may occur. If your application is a free-standing executable run independently from Visual Basic, or if you exit from Visual Basic (after terminating your application) and restart, no exception will occur.

See the examples in the samples directory, for information on how to work around this current situation.

Understanding Digital Library datastore DL

A DXDatastoreDL represents and manages a connection to a Digital Library datastore; it also provides transaction support and executes datastore commands.

Digital Library datastore connect and disconnect

The Digital Library datastore provides a method for connect as well as a method for disconnect. Normally, you would create a datastore DL, connect to it, do some work, then disconnect when done. The following example illustrates connecting to the Digital Library server `LIBSRVRN` using the user ID `USER1` and the password `PASSWORD`:

```
Dim dsDL as new DXDatastoreDL
dsDL.connect "LIBSRVRN", "USER1", "PASSWORD"
dsDL.disconnect
```

Set and get Digital Library datastore options

The datastore provides some processing or informational options that you can set or get using its methods.

The following example shows how to get and set the option for starting an administrative session. Refer to the reference documentation for the list of options.

```
Dim session_type as Long
Dim dsDL as new DXDatastoreDL
dsDL.setOption dx_OPT_DL_ACCESS, dx_SS_CONFIG
dsDL.connect "LIBSRVRN", "USER1", "PASSWORD"
dsDL.getOption dx_OPT_DL_ACCESS, session_type
If session_type = dx_SS_CONFIG Then
  MsgBox "Datastore access is an administrative session"
End If
dsDL.disconnect
```

Digital Library datastore list servers

The datastore provides a method to list the library servers it can connect to. The following example shows you how to retrieve the servers. The steps are:

1. Create a DXDatastoreDL
2. List the Digital Library servers
3. Create an iterator for the collection and loop through the DXServerInfoDL objects to get the server name and type

```
Dim dsDL as new DXDatastoreDL
Dim col as DXSequentialCollection
Dim iter as DXSequentialIterator
Dim serverInfo as DXServerInfoDL
Dim strServerName as String, strServerType as String
Dim i as Long
i = 0
Set col = dsDL.listServers
Set iter = col.createIterator
Do While iter.more
  i = i + 1
  Set serverInfo = iter.next
  strServerName = serverInfo.serverName
  strServerType = serverInfo.serverType
  MsgBox "Server Name ["& i & "] - "& strServerName &
         "Server Type - "& strServerType
Loop
```

Digital Library datastore list schema and schema attributes

The datastore provides methods for listing the schema, in datastore DL case these are index-classes and their attributes. The following example shows you how to retrieve the list of index classes, as well as the list of attributes. The steps are:

1. Create a DXDatastoreDL
2. Connect to the Digital Library server `LIBSRVRN` using the user ID `USER1` and the password `PASSWORD`
3. List the attributes for each index class
4. Disconnect from the Digital Library server

```
Dim dsDL as new DXDatastoreDL
Dim col as DXSequentialCollection, col2 as DXSequentialCollection
Dim iter as DXSequentialIterator, iter2 as DXSequentialIterator
Dim strIndexClass as String
```

```
Dim def as DXAttributeDef
Dim i as Long, j as Long
dsDL.connect "LIBSRVRN", "USER1", "PASSWORD"
Set col = dsDL.listSchema
Set iter = col.createIterator
i = 0
Do While iter.more
  i = i + 1
  strIndexClass = iter.next
  MsgBox "index class name [" & i & "] - " & strIndexClass
  MsgBox " list attributes for " & strIndexClass & " index class"
  Set col2 = dsDL.listSchemaAttributes(strIndexClass)
  Set iter2 = col2.createIterator
  j = 0
  Do While iter2.more
    j = j + 1
    Set def = iter2.next
    MsgBox " Attribute name [" & j & "] - " & def.name
    MsgBox " datastoreType " & def.datastoreType
    MsgBox " attributeOf " & def.attributeOf
    MsgBox " type " & def.type
    MsgBox " size " & def.size
    MsgBox " id " & def.id
    MsgBox " nullable " & def.nullable
    MsgBox " precision " & def.precision
    MsgBox " scale " & def.scale
  Loop
  MsgBox " " & j & " attributes listed for " & strIndexClass & " index class"
Loop
dsDL.disconnect
```

Understanding datastore text search (Text Search Engine)

A DXDatastoreTS represents the Text Search Engine. Text Search Engine does not actually store the data, it merely index the data stored in the Digital Library datastore to support text searches on them. The results of a text search are identifiers (item-id), describing the location of the document in the Digital Library datastore. These identifiers can be used to retrieve the document from the datastore.

Text Search Engine datastore connect and disconnect

The TS datastore provides two methods for connect as well as a method for disconnect. The following examples show you the variations:

Connect using text search server TM:

```
Dim dsTS as new DXDatastoreTS
dsTS.connect "TM", "", "", ""
dsTS.disconnect
```

Connect using text search server host name and port number:

```
dsTS.connectPort "apollo", "7502", dx_CTYP_TCPIP
```

Connect using text search server host name and port number:

```
dsTS.connect "apollo", "", "", "PORT=7502; COMMTYPE=T"
```

Connect using the text search server name:

```
dsTS.connectPort "TM", "", dx_CType_TCPIP
```

Connect using text search server name and using Digital Library library server, user ID and password:

```
dsTS.connect "TM", "", "", "LIBACCESS=(LIBSRVR2,USER1,PASSWORD)"
```

Note that the last parameter, also called **connect-string**, can be used to pass a sequence of parameters in one string.

Set and get TS datastore options

The datastore TS provides some options that you can set or get, using its methods. The following example shows how to set and get the option for a TS character code set.

See the reference documentation for the complete listing of options and their descriptions.

```
Dim ccsid as long
Dim dsTS as new DXDatastoreTS
dsTS.setOption dx_OPT_TS_CCSID, dx_CCSID_00850
dsTS.setOption dx_OPT_TS_LANG, dx_LANG_ENU
dsTS.connect "TM", "", "", ""
dsTS.getOption dx_OPT_TS_CCSID, ccsid
If ccsid = dx_CCSID_00850 Then
  MsgBox "Datastore character code set is 850"
End If
dsTS.disconnect
```

Text Search Engine datastore list servers

The datastore TS provides a method to list the Text Search Engine text search servers it can connect to.

The following example shows how to retrieve the list of servers:

```
Dim dsTS as new DXDatastoreTS
Dim col as DXSequentialCollection
Dim iter as DXSequentialIterator
Dim strServerName as String, strLoc as String
Dim serverLocation as Long
Dim serverInfo as DXServerInfoTS
Dim i as Long
i = 0
Set col = dsTS.listServers
Set iter = col.createIterator
Do While iter.more
  i = i + 1
  Set serverInfo = iter.next
  strServerName = serverInfo.serverName
  serverLocation = servrInfo.serverLocation
  If serverLocation = dx_SRV_LOCAL Then
    strLoc = "LOCAL SERVER"
  ElseIf serverLocation = dx_SRV_REMOTE Then
    strLoc = "REMOTE SERVER"
  End If
  MsgBox "Server Name [" & i & "] - " & strServerName &
         "Server Location - " & strLoc
Loop
```

Text Search Engine datastore list schema

The datastore provides methods for listing the schema, in the Text Search Engine case, these are text search-indexes.

The following example shows how to retrieve the list of search indexes:

```
Dim dsTS as new DXDatastoreTS
Dim col as DXSequentialCollection
Dim iter as DXSequentialIterator
Dim strIndexName as String
Dim strLibId as String
Dim indx as DXIndexTS
Dim i as Long
dsTS.connect "TM", "", "", ""
Set col = dsTS.listSchema
Set iter = col.createIterator
i = 0
Do While iter.more
  i = i + 1
  indx = iter.next
  strIndexName = indx.indexName
  strLibId = indx.libraryId
  MsgBox "index name [" & i & "] - " & strIndexName &
         "Library - " & strLibId
Loop
dsTS.disconnect
```

Collections and iterators

The ActiveX collection classes are comprised of DXSequentialCollection, DXResults, DXParts, and DXFolder. DXSequentialCollection contains VARIANTs as members, whereas DXResults and DXFolder contains DXDDOs, and DXParts contains DXBlobDLs and DXClobDLs. DXSequentialCollection is mainly used for retrieving datastore information such as schemas and servers, so it only provides a bare minimum set of methods. The others provide methods for adding, retrieving, removing, and replacing its members. Iterators are provided to let you iterate over collection members. An iterator is created by calling the method createIterator on the collection. Typically, you would write the following code to iterate over a collection:

```
Dim iter as DXSequentialIterator
Set iter = coll.createIterator 'create an iterator for coll
Do While iter.more 'while there are more members
  Set member = iter.next 'get the current member and advance iter to the next member
  'Do something with the member
  ...
Loop
```

DXSequentialIterator provides additional methods to move the iterator bidirectionally. The above code could be rewritten as follows:

```
Dim iter as DXSequentialIterator
Set iter = coll.createIterator 'create an iterator for coll
Do While iter.more     'while there are more members
  Set member = iter.at 'get the current member
  'Do something with the member
  ...
  iter.setToNext       'advance to the next position
Loop
```

All except DXSequentialCollection allow you to do some operations at the current member before moving along to the next member. Such an operation would be, for example, replacing a member with a new one, or removing it. Note that when you remove the current member, the iterator will be advanced to the next member. Therefore, when removing member inside a loop construct, do not forget to take this fact into account, that is, you need to perform a check, like the following:

```
...
If removeCondition = True Then
  coll.removeElementAt    'remove current member, do not advance iter since
                          'it is advanced to the next after the removal operation
Else
  iter.setToNext          'no removal, advance the iterator to the next position
End If
```

The above check is necessary to avoid skipping the next member after removing the current one.

Using DDO (Dynamic Data Object)

DXDDO can be regarded as a container of attributes. An attribute, or also called data-item (these terms will be used interchangeably), has a name, value, and properties. Each attribute is identified by a data_id, which is a number started from 1, up to the total number of attributes in the DDO. Since the number, name, value, and property of an attribute may vary, DXDDO provides flexible mechanisms to represent data originated from a variety of datastores and format, be it items from different index classes in Digital Library, or rows from different tables in a relational database, etcetera. The DXDDO itself can have properties which apply to the whole DXDDO, instead of only to one particular attribute. You need to associate a DXDDO with a datastore to be able to call the methods add, retrieve, update, and delete, to send its attributes into the datastore and retrieve them back. This is done by calling the **setDatastore** method. Every DXDDO has a persistent object identifier (PID), which contains information to locate the data-items in the datastore.

Pid

All DDO must have a persistent object identifier or Pid. It contains information about datastore name, datastore type, id, and object type. The id identifies the location of the DDO's persistent data in the datastore. The object-type indicates the index-class the DDO belongs to. For the Digital Library datastore, this id is the item-id. The item-id is one of the most important items of information for the retrieve, update, and delete operations. For add, the item-id will be created and returned by the datastore. To create a DDO of a known item for retrieval, you could do the following:

```
Dim dsDL as new DXDatastoreDL      'create a Digital Library datastore
Dim ddo as new DXDDO
ddo.setObjectType "GRANDPA"        'set the index-class name it belongs to
ddo.setPid "LN#U5K6ARLGM3DB4"      'set the item-id
ddo.setDatastore dsDL              'associate ddo with dsDL
```

Then you can connect to the datastore and call retrieve to retrieve this DDO. This topic will be discussed further in the section called "Retrieving a document or folder" on page 183.

Adding data-items and properties

Suppose the index class `GRANDPA` has the following two attributes:

Attribute:	**data_id=1**	**2**
Name:	Title	Subject
Type:	String	String
Can be null?	no	yes

then, you can represent the above information in a DXDDO as follows:

```
Dim dsDL as new DXDatastoreDL      'create a Digital Library datastore
Dim cddo as new DXDDO              'create a DDO to hold an object type
cddo.setObjectType "GRANDPA"       'set the index-class name it belongs to
cddo.setDatastore dsDL             'associate ddo with dsDL
Dim data_id as Integer

'Add the first attribute
data_id = cddo.addData("Title")    'add a new attribute named "Title"

'Add a property named: dx_Property_Type, set to value : variable length string
cddo.addDataPropertyAndValue data_id, dx_Property_Type, dx_VString

'Add a property named: dx_Property_Nullable, set to value : boolean false
cddo.addDataPropertyAndValue data_id, dx_Property_Nullable, False

'Add the second attribute
data_id = cddo.addData("Subject")  'add a new attribute named "Subject"

'Add a property named: dx_Property_Type, set to value : variable length string
cddo.addDataPropertyAndValue data_id, dx_Property_Type, dx_VString

'Add a property named: dx_Property_Nullable, set to value : boolean true
cddo.addDataPropertyAndValue data_id, dx_Property_Nullable, True
```

The above example illustrates the standard properties that an attribute should have, namely dx_Property_Type and dx_Property_Nullable. In practice, you may have as many additional properties as required by your application.

Adding properties to a DDO

So far, the above DXDDO has all the attributes information it needs to have. However, there is no information to indicate the kind of DXDDO it is; it must be either a document or a folder. This information is recorded under DXDDO properties, as opposed to attribute properties.

```
cddo.addProperty dx_Property_Item_Type, dx_Document 'it is a document
```

Setting and getting data_item values

Once you created a data_item and its properties, you can set its value:

```
'Set Title value to the given string assume we know the data_id for
'the data_item "Title" is 1
cddo.setData 1, "One dark and stormy night"

'Set Subject value to the given string assume we do not know the data_id

'for the data_item "Subject"
data_id = cddo.dataId("Subject") 'find data_id for data_item named "Subject"
cddo.setData data_id, "Mystery"
```

To get the value back, you use `getData` method:

```
Dim data as String
data = cddo.getData(1)
MsgBox "Title = " & data 'displays "One dark and stormy night"
MsgBox "Subject = " & cddo.getData(data_id) 'displays "Mystery"
```

Getting the properties

When processing a DXDDO, the first thing you want to know is probably its type, a document or a folder. The following code illustrates such a check:

```
Dim prop_id as Integer
Dim myType as Integer
prop_id = cddo.propertyId(dx_Property_Item_Type)
If prop_id > 0 Then
   myType = cddo.getProperty(prop_id)
   Select Case myType
    case dx_Document
      'process document
      ...
    case dx_Folder
      'process folder
      ...
  End Select
End If
```

To retrieve properties of an attribute, you need to have the data_id of the attribute:

```
data_id = cddo.dataId("Title") 'get data_id of Title
'How many prop does it have
Dim number_of_data_prop as Integer
number_of_data_prop = cddo.dataPropertyCount(data_id)
'Displays all data properties belonging to this attribute
'Notice that the loop index starts from 1, 1 <= i <= number_of_data_prop
Dim i as Integer
For i = 1 To number_of_data_prop
    MsgBox Str(i) & " Property Name =
    " & cddo.getDataPropertyName(data_id, i) & _
    " value = " & cddo.getDataProperty(data_id, i)
Next
```

It is important to notice that data_id and property_id starts from 1; if you specify a zero, you will get an exception.

Displaying the whole DDO

During the course of application development, a developer may need to display the content of a DXDDO for debugging purposes. The following code will do the task:

```
Dim number_of_attribute as Integer
number_of_attribute = cddo.dataCount
Dim number_of_prop as Integer, k as Integer, j as Integer
Dim number_of_data_prop as Integer, i as Integer
number_of_prop = cddo.propertyCount
'List DDO properties
For k = 1 To number_of_prop
    MsgBox Str(k) & " Property Name =
    " & cddo.getPropertyName(k) & ", value =
    " _ & cddo.getProperty(k)
Next
'List data-items and their properties
For i = 1 To number_of_attribute
  MsgBox Str(i) & " Attr. Name =
    " & cddo.getDataName(i) & ", value =
    " & _cddo.getData(i)
  number_of_data_prop = cddo.dataPropertyCount(i)
  For j = 1 To number_of_data_prop
    MsgBox Str(j) & " Data Prop. Name =
      " & cddo.getDataPropertyName(i, j) _ & ", value =
      " & cddo.getDataProperty(i, j)
  Next
Next
```

DDO representing Digital Library information

A DDO associated with DXDatastoreDL has some specific information to represent Digital Library document metaphor: document, folder, parts, item, item-id, rank, etc. The following subsection explains how these information are recorded.

DDO properties

As discussed before, the type of item, either a document or folder, is recorded as a property of the DDO, under the name dx_Property_Item_Type. To get the item-type of the DDO, you call:

```
Dim a as VARIANT
Dim item_type as Integer
a = cddo.getPropertyByName(dx_Property_Item_Type)
If IsEmpty(a) Then
Else
  item_type = a
End If
```

After the call, the item_type will be equal to dx_Document for document, or dx_Folder if it is a folder. The if statement is a check to make sure that the property exists. See “Adding properties to a DDO” on page 164 and “Getting the properties” on page 164, for more information.

Representing Digital Library documents

A DDO representing a Digital Library document has a property dx_Property_Item_Type equal to dx_Document. Its Pid contains index-class name as the object-type, and item-id in the Pid’s id.

The parts inside a document are represented as a DXParts object, which is a collection of blobs, each of which represented as DXBlobDL object. A document DDO has a specific attribute with a reserved name dx_DKPARTS, whose value is a DXParts object.

To get to each part in a document, you need to pull out DXParts first, then create an iterator to iterate over each part as you would normally process a collection. If the document does not have a part at all, the dx_DKPARTS will have a null value; the dx_DKPARTS attribute will always present in a document DDO created by the datastore.

Documents associated with a combination of parametric and text query (combined query) may have a transient attribute dx_DKRANK the value of which is a long integer rank computed by the text search engine. Sample code to create and process DXParts is given in the sections “Creating and using dx_DKPARTS attribute” on page 167 and “Retrieving a document or folder” on page 183.

Representing Digital Library folders

A DDO representing a Digital Library folder has a property dx_Property_Item_Type equal to dx_Folder. Similar to a document DDO, its Pid contains index-class name as the object-type, and item-id in the Pid’s id.

The folder table-of-contents inside a DL folder is represented as a DXFolder object, which is a collection of DDOs each of which represents an item, either a document

or another folder, belongs to this DL folder. A folder DDO has a specific attribute with a reserved name dx_DKFOLDER, which value is a DXFolder object.

To get to each DDO member of the folder, you need to pull out DXFolder first, then create an iterator to iterate over each item member as you would normally process a collection. If the folder does not have a member at all, the dx_DKFOLDER will have a null value; but the dx_DKFOLDER attribute will always present in a folder DDO created by the datastore. Sample code to create and process DXFolder is given in the sections “Creating and using dx_DKFOLDER attribute” on page 168 and “Retrieving a document or folder” on page 183.

DDO representing Text Search Engine information

A DDO associated with DXDatastoreTS has some specific information to represent text search results. It does not have a property item-type like in DXDatastoreDL. The format of id is also different. A DDO resulting from a text query corresponds to a text part inside a Digital Library item. It has a set of standard attributes, described below.

dx_DKDLITEMID
: the Digital Library item-id where this text part belongs. This item-id is used to retrieve the whole item from Digital Library datastore.

dx_DKPARTNO
: a long integer part-number of this text part. It is used together with the item-id to retrieve this part from Digital Library datastore.

dx_DKREPTYPE
: the Digital Library rep-type of this text part. This attribute, when used in conjunction with the item-id and part-number, retrieves the part from the Digital Library datastore.

dx_DKRANK
: a long integer rank signifies the relevance of this part to the results set of the text query. A higher rank means better match. See the Text Search Engine manual for further information.

dx_DKSIZE
: an integer number of word-occurrences. See the Text Search Engine manual for further information.

dx_DKRCNT
: an integer number of matches. See the Text Search Engine manual for further information.

The Pid for a text search DDO has the following information:

- datastore type – TS
- datastore name – the server name used to connect to the datastore
- object type – text search index
- id – Text Search Engine document ID

Creating and using dx_DKPARTS attribute

As mentioned earlier, dx_DKPARTS attribute in a DDO represents the collection of parts in a document. The value of this attribute is a DXParts object, a collection of XDOs. This attribute is set during DDO retrieve; or it can be created and set by you, as shown below:

```
Dim dsDL as new DXDatastoreDL
Dim parts as new DXParts       'create a new DXParts, collection of parts
Dim blob as new DXBlobDL       'create a new XDO blob
blob.init dsDL
blob.setPartId 5               'set part number to 5
blob.setPid "LN#U5K6ARLGM3DB4" 'the item-id this part belongs to
blob.setContentClass dx_CC_GIF 'set content class type GIF
blob.setRepType dx_DK_REP_NULL  'set rep type for the part
blob.setContentFromClientFile "choice.gif" 'set the blob's content
blob.setInstanceOpenHandler "netscape", True 'the viewer program
parts.addElement blob          'add the blob to the parts collection
... 'create and add some more blobs to the collection as necessary
Dim ddo as new DXDDO           'create a ddo
... 'sets some of its attributes
ddo.addProperty dx_Property_Item_Type, dx_Document
'Create dx_DKPARTS attribute and sets it to refer to the DXParts object
Dim data_id as Integer
data_id = ddo.addData dx_DKPARTS 'add attribute dx_DKPARTS
ddo.addDataPropertyAndValue data_id, dx_Property_Type, dx_Collection_XDO
ddo.addDataPropertyAndValue data_id, dx_Property_Nullable, True
ddo.setData data_id, parts     'sets the attribute value
```

Once a DXParts is set to be an attribute value of a DDO, the DDO owns it and will take care of its deletion. Getting back the parts from a DDO is a lot simpler:

```
data_id = ddo.dataId(dx_DKPARTS)  'get dx_DKPARTS data-id
If data_id = 0 Then
   MsgBox " parts data-item not found"
End If
Dim col as DXParts
Set col = ddo.getData(data_id)   'get the parts collection
'Create iterator and process the part collection member one by one
If col Is Nothing Then
Else
  Dim iter as DXSequentialIterator
  Dim blob as DXBlobDL
  Set iter = col.createIterator
  Do While iter.more
     Set blob = iter.next
     If blob Is Nothing Then
     Else
      blob.open                  'display the blob using the viewer
      ... 'other processing
     End If
  Loop
End If
```

Creating and using dx_DKFOLDER attribute

In a folder DDO, the dx_DKFOLDER attribute represents a collection of folders and documents belonging to this folder. The value of this attribute is a DXFolder object, a collection of DDOs. Similar to dx_DKPARTS, dx_DKFOLDER is set during DDO retrieve, or it can be created and set by you, as shown below:

```
Dim dsDL as new DXDatastoreDL
Dim folder as new DXFolder         'create a new DXFolder, collection of DDO
Dim member as new DXDDO            'create the first member of this folder
... 'sets the member DDO attributes and properties
folder.addElement member           'add member to the folder collection
... 'create and add some more member DDO to the DDO collection as necessary
Dim ddo as new DXDDO               'create a folder ddo
... 'sets some of its attributes
ddo.addPropertyAndValue dx_Property_Item_Type, dx_Folder
'Create dx_Folder attribute and sets it to refer to the DXFolder object
Dim data_id as Integer
data_id = ddo.addData(dx_DKFOLDER)  'add attribute dx_DKFOLDER
```

```
ddo.addDataPropertyAndValue data_id, dx_Property_Type, dx_Collection_DDO
ddo.addDataPropertyAndValue data_id, dx_Property_Nullable, True
ddo.setData data_id, folder        'sets the attribute value
```

Once a DKFolder is set to be an attribute of a DDO, the DDO owns it and will take care of its deletion. Getting back the folder from a DDO is also a lot simpler:

```
data_id = ddo.dataId(dx_DKFOLDER)     'get dx_DKFOLDER data-id
If data_id = 0 Then                  'folder not found
  MsgBox "folder data-item not found"
End If
Dim col as DXFolder
Set col = ddo.getData(data_id)        'get the parts collection
'Create iterator and process the DDO collection member one by one
If col Is Nothing Then
Else
  Dim item as DXDDO
  Dim iter as DXSequentialIterator
  Set iter = col.createIterator
  Do While iter.more
     Set item = iter.next
     If item Is Nothing Then
     Else
      item.retrieve                  'process the member DDO
     ... 'other processing
     End If
  Loop
End If
```

Deleting a DDO

When a DXDDO is deleted, the persistent copy in the datastore is unchanged. In contrast, the method **del** in the DDO deletes the persistent copy in the datastore, and its representation in memory is unchanged.

Using XDO

An XDO represents a single part object in Digital Library. There are two types of XDO: DXBlobDL and DXClobDL. DXBlobDL is for binary object and DXClobDL is for character object. Both DXBlobDL and DXClobDL require datastore DXDatastoreDL as input to initiate the object instance. An XDO needs to have a Pid in order to store its data persistently. `itemId` and `partId` are required for XDO to locate the persistent data in Digital Library datastore.

XDO as a part of DDO instead of stand alone XDO

In Digital Library when DDO is a document which is a collection of part objects, an XDO represents a single part object. You can manipulate the XDO as a part of DDO or as a stand alone object. To manipulate it as a stand alone object you need to know the existing itemId for the XDO. To handle as a part of DDO you don't need to know the itemId; the DDO will provide the itemId for the XDO.

XDO as part of DDO

Following are the major steps to relate the XDO with DDO:

```
'Create DDO
Dim ddo as new DXDDO
ddo.setObjectType indexClassName
ddo.setDatastore dsDL
ddo.addPropertyAndValue dx_Property_Item_Type, dx_Document
```

```
...
...
Dim parts as new DXParts
'Create XDO
Dim axdo as new DXBlobDL
axdo.init dsDL
axdo.setPartId partId                        'set partId
axdo.setContentClass dx_CC_GIF
axdo.setAffiliatedType dx_Base
axdo.setContentFromClientFile imageNames(i)
'Add XDO to the DXParts collection
parts.addElement axdo
...
...
'Add DDO
 dataId = ddo.addData(dx_DKPARTS)
ddo.addDataPropertyAndValue dataId, dx_Property_Type, dx_Collection_XDO
ddo.setData dataId, parts
ddo.add
```

XDO data members

You should be aware of some important information about an XDO, such as RepType, ContentClass, AffiliatedType, AffiliatedData, SearchEngine, SearchIndex and SearchInfo. Without setting any values for these members, the object content will not be indexed by a search engine and the default values will be provided as follows:

```
RepType is  dx_DK_REP_NULL
ContentClass is  dx_CC_UNKNOWN
AffiliatedType is dx_Base
AffiliatedData is NULL
```

Tip: For the available values of ContentClass, see the enumerated constant CONTENT_CLASS.

Indexing XDOs by search engine

If you want the object content to be indexed by the search manager, the values of SearchEngine, SearchIndex and SearchInfo are required. Please be aware that the SearchIndex value is a combination of two names: search service name and search index name. For example, in System Admin Client you have the search server named `TM` and you defined a search index named `TMINDEX` under it, then the correct value for the SearchIndex is `TM-TMINDEX`.

Programming tips

The combination of itemId, partId and repType is the key to identify an XDO. To handle a stand-alone XDO you need to provide the itemId and partId while repType is optional since the system will provide a default value for it. During add operation, if you set partId to `0`, the system will assign an available partId for it and you can retrieve the partId value after add if you want to do some other operation with that object later.

Note: There are two situations for which a valid part ID is required, and you cannot set the part ID to zero:

1. Adding a part to be indexed by the search manager
2. Adding a large object that will be divided into MAXPIECE-set chunks

Stand-alone XDO

1. Add an XDO from buffer:

```
Dim dsDL as new DXDatastoreDL
Dim itemId as String, fileName as String
Dim partId as Integer
partId = 37                          'partId 37 not being used yet
itemId = "CPPIORH4JBIXWIY0"          'existing itemId
fileName = "g:\\test\\cheetah.gif"  'file to be added
dsDL.connect "LIBSRVRH", "FRNADMIN", "PASSWORD" 'connection to datastore
Dim axdo as new DXBlobDL
axdo.init dsDL               'create XDO
axdo.setPartId partId                'set partId
axdo.setPid itemId                   'set itemId
axdo.setContentClass dx_CC_GIF       'set ContentClass
axdo.setContentFromClientFile fileName 'set file content to buffer area
axdo.add                             'add from the buffer
MsgBox "after add partId=" & axdo.getPartId
dsDL.disconnect                       'disconnect from datastore
```

2. Add an XDO from file (use the code from the preceding example and replace the following statements):

```
axdo.setContentFromClientFile fileName
axdo.add
```

   with:

```
axdo.add fileName
```

3. Add an XDO to be indexed by Text Search Engine:

```
Dim partId as Integer
partId = 0                          'let system decide the partId
Dim itemId as String, fileName as String
itemId = "CPPIORH4JBIXWIY0"         'existing itemId
fileName = "g:\\test\\cheetah.gif" 'file content to be indexed
Dim dsDL as new DXDatastoreDL       'required datastore
dsDL.connect "LIBSRVRH", "FRNADMIN", "PASSWORD" 'connection to datastore
Dim axdo as new DXBlobDL
axdo.init dsDL                      'create XDO
axdo.setPartId partId               'set partId
axdo.setPid itemId                  'set itemId
axdo.setContentClass dx_CC_ASCII    'set ContentClass to text

'---set searchEngine ----- (deprecated)
axdo.setSearchEngine "SM"
axdo.setSearchIndex "TM-TMINDEX"
axdo.setSearchInfo "ENU"

'---set searchEngine ----- (using an extension object)
Dim srcheng As New DXSearchEngineInfoDL
srcheng.setSearchEngine "SM"
srcheng.setSearchInfo "ENU"
srcheng.setSearchIndex "TMMUF-TMINDEX"
xdo.setExtension srcheng

axdo.setContentFromClientFile fileName     'set file content to buffer area
axdo.add                                   'add from the buffer
MsgBox "after add partId = " & axdo.getPartId 'display the partId after add
axdo.retrieve                               'retrieve object
axdo.setInstanceOpenHandler "notepad", true 'set open viewer
axdo.open                                   'open content with viewer
dsDL.disconnect                             'disconnect from datastore
```

4. Add an annotation type XDO:

   To add an annotation object, insert following statements before the add operation.

```
axdo.setAffiliatedType dx_Annotation
axdo.setAffiliatedData 14, 1, 5, 5
```

5. Index an existing XDO using search engines:

   This example shows how to index an existing XDO using search engines.

```
Private Sub cout(str As String)
Results.Text = Results.Text & Chr$(13) & Chr$(10) & str
End Sub

Private Sub Run_Click()

itemId = "N2JJBERBQFK@WTVL"
partId = 13
Set dsDL = New DXDatastoreDL
Set xdo = New DXBlobDL
xdo.init dsDL
xdo.setPartId partId
xdo.setPid itemId

cout "Check if was indexed, if not then do indexing:"
Dim catflag As Boolean
catflag = xdo.isCategoryOf(dx_Indexed_Object)
cout "isCategoryOf indexed obj = " & catflag
If (catflag) Then
    cout "==alreay indexed, do getExtension to show informations:"
    Dim srcheng2 As Variant
    Set srcheng2 = xdo.getExtension("DXSearchEngineInfoDL")
    On Error GoTo Errors
    cout " srcheng2.getSearchEngine=" & srcheng2.getSearchEngine
    cout " srcheng2.getSearchIndex=" & srcheng2.getSearchIndex
    cout " srcheng2.getSearchInfo=" & srcheng2.getSearchInfo
    cout " srcheng2.getTextIndex=" & srcheng2.getTextIndex
    cout " srcheng2.getCatalog=" & srcheng2.getCatalog
    cout " srcheng2.getDataBase=" & srcheng2.getDataBase
    cout " srcheng2.getServerName=" & srcheng2.getServerName
    cout " srcheng2.getSearchClassId=" & srcheng2.getSearchClassId
    cout " srcheng2.getSearchTimestamp=" & srcheng2.getSearchTimestamp
    cout " xdo.retrieveObjectState=" & xdo.retrieveObjectState(dx_Indexed_Object)
Else
    cout "==not indexed, do new DXSearchEngingInfoDL to prepare:"
    Dim srcheng As New DXSearchEngineInfoDL
    srcheng.setSearchEngine "SM"
    srcheng.setSearchInfo "ENU"
    srcheng.setSearchIndex "TMMUF-TMINDEX"
    cout " set srcheng extension obj..."
    xdo.setExtension srcheng
    On Error GoTo Errors
    cout " about to setToBeIndexed..."
    xdo.setToBeIndexed
    On Error GoTo Errors
    cout " setToBeIndexed success..."

End If
    cout "isCategoryOf media obj=" & xdo.isCategoryOf(dx_Media_Object)
    cout "isCategoryOf indexed obj=" & xdo.isCategoryOf(dx_Indexed_Object)

Exit Sub
Errors:
    cout "Errors:" & str(Err.Number) & Err.Description

End Sub
```

6. Add, retrieve, and delete an XDO Media Object:

   This example shows how to add, retrieve, and delete an XDO Media Object.

```
Private Sub cout(str As String)
Results.Text = Results.Text & Chr$(13) & Chr$(10) & str
End Sub

Private Sub Add_Click()

'following itemId should be already exist
'modify it with your own known itemId
'itemId = "POL025$BQVH$TZBS"
itemId = "FPN3ZP$MPZI@U0C0"
partId = 95
Set dsDL = New DXDatastoreDL
Set xdo = New DXBlobDL
xdo.init dsDL
xdo.setPartId partId
xdo.setPid itemId

'media content class is 209
xdo.setContentClass dx_CC_IBMVSS
cout "Do new DXMediaStreamInfoDL.."
Dim aVS As New DXMediaStreamInfoDL
aVS.setMediaFullFileName "/icing.mpg1"
aVS.setMediaObjectOption dx_VS_SingleObject
aVS.setMediaHostName "vcharger1.stl.ibm.com"
aVS.setMediaUserId "root"
aVS.setMediaPassword "video2"
aVS.setMediaNumberOfUsers 1
aVS.setMediaAssetGroup "AG"
aVS.setMediaType "MPEG1"
aVS.setMediaResolution "SIF"
aVS.setMediaStandard "NTSC"
aVS.setMediaFormat "SYSTEM"
cout "set mediaStream extension obj..."
xdo.setExtension aVS
On Error GoTo Errors
cout "about to add media object..."
xdo.Add
On Error GoTo Errors
cout "add successfully..."
Add.Enabled = False
Exit Sub

Errors:
  cout "Errors:" & str(Err.Number) & Err.Description

End Sub

Private Sub Retrieve_Click()

  cout "itemId=" & xdo.getPid
  cout "partId=" & xdo.getPartId
  cout " "
  Dim state As Integer
  Dim mflag As Boolean
  mflag = xdo.isCategoryOf(dx_Media_Object)
  On Error GoTo Errors
  cout "isCategoryOf media obj = " & mflag
  If (mflag) Then
      cout "Get mediaStream extension obj..."
      Dim aVS2 As Variant
      Set aVS2 = xdo.getExtension("DXMediaStreamInfoDL")
      On Error GoTo Errors
      cout " aVS2.getMediaCopyRate=" & aVS2.getMediaCopyRate
      cout " aVS2.getMediaInvalidCommands=" & aVS2.getMediaInvalidCommands
      cout " aVS2.getMediaDurSeconds=" & aVS2.getMediaDurSeconds
      cout " aVS2.getMediaDurFrames=" & aVS2.getMediaDurFrames
      cout " aVS2.getMediaFrameRate=" & aVS2.getMediaFrameRate
```

```
        cout " aVS2.getMediaBitRate=" & aVS2.getMediaBitRate
        cout " aVS2.getMediaNumberOfUsers=" & aVS2.getMediaNumberOfUsers
        cout " aVS2.getMediaAssetGroup=" & aVS2.getMediaAssetGroup
        cout " aVS2.getMediaType=" & aVS2.getMediaType
        cout " aVS2.getMediaResolution=" & aVS2.getMediaResolution
        cout " aVS2.getMediaStandard=" & aVS2.getMediaStandard
        cout " aVS2.getMediaFormat=" & aVS2.getMediaFormat
        cout " aVS2.getMediaState=" & aVS2.getMediaState
        state = xdo.retrieveObjectState(dx_Media_Object)
        cout "xdo.retrieveObjectState(media) =" & state
        On Error GoTo Errors
    End If

    cout "===before retrieve==="
    cout " getLength=" & xdo.getLength
    cout " getSize=" & xdo.getSize
    cout " updatedTimestamp=" & xdo.getUpdatedTimestamp
    cout " createdTimestamp=" & xdo.getCreatedTimestamp
    cout " mimeType=" & xdo.mimeType
    cout "===about to retrieve as filedx.ivs ==="
    xdo.Retrieve ("filedx.ivs")
    On Error GoTo Errors
    cout "  retrieve successfully"
    cout "===after retrieve==="
    cout " getLength=" & xdo.getLength
    cout " getSize=" & xdo.getSize
    cout " getContentClass=" & xdo.getContentClass
Exit Sub

Errors:
    cout "Errors:" & str(Err.Number) & Err.Description

End Sub

Private Sub Delete_Click()

    cout "itemId=" & xdo.getPid
    cout "partId=" & xdo.getPartId
    Dim mflag As Boolean
    mflag = xdo.isCategoryOf(dx_Media_Object)
    cout "isCategoryOf media obj = " & mflag
    If (mflag) Then
        cout "setting delete Option...."
        xdo.setOption dx_Opt_DL_Delete_Option, dx_Delete_NoDropitemMediaAvail
        Dim ln As Long
        xdo.getOption dx_Opt_DL_Delete_Option, ln
        cout "getOption deleteOpt=" & ln
    End If
      cout "===about to call delete==="
      xdo.del
      cout "  delete successfully"
      cout "===after  delete==="
      mflag = xdo.isCategoryOf(dx_Media_Object)
      cout "  isCategoryOf media obj = " & mflag
      Add.Enabled = True
      Retrieve.Enabled = False
      Delete.Enabled = False
Exit Sub

Errors:
    cout "Errors:" & str(Err.Number) & Err.Description

End Sub
```

XDO in datastore

To retrieve, update or delete the object in datastore, you should provide the correct itemId, partId and repType to identify the object.

```
Dim partId as Integer
partId = 17                          'partId of object
Dim itemId as String, fileName as String
itemId = "CPPIORH4JBIXWIY0"          'existing itemId
fileName = "g:\\test\\choice.gif"    'file to be updated
Dim dsDL as new DXDatastoreDL        'required datastore
dsDL.connect "LIBSRVRH", "FRNADMIN", "PASSWORD" 'connection to datastore
Dim axdo as new DXBlobDL
axdo.init dsDL                       'create XDO
axdo.setPartId partId                'set partId
axdo.setPid itemId                   'set itemId
axdo.retrieve                        'retrieve the object
axdo.setContentFromClientFile fileName 'set file content to buffer area
axdo.update                          'update the object with buffer data
axdo.retrieve "new.gif"              'retrieve content to a file
axdo.del                             'delete object from datastore
dsDL.disconnect                      'disconnect from datastore
```

Querying the datastore

You can issue a query to a datastore and get back results in the form of a DXResultSetCursor or DXResults. Also, you can use a query object which represents a query and can be associated with zero or more datastores. With the help of its datastores, the query objects performs query processing tasks, such as preparing and executing a query, monitoring the status of query execution and storing the results. The result of a query is usually a DXResults object. There are three types of query objects, Parametric, Text and Combined. The Combined query is composed of multiple Text and Parametric queries. The datastore uses two ways to run a query execute and evaluate. The execute returns DXResultSetCursor while evaluate returns DXResults. The DXResultSetCursor is used to handle a large result set, as well as perform delete and update operations on the current position of the result set cursor. You can use the fetchNextN method to fetch a group of objects into a collection. The DXResults contains all of the results from the query. You can iterate over the items in the collection forward or backwards. This collection is also queryable and can be used as a scope for another query. See “Queryable collection” on page 189 and “Combined query” on page 190 for more information.

Parametric query

This section explains the ways in which you use the parametric query function.

Formulating a parametric query

Create a query string representing a query against the index class GP2DLS2. The condition of the query is to search for all documents or folders with an attribute of DLSEARCH_DocType <> NULL. The maximum number of results that will be returned is 5. The content is set to yes so that contents of the document or folder will be returned. Specify that Digital Library use dynamic SQL for this query and that folders and documents are to be displayed. If the attribute name has more than one word or in a DBCS language it should be enclosed in single quotes. If the attribute value is in a DBCS language it should be enclosed in double quotes.

```
Dim cmd as string
cmd = "SEARCH=(INDEX_CLASS=GP2DLS2,"
cmd = cmd & "MAX_RESULTS=5,"
cmd = cmd & "COND=(DLSEARCH_DocType <> NULL));"
cmd = cmd & "OPTION=(CONTENT=YES;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;"
cmd = cmd & "TYPE_FILTER=FOLDERDOC)"
```

Formulating a parametric query on multiple criteria

The parametric query allows you to specify more than one search criteria. The following example shows how to specify a query against two index classes.

```
Dim cmd as String
cmd = "SEARCH=(INDEX_CLASS=GP2DLS2,MAX_RESULTS=3,";
cmd += cmd & "COND=(DLSEARCH_DocType <> NULL);";
cmd += cmd & "INDEX_CLASS=GP2DLS1,MAX_RESULTS=8,";
cmd += cmd & "COND=('First name' == \"Robert\"));";
cmd += cmd & "OPTION=(CONTENT=YES;";
cmd += cmd & "TYPE_QUERY=DYNAMIC;";
cmd += cmd & "TYPE_FILTER=FOLDERDOC)";
```

Executing a parametric query

The steps are:

- Create a DXDatastoreDL
- Connect to the Digital Library server `LIBSRVRN` using the user ID `USER1` and the password `PASSWORD`
- Create a parametric query. The query specifies to find all documents for the GP2DLS3 index class with the attribute last name equal to `SUMMERS`
- Execute the query and loop through the results using DXResults
- Disconnect from the Digital Library server

```
Dim dsDL as new DXDatastoreDL
Dim qry as DXParametricQuery
Dim results as DXResults
Dim iter as DXSequentialIterator
Dim item as DXDDO
dsDL.connect "LIBSRVRN", "USER1", "PASSWORD"
Dim cmd as String
cmd = "SEARCH=(INDEX_CLASS=GP2DLS3,"
cmd = cmd & "COND=('last name' == ""SUMMERS""));"
cmd = cmd & "OPTION=(CONTENT=YES;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;TYPE_FILTER=DOC)"
Set qry = dsDL.createQuery(cmd)
qry.execute
Set results = qry.result
Set iter = results.createIterator
Do While iter.more
   Set item = iter.next
   'Do something with the DDO...
Loop
dsDL.disconnect
```

Executing a Digital Library query from the datastore

The steps are:

- Create a DXDatastoreDL
- Connect to the Digital Library server `LIBSRVRN` using the user ID `USER1` and the password `PASSWORD`
- Specify the query to find all folders and/or documents for the GP2DLS4 index class

- Execute the query and loop through the results using DXResultSetCursor. A null is returned from fetchNext if the cursor is pass the last item in the result
- Disconnect from the Digital Library server

```
Dim dsDL as new DXDatastoreDL
Dim cur as DXResultSetCursor
dsDL.connect "LIBSRVRN", "USER1", "PASSWORD"
Dim cmd as String
cmd = "SEARCH=(INDEX_CLASS=GP2DLS4);"
cmd = cmd & "OPTION=(CONTENT=YES;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;TYPE_FILTER=FOLDERDOC)"
Set cur = dsDL.execute(cmd)
Dim item as DXDDO
Do While cur.isValid
   Set item = cur.fetchNext
   'Do something with the ddo item
Loop
cur.destroy    'in case the variable is being used by another query
dsDL.disconnect
```

Evaluate a Digital Library query from the datastore

- Create a DXDatastoreDL
- Connect to the Digital Library server `LIBSRVRN` using the user ID `USER1` and the password `PASSWORD`
- Specify the query to find all folders and/or documents for the GP2DLS5 index class with a condition of DLSEARCH_Date >= 1995 and DLSEARCH_Date <= 1996. Since content is equal to NO only the document and folder PIDs (persistent identifier) are set
- Evaluate the query and loop through the results using DXResults
- Disconnect from the Digital Library server

```
Dim dsDL as new DXDatastoreDL
Dim results as DXResults
Dim iter as DXSequentialIterator
Dim item as DXDDO
dsDL.connect "LIBSRVRN", "USER1", "PASSWORD"
Dim cmd as String
cmd = "SEARCH=(INDEX_CLASS=GP2DLS5,;"
cmd = cmd & "COND=((DLSEARCH_Date >= ""1995"") AND "
cmd = cmd & "(DLSEARCH_Date <= ""1996"")));"
cmd = cmd & "OPTION=(CONTENT=NO;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;TYPE_FILTER=FOLDERDOC)"
Set results = dsDL.evaluate(cmd)
Set iter = results.createIterator
Do While iter.more
   Set item = iter.next
   'Do something with the DDO item
Loop
```

A complete sample of a parametric query is provided in the samples directory.

Text query

This section explains the ways in which you use the text query function.

Formulating a text query

Create a query string representing a query against the text index `TMINDEX`. The condition of the query is to search for all text documents with the word `UNIX` or `member`. The maximum number of results that will be returned is 5.

```
Dim cmd as String
cmd = "SEARCH=(COND=(UNIX OR member));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)"
```

Formulating a text query on multiple indexes

The text query allows you to specify that a search is to be performed against more than one index. The following example shows how to specify a query against two indexes.

Important: If you specify more than one text search index in the query, the indexes must be the same type. For example, you can specify two precise indexes in the query, but you cannot specify a precise index and a linguistic index within the query.

```
Dim cmd as String
cmd = "SEARCH=(COND=(UNIX OR member));";
cmd += cmd & "OPTION=(SEARCH_INDEX=(TMINDEX,TMINDEX2); MAX_RESULTS=5)";
```

Executing a text query

The steps are:

- Create a DXDatastoreTS
- Connect to the Text Search Engine server TM
- Create a text query. The query specifies to find all documents with the phrase `UNIX operating` and the word `system`
- Execute the query and loop through the results using DXResults
- Disconnect from the Text Search Engine server

```
Dim dsTS as new DXDatastoreTS
Dim qry as DXTextQuery
Dim iter as DXSequentialIterator
Dim item as DXDDO
Dim results as DXResults
dsTS.connect "TM", "", "", ""
Dim cmd as String
cmd = "SEARCH="
cmd = cmd & "(COND=('UNIX operating' AND system));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"
Set qry = dsTS.createQuery(cmd)
qry.execute
Set results = qry.result
Set iter = results.createIterator
Do While iter.more
   Set item = iter.next
   'Do something with the DDO item
Loop
dsTS.disconnect
```

Executing a Text Search query from the datastore

The steps are:

- Create a DXDatastoreTS
- Connect to the Text Search Engine server host name `zebra` and port number `7502`
- Specify the query to find all text documents with the free text `Web site`
- Specify the query to find all text documents with the free text `Web site`
- Execute the query and loop through the results using DXResultSetCursor. A null is returned from fetchNext if the cursor is pass the last item in the result
- Disconnect from the Text Search Engine server

```
Dim dsTS as new DXDatastoreTS
Dim cur as DXResultSetCursor
Dim item as DXDDO
dsTS.connect "zebra", "7502", dx_CTYP_TCPIP
dim cmd as String
cmd = "SEARCH="
cmd = cmd & "(COND=({web site}));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"
Set cur = dsTS.execute(cmd)
Do While cur.isValid
   Set item = cur.fetchNext
   'Do something with the DDO item
Loop
cur.destroy    'in case the variable is being reused by another query
dsTS.disconnect
```

A complete sample of a text search query is provided in the samples directory.

Evaluate a Text Search query from the datastore

The steps are:

- Create a DXDatastoreTS
- Connect to the Text Search Engine server TM
- Specify query to find all text documents with words that start with the letters `UN`
- Evaluate the query and loop through the results using DXResults
- Disconnect from the Text Search Engine server

```
Dim dsTs as new DXDatastoreTS
Dim item as DXDDO
Dim results as DXResults
Dim iter as DXSequentialIterator
dsTS.connect "TM", "", "", ""
Dim cmd as String
cmd = "SEARCH="
cmd = cmd & "(COND=($MC=*$ UN*));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"
Set results = dsTS.evaluate(cmd)
Set iter = results.createIterator
Do While iter.more
   Set item = iter.next
   'Do something with the DDO item
Loop
dsTS.disconnect
```

Getting match highlighting information for each text query result item

This code sample retrieves match highlighting information for each text query result item during a text query. The MATCH_INFO option is set to YES, indicating that the match highlighting information will be returned. The MATCH_DICT option specifies whether the highlighting information will be obtained using a dictionary. Because the MATCH_DICT option is set to NO, the dictionary will not be used to retrieve highlighting information. The match information is returned in the attribute DKMATCHESINFO in the DKDDO returned from a text query. The value of the attribute DKMATCHESINFO will be a DKMatchesInfoTS object.

Important: This process is time consuming because the document is retrieved from DL datastore and analyzed linguistically, and potential matches are determined. These processes will have an impact on the performance of a text query.

```
Dim dsTS As new DXDatastoreTS
Dim cur as DXResultSetCursor
Dim item as DXDDO
Dim count As Integer, i As Integer
Dim pidId As String, dataName As String
Dim data As Variant
Dim mInfo As DXMatchesInfoTS
Dim mSect As DXMatchesDocSectionTS
Dim mPara As DXMatchesParagraphTS
Dim mText As DXMatchesTextItemTS
Dim li As Integer, lj As Integer, lk As Integer
Dim numSects As Integer, numParas As Integer, numTextItems As Integer
Dim numNewLines As Integer
Dim strDataName As String
Dim iCCSID As Integer, iLang As Integer
Dim iOffset As Integer, iLen As Integer
Dim cmd As String
dsTS.Connect "TM", "", "", "LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)"
cmd = "SEARCH=(COND=(UNIX));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX;MATCH_INFO=YES;MATCH_DICT=NO"
cmd = cmd & "MAX_RESULTS=2)"
Set cur = dsTS.execute(cmd)
Do While cur.isValid
   Set item = cur.fetchNext
   pidId = item.getPid
   count = item.dataCount
   For i = 1 To count
    dataName = item.getDataName(i)
    If dataName = dx_DKMATCHESINFO Then
     Set mInfo = item.GetData(i)
     strDataName = mInfo.getDocumentName
     numSects = mInfo.numberOfSections
     For li = 1 To numSects
      Set mSect = mInfo.getSection(li)
      strDataName = mSect.getSectionName
      numParas = mSect.numberOfParagraphs
      For lj = 1 To numParas
       Set mPara = mSect.getParagraph(lj)
       iCCSID = mPara.getCCSID
       iLang = mPara.getLanguageId
       numTextItems = mPara.numberOfTextItems
       For lk = 1 To numTextItems
        Set mText = mPara.getTextItem(lk)
        strDataName = mText.GetText
        If mText.isMatch = True Then
         iOffset = mText.getOffset
         iLen = mText.getLength
        End If
       numNewLines = mText.numberOfNewLines
       Next
      Next
     Next
    Else
     data = item.GetData(i)
    End If
   Next                    ' end for i = 1 to count
Loop
cur.destroy
dsTS.disconnect
```

Getting match highlighting information for a particular text query result item

This code sample retrieves the match highlighting information for a specific item returned from a text query. The match information contains the text of the document

and the highlighting information for every match of the corresponding query. The `resultSetCursor` passed into this routine must be in an open state.

Important: This process is time consuming because the document is retrieved from DL datastore and analyzed linguistically, and potential matches are determined. These processes will have an impact on the performance of a text query.

```
Dim dsTS As new DXDatastoreTS
Dim cur as DXResultSetCursor
Dim item as DXDDO
Dim count As Integer, i As Integer
Dim pidId As String, dataName As String
Dim data As Variant
Dim mInfo As DXMatchesInfoTS
Dim mSect As DXMatchesDocSectionTS
Dim mPara As DXMatchesParagraphTS
Dim mText As DXMatchesTextItemTS
Dim li As Integer, lj As Integer, lk As Integer
Dim numSects As Integer, numParas As Integer, numTextItems As Integer
Dim numNewLines As Integer
Dim strDataName As String
Dim iCCSID As Integer, iLang As Integer
Dim iOffset As Integer, iLen As Integer
Dim docId As String, indexName As String
Dim cmd As String
dsTS.Connect "TM", "", "", "LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)"
cmd = "SEARCH=(COND=(UNIX));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX"
cmd = cmd & "MAX_RESULTS=2)"
Set cur = dsTS.execute(cmd)
Do While cur.isValid
   Set item = cur.fetchNext
   pidId = item.getPid
   docId = pidId
   count = item.dataCount
   indexName = item.getObjectType
   For i = 1 To count
    dataName = item.getDataName(i)
    data = item.GetData(i)
   Next                          ' end for i = 1 to count
    Set mInfo = ds.getMatches(cur, docId, indexName, False)
    If (mInfo Is Nothing) = False Then
     Set mInfo = item.GetData(i)
     strDataName = mInfo.getDocumentName
     numSects = mInfo.numberOfSections
     For li = 1 To numSects
      Set mSect = mInfo.getSection(li)
      strDataName = mSect.getSectionName
      numParas = mSect.numberOfParagraphs
      For lj = 1 To numParas
     Set mPara = mSect.getParagraph(lj)
     iCCSID = mPara.getCCSID
     iLang = mPara.getLanguageId
     numTextItems = mPara.numberOfTextItems
     For lk = 1 To numTextItems
      Set mText = mPara.getTextItem(lk)
      strDataName = mText.GetText
       If mText.isMatch = True Then
       iOffset = mText.getOffset
       iLen = mText.getLength
     End If
     numNewLines = mText.numberOfNewLines
    Next
   Next
  Next
```

```
 End If
Loop
cur.destroy
dsTS.disconnect
```

Result set cursor

DXResultSetCursor is a datastore cursor which manages a collection of DDO objects. The collection is a resulting set of a query submitted to the datastore.

When the result set cursor is created it is in an open state. To re-execute the query the user should close the cursor and reopen it.

Open and close the result set cursor to re-execute the query:

```
Dim cmd as String
cmd = "SEARCH=(INDEX_CLASS=GP2DLS4);"
cmd = cmd & "OPTION=(CONTENT=YES;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;"
cmd = cmd & "TYPE_FILTER=FOLDERDOC)"
Dim cur as DXResultSetCursor
...
Set cur = dsDL.execute(cmd)
cur.close
cur.open
```

Set position and get position in a result set cursor:

The result set cursor allows you to set cursor position and get the current position.

- Execute a query
- Set the position to next and fetch a DDO
- Get the current position of the result set cursor. A null is returned from fetchObject if the cursor is pass the last item in the result.

```
Dim cmd as string
cmd = "SEARCH=(INDEX_CLASS=GP2DLS4);"
cmd = cmd & "OPTION=(CONTENT=YES;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;"
cmd = cmd & "TYPE_FILTER=FOLDERDOC)"
Dim cur as DXResultSetCursor
Dim item as DXDDO
Dim i as Long
i = 0
...
Set cur = dsDL.execute(cmd)
Do while cur.isValid
   cur.setToNext
   Set item = cur.fetchObject
   If item is Nothing Then
   Else
     i = cur.getPosition
     'Do something with the DDO
   End If
Loop
cur.destroy    'in case the variable is being reused by another query
```

Another way to do this is:

```
Set cur = dsDL.execute(cmd)
Do While cur.isValid
   cur.setPosition dx_NEXT, 1 'the 1 is ignored for dx_NEXT
   Set item = cur.fetchObject
   If item is Nothing Then
```

```
   Else
     i = cur.getPosition
     'Do something with the DDO item
   End If
Loop
cur.destroy    'in case the variable is being reused by another query
```

You can use the RELATIVE positioning. In thefollowing example, every other item in the result set cursor is skipped:

```
Dim increment as Long
    increment = 2
     Set cur = dsDL.execute(cmd)
     Do While cur.isValid
       cur.setPosition dx_RELATIVE, increment
       Set item = cur.fetchObject
       If item is Nothing Then
       Else
         i = cur.getPosition
         'Do something with the DDO
       End If
Loop
cur.destroy    'in case the variable is being reused by another query
```

Creating a collection from a result set cursor:

The result set cursor allows you to supply a collection to be filled in with a certain number of items from the result set cursor. In the following example, all items from the result set cursor are fetched into the sequential collection. A zero in the first parameter of fetchNextN indicates that all items of the cursor will be put into the collection. The first parameter specifies how many items to put into the collection. If the fItems is true as least one item was returned.

```
Dim seqColl as DXSequentialCollection
Dim fItems as Boolean
fItems = False
Dim how_much as Long
how_much = 0
fItems = cur.fetchNextN(how_much, seqColl)
```

Remember to call destroy on the cursor before reusing the variable for another query; for example, in a loop where a query is executed with different criteria.

Retrieving a document or folder

This section describes the methods you should use to retrieve documents, parts, and folders.

Retrieving a document

In order to retrieve a document from DXDatastoreDL, you need to know its index-class name and item-id. You also need to associate the DDO to a datastore and establish a connection:

```
Dim dsDL as new DXDatastoreDL
dsDL.connect "LIBSRVR2", "userId1", "passwd1"
Dim ddo as new DXDDO
ddo.setObjectType "GRANDPA"      'set the index-class name it belongs to
ddo.setPid "LN#U5K6ARLGM3DB4"    ' set the item-id
ddo.setDatastore dsDL            'associate to dsDL
ddo.retrieve                     'retrieve it
```

In case the DDO is a result of a parametric query with the query option CONTENT=NO, the DDO is empty (does not have the attribute values), but all information required to retrieve it is already set. You simply need to call:

```
ddo.retrieve
```

to retrieve the DDO. After a call to retrieve, the DDO will have all of its attribute value set to the value of persistent data stored in the datastore. If the document has parts, the dx_DKPARTS attribute is set to a DXParts object. However, the content of each part in this collection is not yet retrieved. Since the size of a part may be big, it is not necessarily desirable to retrieve all of them into memory at once. It is better to retrieve each part you want explicitly. See “Retrieving parts” below for an example on how to retrieve a part.

Retrieving parts

Once you have retrieved a DDO, you can get its parts stored in dx_DKPARTS attribute,

```
Dim parts as DXParts
Set parts = ddo.getDataByName(dx_DKPARTS)
```

Note that the above code assumes that the dx_DKPARTS attribute exists. It will throw an exception if the attribute does not exist. See also “Retrieving a folder” for another example of extracting attribute value by getting the data_id first and testing it for zero.

To retrieve each part, you need to create an iterator to iterate over the collection and retrieve each part one by one. See also “Creating and using dx_DKPARTS attribute” on page 167.

```
'Create iterator and process the part collection member one by one
If parts Is Nothing Then
Else
   Dim blob as DXBlobDL
   Dim iter as DXSequentialIterator
   Set iter = parts.createIterator
   Do While iter.more
      Set blob = iter.next
      If blob Is Nothing Then
      Else
        blob.retrieve           'retrieve the blob's content
        blob.open               'other processing, as needed
      End If
   Loop
End If
```

Note that similar to DDO results of a parametric query, each part XDO inside DXParts collection is empty (does not have its content set), but it has all the information needed for its retrieval; this part is ready to be retrieved. You only need to call

```
blob.retrieve
```

to bring its content and the rest of other information into memory.

Retrieving a folder

Retrieving a folder DDO is the same as retrieving a document DDO. After retrieve, the folder DDO will have all of its attributes set, including a special attribute dx_DKFOLDER. This attribute value is set to DXFolder object, a collection of DDO

member of this folder. Like parts in DXParts, these member DDO are relatively empty, they only contains enough information to retrieve them. You can retrieve it whenever you need.

```
data_id = ddo.dataId(dx_DKFOLDER)     'get dx_DKFOLDER data-id
If data_id = 0 Then                   'folder not found
   MsgBox " folder data-item not found"
   Exit sub
End If
Dim col as DXFolder
col = ddo.getData(data_id)            'get the folder collection
'Create iterator and process the DDO collection member one by one
If col Is Nothing Then
Else
   Dim item as DXDDO
   Dim iter as DXSequentialIterator
   Set iter = col.createIterator
   Do While iter.more
      Set item = iter.next
      If item Is Nothing Then
      Else
        item.retrieve                  'retrieve the member DDO
        .... 'other processing
      End If
   Loop
End If
```

See also “Creating and using dx_DKFOLDER attribute” on page 168 for more information.

Creating, updating, and deleting documents or folders

This section describes the processes involved in creating, updating, and deleting documents and folders.

Creating a new document

To create a new document and save its persistent data in the datastore, you need to create a DDO with all its attributes and other information set, excepts its item-id. The item-id will be assigned and returned by the datastore. Some of the previous examples can be combined to provide a complete steps:

```
'step 1: create a datastore and connect to it
Dim dsDL as new DXDatastoreDL
dsDL.connect "LIBSRVR2", "userId1", "passwd1"

'step 2: create a document (or folder) DDO and set all its attributes and
'other required information - see the section on "Using DDO"
Dim cddo as new DXDDO
cddo.setObjectType "GRANDPA"    'set the index-class name it belongs to
cddo.setDatastore dsDL          'associate to dsDL

'step 2.a: add attributes according to index-class GRANDPA
Dim data_id as Integer
data_id = cddo.addData("Title") 'add a new attribute named "Title"
cddo.addDataPropertyAndValue data_id, dx_Property_Type,
    dx_VString 'add type property VSTRING
    cddo.addDataPropertyAndValue data_id,
       dx_Property_Nullable, False 'add nullable property
    data_id = cddo.addData("Subject")  'add a new attribute named "Subject"
    cddo.addDataPropertyAndValue data_id, dx_Property_Type, dx_VString
    cddo.addDataPropertyAndValue data_id, dx_Property_Nullable, True
    'Add some more attributes as necessary
```

```
    ' ...

'step 2.b: add dx_DKPARTS attribute
Dim blob as new DXBlobDL
blob.setDatastore dsDL          'create a new XDO blob
blob.setPartId 5                'set part number to 5
blob.setContentClass dx_CC_GIF  'set content class type GIF
blob.setRepType dx_DK_REP_NULL  'set rep type for the part
blob.setContentFromClientFile "choice.gif" 'set the blob's content
blob.setInstanceOpenHandler "netscape", true 'set the blob's viewer
parts.addElement blob           'add the blob to the parts collection
... 'create and add some more blobs to the collection as necessary
'Create dx_DKPARTS attribute and sets it to refer to the DXParts object
data_id = ddo.addData(dx_DKPARTS) 'add attribute dx_DKPARTS
cddo.addDataPropertyAndValue data_id, dx_Property_Type,
    dx_Collection_XDO 'add type property
    cddo.addDataPropertyAndValue data_id, dx_Property_Nullable,
       True 'add nullable property
    cddo.setData data_id, parts     'sets the attribute value

'step 2.c: sets the item type : document
cddo.addPropertyAndValue dx_Property_Item_Type, dx_Document

'step 3: make it persistent
cddo.add                      'a document is created in the datastore
```

After the last step, you will have a new document with the above information created in the datastore. When a document DDO is added to a datastore, all its attributes are being added, including all parts inside DXParts collection. The same rule applies to adding a folder DDO, the DXFolder collection members are added to the datastore as a Digital Library folder. Recall that a Digital Library folder actually contains a table-of-contents of its members, which are existing documents and folders. Therefore, all folder members must be created first in the datastore before you can add a folder DDO. In theory, you can add the same document to a different datastore of the same type, provided that you adjust the information accordingly. To add this document to the server `LIBSRVRN`, which has an index class `GRANDPA2` with the same structure as `GRANDPA`, you could do the following steps:

```
'Create datastore and connect to LIBSRVRN
Dim dsN as new DXDatastoreDL
dsN.connect "LIBSRVRN", "userIdN", "passwdN"
'Update the Pid
ddo.setObjectType "GRANDPA2"       'set the new index-class
ddo.setPid ""                      'blank the item-id
ddo.setDatastoreName "LIBSRVRN"    'set the new datastore name'
ddo.setDatastore dsN               'set the new datastore
ddo.add                            'add it
```

Updating a document or a folder

Updating a document is quite simple; you need to update to have the item-id and object-type set. After updating the desired attributes, for example, set another value for the attribute or add more parts to DXParts collection. Finally, call the update method to have the update reflected in the datastore:

```
'Update the value of attribute Title
Dim data_id as Integer
data_id = ddo.getDataByName("Title")
ddo.setData data_id, "Guess who is behind all this"
ddo.update
```

After the call to update, the value of attribute Title in the datastore will be updated. The parts in this document will not be updated unless their content changed. The

associated datastore and its connection must be valid when you call the update method. Updating a folder DDO takes similar steps. You need to update some attribute values, or add/remove some elements from DXFolder, then call the update method.

Updating parts

The collection of parts in a document is represented as a DXParts object. DXParts has two members to add and remove a part from the collection and have it reflected in the datastore immediately. The document must already exists in the datastore.

Add and remove member: Given a document DDO, you could use the following code fragments to add a part to this document immediately,

```
Dim ddo as DXDDO                'a document DDO
Dim newPart as DXBlobDL         'a new part to be added
... 'ddo and newPart are initialized somewhere along the line
Dim parts as DXParts
Set parts = ddo.getDataByName(dx_DKPARTS) 'get DXParts
parts.addMember ddo, newPart
```

The last statement gives the same effect as:

```
newPart.add              'add the part to datastore, into the specified document
parts.addElement newPart  'add newPart into this collection
```

The `newPart` must have its Pid set to make it work. Similarly, to remove `newPart` from the collection, you would do:

```
parts.removeMember ddo, newPart
```

which is equivalent to:

```
Dim iter as DXSequentialIterator
Set iter = parts.createIterator
...
Set newPart = parts.removeElementAt(iter)    'remove the part pointed by iter
newPart.del                                  'delete the part from this document
                                             'the copy in memory is unchanged
```

To compare with the same method in DXFolder, the `removeMember` method in DXParts actually deletes the persistent copy of the part in the datastore. In DXFolder, this method only removes the item from the folder table-of-contents; it does not actually delete the item. See “Updating Folder”.

Differences with update on a document DDO: Add and remove member methods on DXParts provide conveniences for adding and removing a part in the collection. Compared to the `update` method in a document DDO, add and remove member are faster too. The `update` method on a DDO will update all of the attributes in the DDO including DXParts and all of its member which changed. So, the steps would be

```
...
Dim parts as DXParts
Set parts = ddo.getDataByName(dx_DKPARTS) 'get DXParts, assume it exists
parts.addElement newPart
ddo.update                               'updates the whole ddo
```

Updating Folder

The collection of documents and folders in a Digital Library folder is represented as a DXFolder object. In the datastore, a folder holds a table of contents referring to its

object members, instead of keeping all actual objects. DXFolder has two members to add and remove a member (be it a document or a folder) from the collection and have it reflected in the datastore immediately. The removed or added document or folder must already exists in the datastore.

Add and Remove Member: Given a folder DDO, you could use the following code fragments to add another document or folder DDO to it,

```
Dim folderDDO as DXDDO      'a folder DDO
Dim newMember as DXDDO      'a new DDO to be added as a member of this folder
... 'folderDDO and newMember are initialized somewhere along the line
Dim folder as DXFolder
'get DXFolder, assume it exists
Set folder = folderDDO.getDataByName(dx_DKFOLDER)
folder.addMember folderDDO, newMember
```

The `newMember` and `folderDDO` must exists in the datastore to make it work. The last statement gives the same effect as:

```
folder.addElement newMember   'add newMember to this collection
folderDDO.update              'reflects it in the datastore
```

Similarly, to remove `newMember` from the collection, you would do:

```
folder.removeMember folderDDO, newMember
```

which is equivalent to a longer steps:

```
Dim iter as DXSequentialIterator
Set iter = folder.createIterator
...
'remove newMember from this collection
Set newMember = folder.removeElementAt(iter)
folderDDO.update  'reflects it in the datastore
...
```

Note that removing a member from the folder only removes that member from the folder table of contents. It does not delete the member itself from memory or datastore.

Differences with update on a folder DDO: Add and remove member methods on DXFolder provide conveniences for adding and removing a document or folder in the collection. Compared to the `update` method in a folder DDO, add and remove member are faster too. The `update` method on a DDO will update all of the attributes in the DDO including DXFolder and all of its members (i.e. the folder table of contents), whereas the add and remove member operation only involves adding or removing one particular member to or from the folder table of contents.

Deleting a document or a folder

You only need to call the `del` method in the DDO to delete the persistent data from the datastore:

```
ddo.del
```

The DDO must have its item-id and object-type set, and a valid connection to a datastore. The above statement can be used to delete a folder too. Only the persistent data is deleted, the in-memory copy of the DDO does not change. Therefore, you can add this DDO back to the same or different datastore later. See “Creating a new document” on page 185 for more information.

Advanced topics

This section describes different types of queries that you can perform.

Queryable collection

A queryable collection is a collection that can be queried further, providing a smaller set of more refined results. One example of a queryable collection is the DXResults class. DKResults is a collection of DDO, the results of query evaluation.

Getting the result of a query

The following example illustrates how to submit a parametric query and get results:

```
'Establish a connection
Dim dsDL as new DXDatastoreDL
dsDL.connect "LIBSRVR2", "userId1", "passwd1"
'Create a query object
Dim query1 as String
query1 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> NULL));"
Dim pq as DXParametricQuery
Set pq = dsDL.createQuery(query1)
pq.execute
Dim rs as DXResults
Set rs = pq.result
```

The results are in the `rs` object, which is an instance of DXResults. You can use previous code examples to process the collection and get the DDO out. See “Collections and iterators” on page 162 for more information.

Evaluating a new query

Given a result from the above query, you can submit another query to this result as to refine the current result to get a smaller set. Its concept is similar to using the current result as a scope for the new query. Here are the steps to do it:

```
Dim query2 as String
query2 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Subject == 'Mystery'));"
Dim rs2 as DXResults
Set rs2 = rs.evaluate(query2, dx_Parametric_QL_type
```

The totals of both queries would be equivalent to:

```
"SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> NULL AND Subject == 'Mystery'));"
```

You can repeat this step over and over again to get satisfactory results. Once you started with a parametric query, the subsequent queries must be of the same type, otherwise it will not make any sense, and probably will return a null result.

The same procedure will work for text queries also. The following is an example for text queries:

```
Dim dsTS as new DXDatastoreTS
dsTS.connectPort "barney", "6011", dx_CTYP_TCPIP
Dim tquery1 as String
tquery1 = "SEARCH=(COND=(IBM));OPTION=(SEARCH_INDEX=TMINDEX)"
Dim tq as DXTextQuery
Set tq = dsTS.createQuery(tquery1)
tq.execute
Dim trs as DXResults
Set trs = tq.result
Dim tquery2 as String
```

```
tquery2 = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)"
Dim trs2 as DXResults
Set trs2 = trs.evaluate(tquery2, dx_Text_QL_type)
```

The sum of both queries is equivalent to:

```
"SEARCH=(COND=(IBM AND Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
```

Using queryable collection instead of combined query

Evaluating a queryable collection bears some similarities with some other features in this class library. One of such a feature is Combined Query. The combined query provides a flexibility to submit a combination of parametric and text queries, with or without scopes. However, all of these queries must be submitted at once, not in piece-meal like evaluating a queryable collection. See “Combined query”.

The result of a combined query is a DXResults, so theoretically you can evaluate another parametric query against it; although in the current implementation, it may not always work.

Programming tips

Evaluating queryable collection with subsequent queries provide a flexibility to refine the results of a previous query, step by step, until you get a satisfactory final result. This is quite useful for browsing the content of datastore dynamically, formulating the next query depending on the previous results. However, if you know the total query in advance, it would be more efficient to submit the total query once instead of in pieces.

Combined query

As suggested by its name, a combined query allows you to execute a combination of parametric and text queries, with or without a scope. The final results is intersections between the scopes and the results of each query. Therefore, if you are not careful in formulating the query and including scopes, these collections may not intersect at all, and you would end up with an empty result. If there is at least one parametric and one text query, the resulting DDO returned in DXResults will have an attribute dx_DKRANK, which signifies the highest rank of the matching part belonging to this document.

Important: For each query in a combined query, you must use a dedicated connection to the search engine; you cannot route multiple queries through the same connection.

Combined parametric and text query

To execute a combined query with one parametric, one text query, without any scope, you need to create the combined query object, the parametric and text queries, and pass the latter two queries as input parameters to be executed by the combined query. Here is an example:

```
Dim dsDL as new DXDatastoreDL
dsDL.connect "LIBSRVR2", "userId1", "passwd1"
Dim dsTS as new DXDatastoreTS
dsTS.connectPort "TM", "", dx_CType_TCPIP 'TM is a local alias for the TM server
'Create a parametric query
Dim pquery as String
pquery = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(DLSEARCH_Date > 1994);"
Dim pq as DXParametricQuery
```

```
Set pq = dsDL.createQuery(pquery)
'Create a text query
Dim tquery as String
tquery = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)"
Dim tq as DXTextQuery
Set tq = dsTS.createQuery(tquery)
'Create a combined query
Dim cq as new DXCombinedQuery
'Package the queries in DXNVPair as input parameters
Dim par(1 To 3) As New DXNVPair
par(1).Set dx_PARM_QUERY, pq
par(2).Set dx_TEXT_QUERY, tq
par(3).setName dx_PARM_END     'to signal the end of the parameter list
'Execute the combined query
cq.execute par
'Get the results
Dim res as DXResults
Set res = cq.result
If res Is Nothing Then
Else
   'Process the results
   ...
End If
```

The last if statement is necessary to make sure the DXResults object is not null before calling its method. See examples under “Collections and iterators” on page 162 and “Querying the datastore” on page 175 for more information.

Using a scope

If you happen to have a DXResults object you want to use as the scope, you can modify the above code slightly to insert the scope as an additional parameter:

```
Dim scope as DXResults
'assume that this is the scope
'initialized somewhere as a result
'of some parametric query
...
'Package the query in DXNVPair as input parameters
Dim par(1 to 4) as new DXNVPair
par(1).set dx_PARM_QUERY, pq
par(2).set dx_TEXT_QUERY, tq
par(3).set dx_SCOPE_DL, scope
par(4).setName dx_PARM_END
'Execute the combined query
cq.execute par
...
```

The results of a combined query can also be used as a scope of another combined query.

A complete sample of a combined query is provided in the samples directory.

Ranking

If the combined query contains at least one text query, then the DDO in the result will have the attribute dx_DKRANK. This is a transient attribute, it will not be stored in the datastore but will be computed each time by the text search engine. The value of the rank corresponds to the highest rank of the part in the document which satisfy the text query conditions.

Programming tips

The same observation as in queryable collection also applies here. If you have several parametric queries and scopes, it is more efficient to execute the total query as a single parametric query. This is also true for text queries. The most efficient execution would be if you have one total parametric query and one total text query, without scope. See also “Queryable collection” on page 189, for further information.

The query option MAX_RESULTS=nn limits the number of results returned to the caller. Usually, this option is more applicable to text queries, since the result is sorted by rank in descending order. Thus, if this option is set to 10, it means that the caller only wants the ten highest matching rank of the results. The meaning of this option is different for parametric queries. Since there is no notion of rank, the caller will get the first 10 of the results. This results will be intersected with the result from text query. Therefore, when combining a parametric query and text query, it is advisable not to specify this option for the parametric query.

Text Search Engine

Text Search Engine is the product name of the text search engine. Text Search Engine allows you to specify boolean, proximity, GTR, hybrid and free text queries. The query results returned from Text Search Engine contain the Digital Library item ID, part number and ranking information. This information can be used to create an DL XDO that can be used to retrieve the text document contents in Digital Library.

Programming Tip

To prevent time out of a Text Search Engine connection it is best to perform a connect, perform a query and disconnect as oppose to leaving a connect open for long period of time.

The search options CCSID and LANG work together; if one is specified, you should also specify the other. The default CCSID and LANG are specified by the datastore text search options (`DK_OPT_TS_CCSID` and `DK_OPT_TS_LANG`).

You can specify more than one search option for a query term. The search options are separated by commas. An example of multiple search terms is specified in “GTR query” on page 193.

If you wish to specify both the SC and the MC search option, the SC option must occur first. An example could be ($SC=?,MC=*$ U?I*).

For detailed descriptions of the SC (single required character) and the MC (a sequence of optional characters) search options, refer to the Application Programming Reference modules.

Boolean query

A boolean query is made up of words and phrases, separated by a boolean operator. A phrase is a sequence of words enclosed in single quotes, to be searched as a literal. In the example below we are searching for all text documents with the word `WWW` or the phrase `Web site` in the `TMINDEX` text search index:

```
Dim cmd as String
cmd = "SEARCH=(COND=(www OR 'web site'));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"
```

Free text query

A free text query is made up of words, phrases or sentences. The words do not need to be adjacent to each other. In the example below we are searching for all text documents with the free text `Web site` in the `TMINDEX` text search index:

```
Dim cmd as String
cmd = "SEARCH=(COND=({web site}));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"
```

Hybrid query

A hybrid query is made up of a boolean query, followed by a free text query. In the example below we are searching for all text documents with the words `IBM` and `UNIX`, as well as the free text `Web site` in the `TMINDEX` text search index:

```
Dim cmd as String
cmd = "SEARCH=(COND=(IBM AND UNIX {web site}));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"
```

Proximity query

A proximity query relates to a sequence of search arguments found in the same document, paragraph or sentence. In the example below we are searching for all text documents with the phrase `rational numbers` and the word `math` in the same paragraph.

```
Dim cmd as String
cmd = "SEARCH=(COND=($PARA$ {'rational numbers' math}));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"
```

GTR query

A GTR (Global Text Retrieval) query is optimized for double-byte character set (DBCS) languages like Japanese or Chinese but also supports single-byte character set (SBCS) languages.

Make sure that the text search datastore options `DK_OPT_TS_CCSID` (coded character set identifiers) and `DK_OPT_TS_LANG` (language identifiers) are set properly. The default for `DK_OPT_TS_CCSID` is `DK_CCSID_00850`. The default for `DK_OPT_TS_LANG` is `DK_LANG_ENU`. These values are used as the global defaults for the text query. You can also enter specific `CCSID` and `LANG` information, as shown in the following example; see “Programming Tip” on page 192 for further information.

The following example shows a search for a GTR index for all text documents with the phrase `IBM marketing`. All double-byte characters should be enclosed in single quotes. The phrase to be searched for should be in the specified character code set and language. The match keyword is set to indicate the degree of similarity for the phrase.

```
Dim cmd as String
cmd = "SEARCH=(COND=($CCSID=850,LANG=6011,MATCH=1$ "
cmd = cmd & "'IBM marketing'));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMGTRX)"
```

Load data that is indexed by Text Search Engine

In order to load data into Digital Library that is indexed by Text Search Engine, a Digital Library index class must be created as well as a Text Search Engine index. Before creating a Text Search Engine index the Text Search Engine server must be running. To make sure your environment is set up properly, run the Digital Library

samples TListCatalogDL and TListCatalogTS after they have been updated with your server, user ID etcetera. To create parts in Digital Library that are index by TM, refer to “Using XDO” on page 169. Once the data is loaded into Digital Library, the documents must be placed on the document queue. This is done by a method in the DXDatastoreDL called wakeUpService which takes a search engine name (″SM″). Once this is done, you go into the Digital Library TM administration panel, double click on the TM server, double click on the TM index and single click the index button on the explicit notebook page. (For more information on Text Search Engine administration, refer to the *Text Search Engine: Application Programming Reference*). This will index the documents in the document query. Once the indexing is complete you can perform queries against Text Search Engine.

Handling large objects

This section describes the ways in which Digital Library handles large objects.

What is MAXPIECE?

`MAXPIECE` is an environment variable that indicates the maximum object size processed in Digital Library. This variable defines the largest object in megabytes that is processed as a whole.

When `MAXPIECE` is set during the input and output operation to the Digital Library datastore, an object that is larger than the `MAXPIECE` value will be broken down into chunks, sized equal to or less than the `MAXPIECE` value. If `MAXPIECE` is not set, the large object will be treated as a single entity.

Using the MAXPIECE environment variable

In Windows NT, go to **Start ⟶ Setting ⟶ Control Panel ⟶ System ⟶ Environment ⟶ System Variables** or **User Variables**, enter `MAXPIECE` in the entry field and a number (for example, `4`) in the value field, then click **Set ⟶ OK**. This will set `MAXPIECE` to `4` megabytes.

If you want to store some large objects using the same part ID but different representation type (repType) for each object, make sure the first four characters of the repType do not contain blanks, semicolons, or the string `FRN$`.

Client/Server mode

The Digital Library ActiveX APIs can be configured to run in a client/server mode. As it relies on DCOM as the underlying mechanism for accessing remote objects on the network, it requires Windows NT 4.0 Service Pack 2 (or above) which provides support for launching DLL-based objects remotely via a surrogate.

For the first release, all the server components are installed by default; to enable the ActiveX classes to operate in the client/server mode, a number of registry settings will have to be altered. Several tools can be used for this purpose: regedit, DCOMCnfg, or OLEView. The first two comes with the operating system, the last one is supplied with Visual C++ and the Win32 SDK or it can be downloaded for free from the Microsoft Web site. The first step is to ensure that the key HKEY_LOCAL_MACHINE\Software\Microsoft\OLE has a value of Y. The value of this key globally affects whether any remote clients can launch or connect to already-running objects. The registry entries relevant to remote objects are found under the AppID key in HKEY_CLASSES_ROOT. For server, the AppID entry for Digital Library should have a value named `DllSurrogate` added with an empty string

value, and a value named `RunAs` which can be either a local or domain account, or it can be the string `Interactive User`, to specify that the account of the logged-on user should be used. If this value is missing, the COM Service Control Manager will use `Launching User`, that is, the user that has requested the object (of course, this only works if the client allows this with the requested impersonation level). For client, there should be a value named `RemoteServerName` added with a string value which is the name of the remote machine on which the server resides. Also, to function as a client, the registry key HKEY_CLASSES_ROOT\CLSID\{clsid}\InProcServer32 should be removed.

The easiest way to accomplish all these registry changes is to use **OLEView**. Select **Expert mode** from the **View** menu, and expand the node on **All Objects**. All the automation classes will appear as Digital Library classes. After you have selected a class, the implementation tab on the right side can be used to choose the **use surrogate process** option (equivalent to removing the HKEY_CLASSES_ROOT\CLSID\{clsid}\InProcServer32 key). The **Activation** tab can be used to specify the remote machine name (equivalent to the RemoteServerName registry entry) and activation options (Interactive user or Launching user).

To access an object on the server machine, the launch and activate permissions on the server machine must be set to allow access by a client account. Again, this can be done using DCOMCnfg or OLEView. Because of a bug in Windows NT (see Q171359 in the Microsoft Knowledge Base), the authentication level should be set to something other than none until the bug is fixed; the impersonation level can be set to either identify or impersonate (the default values being connect and identify).

Understanding the workflow and workbasket functions

This section describes the workflow and workbasket functions.

Understanding the workflow service

The ActiveX APIs provide the workflow service for the Digital Library datastore, encompassing the workflow and workbasket functions that are available in the Folder Manager. A workbasket is a container that holds documents and folders that you want to process. A workflow is an ordered set of workbaskets that represent a specific business process. Folders and documents move between workbaskets within a workflow, allowing your applications to create simple business models and to route work through the process until completion.

The workflow model used in the Folder Manager follows these rules:

- A workbasket does not need to be located in a workflow
- A workbasket can be located in one or more workflows
- A workbasket can be located in the same workflow more than once
- A document or folder can only be stored in one workflow at a time; however, workbaskets can be located in multiple workflows
- A document or folder can be stored in a workbasket that is not located in a workflow

The DXWorkFlowServiceDL class represents the workflow service of the Digital Library datastore. This class provides the capability to start, change, remove, route, and complete a document or folder in a workflow. Additionally, the DXWorkFlowServiceDL class allows you to retrieve information about workbaskets

and workflows in a Digital Library datastore. In Digital Library, the DXWorkFlowDL and DXWorkBasket classes are the object-oriented representations of a workflow item and a workbasket item, respectively.

Establishing a connection

You must establish a connection to a Digital Library datstore before you can use the workflow service. The Digital Library datastore provides connect and disconnect methods. The following connection example would connect to a Digital Library datastore named LSMUFASA, using the user ID FRNADMIN and the password PASSWORD. The complete sample of datastore connection, wfs.frm (written in Visual Basic), is available in the samples directory.

```
Dim dsDL As New DXDatastoreDL
Dim wfDL As New DXWorkFlowServiceDL
wfDL.init (dsDL)

dsDL.connect "LIBSRVRX", "FRNADMIN", "PASSWORD"
...                                  ' do some work
dsDL.disconnect
```

The connect method allows an application to connect to a Digital Library datastore. After a workflow service is created, subsequent methods of the workflow service may be issued.

List workflows

The workflow service provides a method that lists the workflows in the system. The following example demonstrates the retrieval of the list of workflows; this list is returned in a sequential collection of DXWorkFlowDL objects. The complete sample of this function, wfs.frm (written in Visual Basic), is available in the samples directory.

```
Dim dsDL As New DXDatastoreDL
Dim dsDL As New DXWorkFlowServiceDL
wfDL.init dsDL
dsDL.connect "LIBSRVRX", "FRNADMIN", "PASSWORD"

Dim wfList1 As New DXSequentialCollection
Set wfList1 = wfDL.listWorkFlows
Dim pIter1 As New DXSequentialIterator
Set pIter1 = wfList1.createIterator

Dim pwf1 As DXWorkFlowDL
Do While pIter1.more
   Set pwf1 = pIter1.Next
   pwf1->retrieve();
   ...                       ' do some work
Loop
dsDL.disconnect
```

List workbaskets

The workflow service provides a method that lists the workbaskets in the system. The following example demonstrates the retrieval of the list of workbaskets; this list is returned in a sequential collection of DXWorkBasketDL objects. The complete sample of this function, wfs.frm (written in Visual Basic), is available in the samples directory.

```
Dim dsDL As New DXDatastoreDL
Dim dsDL As New DXWorkFlowServiceDL
wfDL.init dsDL
```

```
dsDL.connect "LIBSRVRX", "FRNADMIN", "PASSWORD"

Dim wbList1 As New DXSequentialCollection
Set wbList1 = wfDL.listWorkFlows
Dim pIter1 As New DXSequentialIterator
Set pIter1 = wbList1.createIterator

Dim pwb1 As DXWorkBasketDL
Do While pIter1.more
   Set pwb1 = pIter1.Next
   pwb1->retrieve();
   ...                        ' do some work
Loop
dsDL.disconnect
```

List items in a workflow

The workflow service provides a method that lists the item IDs of the items in a workflow. The following example demonstrates the retrieval of the list of item IDs for the items in a particular workflow; this list is returned in a sequential collection of DXString objects. The complete sample of this function, wfs.frm (written in Visual Basic), is available in the samples directory. The workflow name HI7MOPALUPFQ1U47 is fictitious; you must use a valid workflow ID instead of the workflow name in the example.

```
Dim dsDL As New DXDatastoreDL
Dim dsDL As New DXWorkFlowServiceDL
wfDL.init dsDL
dsDL.connect "LIBSRVRX", "FRNADMIN", "PASSWORD"
Dim itemIDWF As String
itemIDWF = "H17MOPALUPFQ1U47"
Dim wf As New DXWorkFlowDL
wf.init wfDL
wf.setID itemIDWF
wf.retrieve

Dim pColDoc1 As New DXSequentialCollection
Set pColDoc1 = wf.listItemIDs
Dim pIter1 As New DXSequentialIterator
Set pIter1 = wbList1.createIterator

Dim DocID As String
Do While pIter1.more
   DocID = pIter1.Next
   ...                        ' do some work
Loop
dsDL.disconnect
```

Executing a workflow

The workflow service provides methods that allow you to execute a workflow. The following example demonstrates how to start an item in a workflow, how to route an item to a workbasket, and how to complete an item in a workflow. The complete sample of this function, wfs.frm (written in Visual Basic), is available in the samples directory. You must modify the sample program to reflect these changes:

- Use a valid item ID instead of EP8L8OR9MHH##QES
- Use a valid workflow ID instead of HI7MOPALUPFQ1U47
- Use a valid workbasket ID instead of E3PP1UZOZUFQ1U3M

```
Dim dsDL As New DXDatastoreDL
Dim dsDL As New DXWorkFlowServiceDL
wfDL.init dsDL
dsDL.connect "LIBSRVRX", "FRNADMIN", "PASSWORD"
```

```
Dim itemID As String
Dim itemIDWF As String
Dim itemIDWB As String
itemID   = "EP8L8OR9MHH##QES"
itemIDWF = "H17MOPALUPFQ1U47"
itemIDWB = "E3PP1UZOZUFQ1U3M"
wfDL.startWorkFlowItemInFirstWorkBasket itemID, itemIDWF, TRUE,
                                        DX_WIP_DEFAULT_PRIORITY
...                                  ' do some work
wfDL.routeWipItem itemID, itemIDWB, TRUE, DX_NO_PRIORITY_CHANGE
...                                  ' do some work
wfDL.completeWorkFlowItem itemID
dsDL.disconnect
```

Creating a workflow

The workflow service allows you to create a new workflow. The typical workflow creation process follows these steps:

1. Create an instance of DXWorkFlowDL
2. Set the workflow name to `GOLF`
3. Set the workbasket sequence to NULL to indicate that this workflow contains no workbaskets
4. Set the privilege to **All Privileges**
5. Set the disposition to DX_WF_SAVE_HISTORY
6. Call the add method

The following example uses these steps to create a workflow. The complete sample of this function, wfs.frm (written in Visual Basic), is available in the samples directory. If you connect to the datastore as a normal user (DX_SS_NORMAL), you will not get the workflow that is defined after you connect. Therefore, this sample uses DX_SS_CONFIG.

```
Dim dsDL As New DXDatastoreDL
Dim wfDL As New DXWorkFlowServiceDL
wfDL.init dsDL
dsDL.setOption DX_OPT_DL_ACCESS, DX_SS_CONFIG
dsDL.connect "LIBSRVRX", "FRNADMIN", "PASSWORD"
Dim newwf As New DXWorkFlowDL
newwf.init wfDL
newwf.setName "GOLF"
newwf.setAccessList "All Privileges"
newwf.setHistoryDisposition DX_WF_SAVE_HISTORY
newwf.add
...                                  ' do some work
dsDL.disconnect
```

Creating a workbasket

The workflow service allows you to create a new workbasket. The typical workbasket creation process follows these steps:

1. Create an instance of DXWorkBasketDL
2. Set the workbasket name to `Hot Items`
3. Set the privilege to **All Privileges**
4. Call the add method

The following example uses these steps to create a workflow. The complete sample of this function, wfs.frm (written in Visual Basic), is available in the samples

directory. If you connect to the datastore as a normal user (DX_SS_NORMAL), you will not get the workflow that is defined after you connect. Therefore, this sample uses DX_SS_CONFIG.

```
Dim dsDL As New DXDatastoreDL
Dim wfDL As New DXWorkFlowServiceDL
wfDL.init dsDL
dsDL.setOption DX_OPT_DL_ACCESS, DX_SS_CONFIG
dsDL.connect "LIBSRVRX", "FRNADMIN", "PASSWORD"
Dim newwb As New DXWorkBasketDL
newwb.init wfDL
newwb.setName "Hot Items"
newwb.setAccessList "All Privileges"
newwb.add
...                                 ' do some work
dsDL.disconnect
```

Chapter 7. Using the Dynamic Page Builder

The Digital Library Version 2 Dynamic Page Builder provides the full power of HTML and the ability to access Digital Library from anywhere on the Internet.

This chapter describes how to use the Dynamic Page Builder to create a Web application to make Digital Library data available over the Internet or an intranet.

The Dynamic Page Builder (DPB) is a set of application programming interfaces for building Digital Library World Wide Web applications. The Dynamic Page Builder allows applications to retrieve Digital Library text and image content, and create a dynamic document for display in a Web browser.

The Dynamic Page Builder supports the IVS file format used by the IBM Video Stream Server. After metadata is retrieved, the video player will start streaming the video file.

The Dynamic Page Builder requires IBM's Net.Data as a front-end parsing and formatting tool. Net.Data combines the functions of Common Gateway Interface (CGI), Connection Manager, and Web page generation. Net.Data offers a wizard to help you access Net.Data function from your application. See the Net.Data documentation for more information about using Net.Data. From the Windows NT desktop, go to **Start → Programs → Net.data → Net.data Documentation**, to launch your Web browser to view this material.

When Net.Data encounters the `%FUNCTION(DTW_DLDPB)` statement in its macro file, information is passed to the Dynamic Page Builder cliette back-end program through the Connection Manager. The Dynamic Page Builder cliette processes the request information and returns the result to Net.Data for formatting. It uses the Digital Library C++ class library as the programming tool for the back-end cliette driver.

Configuring the Dynamic Page Builder with Net.Data

Net.Data uses an initialization file (db2www.ini) to customize settings and search paths. The location of the file, as well as its contents and syntax, depends on the platform and Web server you use.

ICS for AIX:	/usr/lpp/internet/server_root/pub
NCSA for AIX:	/usr/httpd/htdocs
IIS for WindowNT:	\inetpub\wwwroot
ICS for WindowNT:	\www\html

The Dynamic Page Builder uses three special path statements and one environment statement:

- ENVIRONMENT (DTW_DLDPB)
- MACRO_PATH
- INCLUDE_PATH
- HTML_PATH

The ENVIRONMENT statement (DTW_DLDPB)

The following statement is for the Dynamic Page Builder on Net.Data; it provides the server setting for the cliette program and macro files:

```
ENVIRONMENT(DTW_DLDPB) dtwdldpb (IN DATABASE, LOGIN, PASSWORD,
        OUT RETURN_CODE) CLIETTE "DTW_DLDPB:$(DATABASE)"
```

- `dtwdldpb` is a shared DLL that provides the non-cliette access to Digital Library. Although it is not currently supported by Dynamic Page Builder, it is required.
- `IN DATABASE,LOGIN,PASSWORD,OUT RETURN_CODE` is the variable setting for macro files, and it provides the capability to allow different users to logon to the cliette process. If the single user mode has been used, `IN DATABASE,LOGIN,PASSWORD` should be removed from db2www.ini and the macro files to improve performance.

The MACRO_PATH statement

The MACRO_PATH identifies one or more directories to search for macro files. The syntax of the MACRO_PATH is:

```
MACRO_PATH [=] path[;path;..]
```

The INCLUDE_PATH statement

The INCLUDE_PATH identifies one or more directories in which to search for a file specified on the %INCLUDE statement in a macro file:

```
Net.Data db2www.ini file
INCLUDE_PATH=c:\inetpub\wwwroot
Macro file
%INCLUDE "dlheader.html"
```

The HTML_PATH statement

The syntax of the HTML_PATH statement is similar to the MACRO_PATH statement:

```
HTML_PATH [=] path[;path;..]
```

This is the special path variable; all the large objects returned from Digital Library will be saved to the `tmplobs` directory, the subdirectory under the first directory name specified in the HTML_PATH statement.

Dynamic Page Builder functions

This section describes the Dynamic Page Builder APIs, the parameters passed to them, the parameters passed through in-line data, and the variables to be defined.

API functions

The following API functions are reserved terms in the Dynamic Page Builder. They are described in detail in the *Application Programming Reference*.

DP_DLConnect	Connect to Digital Library
DP_DLListServer	List all of the DL servers
DP_TMConnect	Connect to the text search engine
DP_DisConnect	Disconnect from search engine

DP_IndexClass	Get the index classes
DP_IndexAttribute	Get the index attributes
DP_PMQuery	Perform a parametric search
DP_PMQuerySQL	Provides parametric search function using DB2 access; the parameters are the same as DP_PMQuery()
DP_PMQuerySQL2	Provides parametric search function using DB2 access; the parameters are the same as DP_PMQuery() with one exception: DP_ATTRIBUTE_NAME
DP_TMQuery	Perform a text search
DP_TMQuerySQL	Provides parametric search function using DB2 access; the parameters are the same as DP_TMQuery()
DP_QBQuery	Perform an image search
DP_CBQuery	Perform a combined search
DP_CBQuerySQL	Provides parametric search function using DB2 access; the parameters are the same as DP_CBQuery()
DP_Folders	Get a folder collection
DP_Parts	Get a parts collection
DP_MetaData	Get the metadata
DP_Retrieve	Get a data object
DP_WorkflowService	Provides basic access to the workflow services class
DP_ListWorkFlows	Lists all the workflows in the system
DP_ListWorkBaskets	Lists all the workbaskets in the system or within the specified workflow
DP_ListItems	Lists all the items in the specified workflow or workbasket
DP_WorkFlowDL	Provides basic access to the DKWorkFlowDL class
DP_WorkBasketDL	Provides basic access to the DKWorkBasketDL class
DP_Object	Perform object execution
DP_SQLTrigger	Perform SQL execution through the input parameter DP_SQLSTATEMENT

Input parameters

TableTable 2 on page 204 shows the parameters passed into functions when API functions are called. Only one input parameter is allowed per function:

Table 2. Input parameters

Parameter:	Applied at:	Description:
CONNECT_STRING	DP_DLConnect	Connect string to the Digital Library datastore
CONNECT_STRING	DP_TMConnect	Connect string to the TM datastore
DP_DATASTORE	DP_DisConnect	Datastore identifier (″DL″, ″TM″)
PMQUERY_STRING	DP_PMQuery	Parametric search query string
PMQUERY_STRING	DP_CBQuery	Parametric search query string
TMQUERY_STRING	DP_TMQuery	Text search query string
TMQUERY_STRING	DP_CBQuery	Text search query string
INDEX_CLASS	DP_IndexAttribute	Digital Library index cass name
DP_THUMBID	DP_Retrieve	Part ID of item retrieved
DP_DESCRIBEID	DP_Parts	Part ID of TOC Part
PID	DP_Folders	Part Item ID
PID	DP_Parts	Part Item ID
PID	DP_MetaData	Part Item ID
PID	DP_Retrieve	Part Item ID
DP_DESCRIBEID	DP_Retrieve	Part ID of TOC Part; filename for tumbnail image

An example of a function call with input parameters is:

```
%FUNCTION(DTW_DLDPB)  DP_DisConnect(DP_DATASTORE)
                  {  DP_DisConnect;   %}
```

Inline data

Dynamic Page Builder APIs also allow parameters to be passed through inline data:

- For `DP_DLConnect()` and `DP_TMConnect()`:
  - CONNECT_STRING: Connect string to the Digital Library datastore
- For `DP_PMQuery()` and `DP_CBQuery()`:
  - DP_THUMBID: Part ID of item retrieved
  - DP_DESCRIBEID: Part ID of TOC part; file name for thumbnail image
  - PMQUERY_STRING: Parametric search query string (if PMQUERY_STRING is supplied, the rest of the parameters will be ignored)
  - INDEX_CLASS: Digital Library index class name
  - PM_MAX_RESULTS: Maximum results of parametric search return
  - PM_CONDITION: Query expression
  - TYPE_FILTER: Filter (″DOC″, ″FOLDER″, ″FOLDERDOC″)
- For `DP_TMQuery()` and `DP_CBQuery()`:
  - TMQUERY: Text search string
  - TM_CONDITION: Query expression
  - TM_MAX_RESULTS: Maximum results of text search return

  - SEARCH_INDEX: TM index class
  - DP_TIMELIMIT: Time limitation for the text search
- For `DP_MetaData()`:
  - PID: Parts item ID
- For `DP_Retrieve()`:
  - PID: parts item ID
  - DP_THUMBID: Part ID of item retrieved
  - DP_DESCRIBEID: Part ID of TOC part; file name for Thumbnail image
- For `DP_Parts()`:
  - PID: Parts item ID
  - DP_DESCRIBEID: Part ID of TOC part; file name for Thumbnail image
- For `DP_Folders()`:
  - PID: Parts Item ID
  - DP_THUMBID: Part ID of item retrieved
  - DP_DESCRIBEID: Part ID of TOC Part; file name for Thumbnail image

Variable definition

You need to define these variables in the db2www.ini file to use them in the macro files:

```
ENVIRONMENT(DTW_DLDPB) dtwdldpb (IN DATABASE, LOGIN, PASSWORD,
START_ROW_NUM, OUT RETURN_CODE) CLIETTE "DTW_DLDPB:$(DATABASE)"
```

- DATABASE: Digital Library server name
- LOGIN: Digital Library logon user ID
- PASSWORD: Digital Library logon user password
- START_ROW_NUM: Starting row number for display in report section
- RETURN_CODE: Return code from DP API

Special output variable

To be able to display the total Number of Rows Affected by `DP_PMQuery()` and `DP_CBQuery` , DP_API assigns the NRA to the second title column of the first row returned.

This is an example of a report section of DP_PMQuery():

```
%FUNCTION(DTW_DLDPB) DP_PMQuery(in...)
        ...
%REPORT {
    Total Number of Hits $(N2)
    %ROW{
    ...
    %}
%}
%}
```

Developing a Net.Data macro for the Dynamic Page Builder

In this section, two sample macro files are shown to illustrate the elements of the Dynamic Page Builder macro file.

Sample macro I

The following sample macro retrieves all the index class information from the Digital Library server and displays it as a selection box, in which you can pick the index class and retrieve its index attributes. The attribute results are then displayed in a second HTML page.

```
%(--------------------  Define Section  --------------------%}
%DEFINE{
        DLTable  = %TABLE(100)
        PROGRAM  = "/cgi-bin/db2www.exe/frndp2.d2w"
%}
%(--------------  FUNCTION  Definition Section  ------------%}

%FUNCTION(DTW_DLDPB) DP_IndexClass (OUT table) {
        DP_IndexClass;

        %REPORT{ %row{  <OPTION value=$(V1)>$(V1)  %}  %}

        %MESSAGE {
                -160:  "Error: 160: Catalog lookup failure"  : exit
        %}
%}

%FUNCTION(DTW_DLDPB) DP_IndexAttribute (in idx, OUT table) {
        DP_IndexAttribute;
        INDEX_CLASS = $(Index_Class);

        %REPORT  {
                %row {
                        <TR>
                        <TD>$(ROW_NUM)</TD>
                        <TD>$(V1)</TD>
                        </TR>
                %}
        %}
        %MESSAGE {
                -160:  "Error: 162: Catalog lookup failure"  : exit
        %}
%}

%(------------------ HTML Input Section --------------------%}
%HTML_INPUT{
<HTML>
        <FORM METHOD=POST ACTION="$(PROGRAM)/report" >
        <center>
        <h2> Result of Index Class in Digital Library server </h2>
        <p> Select the categories for the attributes associated.<br></p>

        <TABLE border=1 width=50%>
        <TR>
        <TD align=center> <SELECT NAME="Index_Class" SIZE=8 WIDTH=80 >
        @DP_IndexClass(DLTable)
        </SELECT></TD>
        </TR>
        </TABLE>
        <BR><BR>
        <TR>
        <TD><INPUT TYPE="submit" NAME="get_IndexAttr"
                VALUE="Get Attribute">
        </TD></TR>
        </CENTER>
        </FORM>
</HTML>
%}
```

```
%(----------------  HTML Report Section  -----------------%}
%HTML_REPORT{
<HTML>
        <CENTER>
        <h2>Index Attribute of $(Index_Class)</h2>
        <TABLE cellpadding=1 cellspacing=5 border=1 width=80%>
        <TR>
        <TH>Row Number</TH>
        <TH>Index Attribute</TH>
        </TR>
        @DP_IndexAttribute(Index_Class,DLTable)
        </TABLE>
        </CENTER>
</HTML>
%}
```

This sample macro file consists of four major sections:

- The Define section
- The FUNCTION Definition section
- The HTML Input section
- The HTML Report section

You can have as many HTML sections as you want in a macro file. Also notice that these HTML sections contain familiar HTML tags, which makes writing macro files easy.

To build a macro, add a macro statement to be processed dynamically at the server. Two examples are: `@DP_IndexClass` in the HTML_INPUT section and `@DP_IndexAttribute` in the HTML_REPORT section.

Examine the sample macro, section by section, to understand the macro execution:

Define section

```
%(-------------------- Define Section ----------------------%}
%DEFINE{
        DLTable  = %TABLE(100)
        PROGRAM  = "/cgi-bin/db2www.exe/frndp2.d2w"
%}
```

You can define one or multiple variables in a single define section. Once defined, these variables can be referenced anywhere in the macro file using the syntax `$(Variable)`.

FUNCTION definition section

```
%(------------ FUNCTION  Definition Section --------------%}
%FUNCTION(DTW_DLDPB) DP_IndexClass (OUT table) {
<—— This function has no need for an ——>
<—— input parameter, but will return ——>
<—— the result in a table.           ——>

    DP_IndexClass;

<—— This function invokes the Dynamic Page Builder API (DP_IndexClass).——>

    %REPORT{ %row{  <OPTION value=$(V1)>$(V1)  %}  %}
<—— The formatted return result is    ——>
<—— set to be the option value of the ——>
<—— selection box.                    ——>

    %MESSAGE {
```

```
<-- error detection.                    -->
        -160: "Error: 160: Catalog lookup failure" : exit
        %}
%}
%FUNCTION(DTW_DLDPB) DP_IndexAttribute (in idx, OUT table) {
    DP_IndexAttribute;
<-- This function invokes the Dynamic    -->
<-- Page Builder API (DP_IndexAttribute) -->

    INDEX_CLASS = $(Index_Class);
<-- pass index class from inline data    -->

    %REPORT  {
<-- format return result                 -->
                %row {
<-- loop through each row                -->
                        <TR>
<-- Table row                            -->
                        <TD> $(ROW_NUM) </TD>
<-- display row number                   -->
                        <TD> $(V1) </TD>
<-- display 1st column                   -->
                        </TR>
                    %}
           %}

%MESSAGE {
             -162:    "Error: 162: Catalog lookup failure" : exit
        %}
%}
```

This function definition section contains two function declarations. The first, `DP_IndexClass`, is the Dynamic Page Builder API ″get″ index class. It takes no input parameters and stores the return result in a single column table.

```
    %REPORT{ %row{  <OPTION value=$(V1)>$(V1)
      %}
%}
```

The return result is assigned to `<OPTION value=> format`, and builds the items of the selection box for you to pick the index class and display the index attributes at the second HTML page. The second function definition `DP_IndexAttribute` is the Dynamic Page Builder API ″get″ index attribute. It takes the single parameter `INDEX_CLASS` and returns the results in a single column table. The passing of input parameters as inline data is available in most of Dynamic Page Builder APIs. In the sample macro, `INDEX_CLASS` is passed through inline data, not input parameters. *Inline data will be ignored if both input parameters and inline data are provided.*

HTML input section

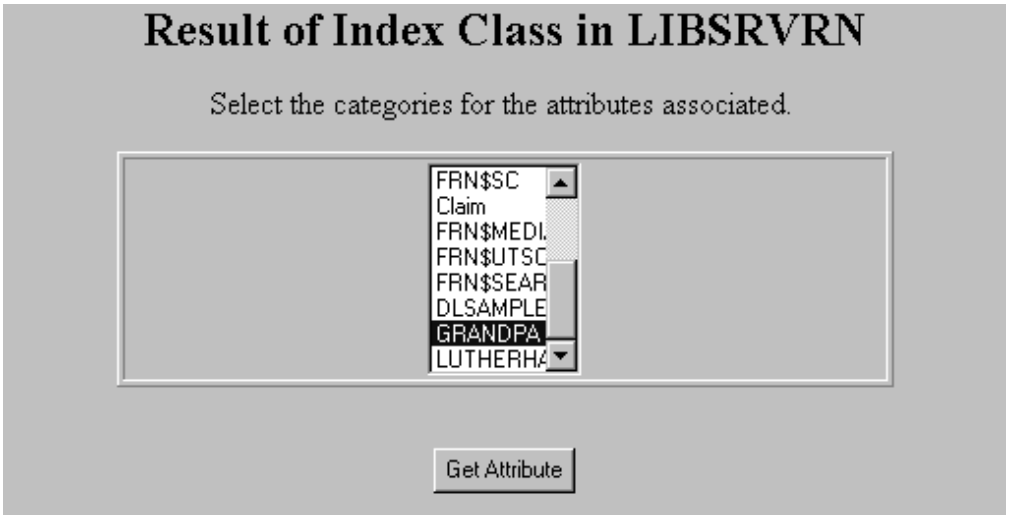
```
%(----------------- HTML Input Section ---------------------%}

%HTML_INPUT{
<-- Identifies the name of this section         -->
<HTML>
        <FORM METHOD=POST ACTION="$(PROGRAM)/report">
<-- Form tag with variable substitution;        -->
<-- this will invoke the $(PROGRAM)/report      -->
<-- section when this form is summitted.        -->
        <center>
        <h2> Result of Index Class in Digital Library server </h2>
        <p> Select the categories for the attributes associated. <br><p>

        <TABLE border=1 width=50%>
        <TR>
        <TD align=center> <SELECT NAME="Index_Class" SIZE=8 WIDTH=80>
```

```
        @DP_IndexClass(DLTable)
<— This line contains a call to DPP            —>
        </SELECT> </TD>
        </TR>
        </TABLE>
        <BR><BR>
        <TR>
        <TD> <INPUT TYPE="submit" NAME="get_IndexAttr"
                VALUE="Get Attribute">
        </TD> </TR>
        </CENTER>
        </FORM>
</HTML>
%}
```

The entire HTML section is surrounded by the HTML section identifier, `%HTML_INPUT { ... %}`. INPUT identifies the name of this section, which has been used in the URL also: http://www.ibm.com/cgi-bin/db2www/sample.d2w/INPUT

This section contains an example of a function call. The expression of `@DP_IndexClass(DLTable)` is a call to the Dynamic Page Builder API. This function is defined in the FUNCTION definition section described previously. The result of the DP_IndexClass function is inserted into the HTML text in the same location that the `@DP_IndexClass()` expression is located.

The following image shows what the input section looks like in the sample application:



HTML report section

```
%(-------------------  HTML Report Section  -----------------------%}
%HTML_REPORT{
<HTML>
        <CENTER>
        <h2>Index Attribute of $(Index_Class)</h2>
<—            Substitution from INPUT section              —>
<TABLE cellpadding=1 cellspacing=5 border=1 width=80%>
        <TR>
        <TH>Row Number</TH>
        <TH>Index Attribute</TH>
        </TR>
        @DP_IndexAttribute(Index_Class,DLTable)
<—                    call to a function                    —>
```

```
        </TABLE>
        </CENTER>
</HTML>
%}
```

Like the INPUT section, this section is standard HTML enhanced with a macro statement that substitutes a variable and a function call. The `Index_Class` variable is substituted into the title statement. DP_IndexAttribute is the function call to the FUNCTION definition called previously; it passes the content of the variable `Index_Class`, which was selected from the INPUT form.

DP_IndexAttribute also accepts inline parameters, as shown in the following statement:

```
              @DP_IndexAttribute(""
,DLTable)
```

This requires a variable defined through inline data in the FUNCTION definition section:

```
%FUNCTION(DTW_DLDPB) DP_IndexAttribute (in idx, OUT table) {
        DP_IndexAttribute;
        INDEX_CLASS = $(Index_Class);
<—              Index_Class referenced here            —>
        ...
%}
```

Index Attribute of GRANDPA

Row Number	Index Attribute
1	DLSEARCH_Author
2	DLSEARCH_Publisher
3	DLSEARCH_DocType
4	DLSEARCH_Date
5	DLSEARCH_Title
6	DLSEARCH_Subject
7	DLSEARCH_Authors
8	DLSEARCH_Editors
9	DLSEARCH_Source
10	DLSEARCH_Volume
11	DLSEARCH_Issue
12	DLSEARCH_Relation

Sample macro II

This section describes a sample macro that performs the parametric search and displays thumbnail images with hyperlinks to the document.

This macro file consists of three major sections:

- The Define section
- The FUNCTION Definition Section

- The HTML Input Section

Sample Macro II calls the HTML INPUT section in the program with the PPID parameter set to Not NULL.

```
%(------------------- Define Section --------------------%}
%define{
    DATABASE  = "LIBSRVRN"
    LOGIN     = "FRNADMIN"
    PASSWORD  = "PASSWORD"
    dlResult  = %TABLE(ALL)
    TOC_PART  = "2"
    THUMBID   = "5"
    PROGRAM   = "/cgi-bin/db2www.exe/frndp59.d2w"
    qst       = "SEARCH=(INDEX_CLASS=GRANDPA,MAX_RESULTS=10);"
%}

%(------------ FUNCTION Definition Section --------------%}
%FUNCTION(DTW_DLDPB) DP_Parts (in PID, OUT table) {
        DP_Parts;
DP_DESCRIBEID=$(TOC_PART);
%REPORT{
        %ROW {
            %IF (ROW_NUM == "3")
                    <FRAME SRC="$(V4)">
            %ENDIF
            %}
    %}
%}

%FUNCTION(DTW_DLDPB) DP_PMQuery (in PMQUERY_STRING,OUT table) {
DP_PMQuery;
DP_THUMBID=$(THUMBID);
        DP_DESCRIBEID=$(TOC_PART);
%REPORT {
        %ROW {
        @DTW_DIVREM(ROW_NUM,"3","5",REM)
        %IF (REM == "1")
                        <TR>
        %ENDIF
        <TD WIDTH=33.33% ALIGN=CENTER>
        <IMG SRC="$(V3)">
        <BR>
        %IF ($(V4) == "YES")
             <A HREF=$(PROGRAM)/input?PPID=$(V1)>$(V9)
        %ENDIF
        </TD>
        %IF (REM == "0")
                </TR>
        %ENDIF
        %}
%}

        %MESSAGE {
     -170:        "Error: 170: Query failure"        : exit
        %}
%}

%(----------------  HTML Section   --------------------%}
%HTML(INPUT){
<HTML>
        <head>
        <BASE HREF="http://jasonwu.stl.ibm.com/" >
        </head>
        <center>
        %IF (PPID != "")
            <FRAMESET FRAMEBORDER="no" ROWS="100%,*,*">
```

```
            <NOFRAMES>
            <BODY BGCOLOR="#ffffff" LINK="#000080"
            ALINK="#00ff00" VLINK="#557f55">
            </BODY>
            </NOFRAMES>
            @DP_Parts(PPID,dlResult)
            </FRAMESET>
%ELSE
        <br><font+1><strong> Query String </strong></font><br><br>
        $(qst) <br> <br>
        <font+1><strong> Result Sets </strong></font> <br>
        <HR size=3>  <br>
        <TABLE cellpadding=7 cellspacing=4 border=1 width=80%>
        @DP_PMQuery(qst,dlResult)
        </TABLE>
%ENDIF
</center>
</HTML>
%}
```

Define section

```
%(--------------------  Define Section  -------------------------%}
%DEFINE{
    DATABASE  = "LIBSRVRN"
<—              Library server name               —>
    LOGIN     = "FRNADMIN"
<— User logon ID —>
    PASSWORD  = "PASSWORD"
<—             User logon password               —>
    dlResult  = %TABLE(ALL)
<—              Result set table                  —>
    TOC_PART  = "2"
<—         Part number of description part        —>
    THUMBID    = "5"
<— Part number to display —>
    PROGRAM    = "/cgi-bin/db2www.exe/frndp59.d2w"
    qst        = "SEARCH=(INDEX_CLASS=GRANDPA,MAX_RESULTS=10);"
<—          Parametric Query string               —>
%}
```

You need the DATABASE,LOGIN, and PASSWORD variables in the Define section if they are defined in the db2www.ini file as ENVIRONMENT(DTW_DLDPB) dtwdldpb (IN DATABASE, LOGIN, PASSWORD, OUT RETURN_CODE) CLIETTE "DTW_DLDPB:$(DATABASE)".

FUNCTION definition section

```
%(------------------  FUNCTION Definition Section ----------------------%}

%FUNCTION(DTW_DLDPB) DP_Parts (in PID, OUT table) {
    DP_Parts;
<—                   Invoke DP_API                —>
    DP_DESCRIBEID=$(TOC_PART);
<—                Assign TOC part number          —>
    %REPORT{
        %ROW {
<—                Loop through each row           —>
            %IF (ROW_NUM == "3")
                <FRAME SRC="$(V4)">
            %ENDIF
<—      If the 3rd row, display the 4th column    —>
        %}
    %}
%}

%FUNCTION(DTW_DLDPB) DP_PMQuery (in PMQUERY_STRING,OUT table) {
    DP_PMQuery;
```

```
<—— Invoke DP_API                                ——>
    DP_THUMBID=$(THUMBID);
<—— Assign part number to be shown               ——>
        DP_DESCRIBEID=$(TOC_PART);
<—— Provide file name in TOC part for thumbnail;——>
<—— random file name will be                     ——>
<—— generated if DP_DESCRIBEID does              ——>
<—— not provide, or can't find the TOC part.     ——>
    %REPORT {
        %ROW {
            @DTW_DIVREM(ROW_NUM,"3","5",REM)
<—— Get remainder of dividing "3";               ——>
<—— For showing 3 records per row                ——>
        %IF (REM == "1")
<—— Remainder = "1"                    ——>
             <TR>
<—— New table row                                ——>
        %ENDIF
        <TD WIDTH=33.33% ALIGN=CENTER> <IMG SRC="$(V3)">
<—— Display the 3rd column                       ——>
        <BR>
            %IF ($(V4) == "YES")
                <A HREF=$(PROGRAM)/input?PPID=$(V1)>$(V9)
<—— Hyper link with PPID value                   ——>
        %ENDIF
        </TD>
        %IF (REM == "0")
            </TR>
<—— Close table row                              ——>
        %ENDIF
    %}
%}

    %MESSAGE {
     -170:    "Error: 170: Query failure"    : exit
    %}
```

This function definition section contains two function declarations. The first, `DP_Parts`, is the Dynamic Page Builder API ″get parts″ declaration for collections. It retrieves all the individual parts in the document and stores them in the `tmplobs` subdirectory of the HTML root directory. It takes one input parameter ″Part Item ID″ and stores the returned result in a five-column table.

$(V1) Part item ID

$(V2) File size

$(V3) File type

$(V4) File name

$(V5) Part ID

`DP_PMQuery` is the Dynamic Page builder API for processing parametric queries. It takes one input parameter, `PMQUERY_STRING`, and returns the results in the variable columns in the table.

$(V1) Part item ID

$(V2) Filter (″Document″, ″Folder″, or ″Unknown″)

$(V3) DKFolder (″YES″, BLANK, or file name of thumbnail image). In the results, you might receive the folder collection after the Parametric query. In this case, `$(V3)` is set to ″YES″ to indicate there is a folder collection, and you need to call `DP_Folder()` to process it. `$(V3)` can be blank to indicate that

no folder collection is associated with it. $(V3) can also be the file name of a thumbnail image if DP_THUMBID is provided and no folder collection can be found.

$(V4) DKParts (″YES″, BLANK). Setting $(V4) to ″YES″ indicates that there is an associated parts collection. You can call DP_Parts() to retrieve all the parts within the current parts collection.

$(V5)...$(Vn)
Values of index attributes.

HTML input section

```
%(------------------- HTML Section ---------------------%}
%HTML(INPUT){
<HTML>
    <head>
    <BASE HREF="http://jasonwu.stl.ibm.com/" >
    </head>
    <center>
    %IF (PPID != "")
<—          detect PPID variable, the first time            —>
<—          through will be NULL                            —>

        <FRAMESET FRAMEBORDER="no" ROWS="100%,*,*">
        <NOFRAMES>
        <BODY BGCOLOR="#ffffff" LINK="#000080" ALINK="#00ff00"
        VLINK="#557f55">
        </BODY>
        </NOFRAMES>
        @DP_Parts(PPID,dlResult)
<—          Invoke DP_API                                   —>
        </FRAMESET>
%ELSE
        <br> <font+1><strong> Query String </strong></font> <br><br>
        $(qst) <br> <br>
        <font+1><strong> Result Sets </strong></font> <br>
        <HR size=3>  <br>
        <TABLE cellpadding=7 cellspacing=4 border=1 width=80%>
           @DP_PMQuery(qst,dlResult)
<—          Invoke DP_API                                   —>
        </TABLE>
    %ENDIF
    </center>
</HTML>
%}
```

To run this program, send a URL request to your Web server, for example, http://WWW/cgi-bin/db2www.exe/frndp59.d2w/input, where WWW is the main Web server directory. To process the INPUT section (because PPID is NULL at this moment), the %ELSE portion is executed to invoke DP_PMQuery() for data processing, displaying a thumbnail image, and building a hyperlink to program itself with an assigned PPID value. You can choose to display the document by clicking the Title hyperlink to call the program and pass PPID as a parameter. The If statement is satisfied with the PPID assigned and DP_Parts() is invoked to retrieve the document.

The following images show what this section looks like in the sample application:

Query String

SEARCH=(INDEX_CLASS=GRANDPA,MAX_RESULTS=10);

Result Sets

On some occasions you might receive a broken image that indicates problems with receiving the images. If this happens, make sure the image object is available in the object server, or the thumbnail id is valid, or the object that resides in the object server is not corrupt.

See the Net.Data Programming and Reference Guide for more information about macro programming.

Improving performance

Live connection

Digital Library Dynamic Page Builder works with the Net.Data Live Connection Manager to improve performance by eliminating start-up overhead. A live connection consists of a connection manager and cliettes. Cliettes are single-threaded processes the connection manager starts and keeps resident while the server is running, processing data, and communicating with the Net.Data environment.

There are two main advantages of live connection:

Enhanced performance:
Save time by avoiding the library server connection.

Sequential display:
This offers the ability to use ″next″ and ″previous″ options, which are useful for navigating large result sets in an application. By setting the variables `START_ROW_NUM` and `RPT_MAX_ROW`, you can break down large query hits to show more manageable portions.

Reuse dynamic pages

Dynamic page builder retrieves image content from Digital Library during run time, and stores it on the server's disk with the file name provided in the TOC part.

A TOC part is one of the parts within the document that identifies the relationship between an individual part and the file name it is associated with. An example TOC part is shown below (″anynet.htm″ is an HTML file):

```
anynet.htm:3
abstract.txt:4
anynet1t.gif:5
sqcloud0.gif:17
sqclouda.gif:18
bigsq.gif:19
dot_clea.gif:20
dot_clea.gif:21
anynet1.gif:22
anynet2.gif:23
sqhome.gif:24
tellsq.gif:25
getsq.gif:26
software.gif:27
```

With the TOC part, you can reduce the overhead associated with file input and output by setting the environment variable `DLDPB_FILEMODE=2` when the connection manager is invoked. With `DLDPB_FILEMODE` set to `2`, the Dynamic Page Builder does not overwrite the existing file, so this file can be reused.

Invoking the wizard

Net.Data offers a wizard to help you access Net.Data function from your application.

The way to invoke the Digital Library Dynamic Page Builder wizard depends on the platform and shell script you use:

Script File:	Platform:
DPWizard.bat	WindowNT
DPWizard.ksh	AIX Korn Shell
DPWizard.csh	AIX C Shell

After you complete all the data fields in the Dynamic Page Builder wizard, a macro file with the file name that you specifed is created in the current directory. Copy the created macro file to the MACRO_PATH directory as defined in the db2www.ini file, so it can be executed from your Web application.

Starting the Dynamic Page Builder sample

There are many different kinds of Web servers available. This section describes how to use some of them.

The following examples assume you use C as your drive, DB2WWW as root directory for ICS on Windows NT, and InetPub as root directory for IIS on Windows NT.

Web server configuration

Windows NT with the IIS V2 Web server

Bring up the Microsoft Internet Service Manager and double click on thecomputer name of WWW service to display the WWW service properties for your machine. Click on Directories to create cgi-bin, tmplobs, and frnroot aliases, and make sure the corresponding directories are specified. Choose READ and EXECUTE access rights for cgi-bin. Choose READ only for tmplobs. For example:

```
Directory: c:\InetPub\wwwroot\cgi-bin Alias: /cgi-bin
Directory: c:\InetPub\wwwroot\tmplobs Alias: /tmplobs
Directory: c:\frnroot Alias: /frnroot  (Assume DL is installed
at C:\FRNROOT)
```

Make sure the corresponding directories have been created on the file system.

Copy Digital Library files from the Net.Data HTML directory to <HOME> Alias, for example:

```
copy c:\db2www\html\*.*            c:\InetPub\wwwroot
copy c:\db2www\cgi-bin\db2www.exe  c:\InetPub\wwwroot\cgi-bin
```

Modify c:\InetPub\wwwroot\db2www.ini to make sure HTML_PATH and INCLUDE_PATH point to c:\InetPub\wwwroot.

Windows NT with the IBM ICS Web server

This is the Net.Data default setup for Windows NT 4.0, so all you need to do is make sure the cgi-bin, and tmplobs, and frnroot aliases are set and the directories exist:

```
Directory: c:\db2www\cgi-bin       Alias: /cgi-bin
Directory: c:\db2www\html\tmplobs  Alias: /tmplobs
Directory: c:\frnroot              Alias: /frnroot
```

You can verify this by using your Web browser to access http://machine/admin-bin/cfgin/mpfrule for Request Routing.

IBM AIX with the IBM ICS Web server

This is the Net.Data default setup for AIX, so you don't need to do any extra configuration with your Web server except to create a directory named `/usr/lpp/internet/server_root/pub/tmplobs` for BLOBs, and make sure you bring up the Web server as a subsystem. `startsrc -s httpd` should do.

Connection manager setup

Windows NT 4.0 platform

- Modify the `dtwcm.cnf` file in the `c:\db2www\connect` directory.
- In this file, locate the section starting with `DTW_DLDPB` and modify the parameters. For example:

```
CLIETTE DTW_DLDPB:LIBSRVRN{     ——> Library server
MIN_PROCESS=1                   ——> Minimum start
MAX_PROCESS=5                   ——> Maximum process
START_PRIVATE_PORT=7150
START_PUBLIC_PORT=7350
EXEC_NAME=cldldpb.exe
DATABASE=LIBSRVRN               ——> Library server
BINDFILE=NOT_USED
LOGIN=FRNDPMIN                  ——> DP user ID
PASSWORD=PASSWORD               ——> DP password
  }
```

- Set `MIN_PROCESS` parameter to zero for the rest of the sections in the file.
- Bring up the Library server and the Object server.
- Click on the Net.data icon and bring up the Live Connection Manager

IBM AIX OS

- Modify the `dtwcm.cnf` file in the `/usr/lpp/internet/db2www/db2` directory
- In this file, locate the section starting with `DTW_DLDPB` and modify the parameters, for example:

```
CLIETTE DTW_DLDPB:LIBSRVRX{             ——> Library Server
MIN_PROCESS=1                           ——> Minimum start
MAX_PROCESS=5                           ——> Maximum process
START_PRIVATE_PORT=7150
START_PUBLIC_PORT=7350
EXEC_NAME=./cldldpb DATABASE=LIBSRVRX   ——> Library server
BINDFILE=NOT_USED LOGIN=FRNDPMIN        ——> DP user ID
PASSWORD=PASSWORD                       ——> DP password
  }
```

- Set `MIN_PROCESS` parameter to zero for the rest of the sections in this file
- Bring up the Library server and the Object server
- Bring up the Live Connection Manager by typing `dtwcm -d` from the directory of `/usr/lpp/internet/db2www/db2`

Configuring sample macro files

- Specify `SERVER`, `DATABASE`, `LOGIN`, `PASSWORDQB_CATALOGQB_DATABASEQB_CONNECT` and `HTMLROOT` parameters in files `frndp15.d2w` and `frndp25.d2w`, which are located under `c:\db2www\macro` for Windows NT and `/usr/lpp/internet/db2www/macro` for AIX:

```
%define{
SERVER = "Enter Web server machine host name here"
DATABASE = "Enter Library server name here"
LOGIN  = "Enter Library server user id here"
PASSWORD = "Enter Library server password here"
QB_CATALOG   = "Enter QBIC Catalog here"
```

```
QB_DATABASE  = "Enter QBIC Database here"
QB_CONNECT   = "Enter QBIC_Server/UserID/Password here"
HTMLROOT     = "Enter Default HTML root directory here"
%}
```

- Setup URL address in files `dlheader.html`, which are located under `c:\InetPub\wwwroot` for Windows NT with IIS Web server, `c:\db2www\html` for Windows NT with ICS Web server, and `/usr/lpp/internet/server_root/pub` for IBM AIX with ICS Web server; for example:

  ```
  <head>
  <title>IBM Digital Library Internet Connection</title>
  <BASE HREF="http://xyz/">
  </head>
  ```

  Note: `xyz` is your Web server URL address for the Internet or Web server host machine name (or IP address) for your intranet.
- Set up URL address in files of dpcolorapplet.html, dphistogramapplet.html, and dpdrawapplet.html, which are located under `-frnroot` (DL installed directory) for Windows NT, `/usr/lpp/internet/server_root/pub` for AIX.

Running the sample macro

- Type `http://xyz/dlqbic.html` from your Web browser (**Note:** `xyz` is your Web server URL address for the Internet or Web server host machine name (or IP address) for your intranet)

Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Programming interface information

This publication, *Guide to Object-Oriented and Internet Application Programming*, documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM DB2 Digital Library.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

ADSTAR	AIX	DB2
DB2 Universal Database	IBM	Net.Data
QBIC	VideoCharger	

Java and HotJava are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

This glossary defines terms and abbreviations specific to the Digital Library system. Terms used in the definitions that are shown in *italics* are defined elsewhere in this glossary.

A

access control. The ability to limit access to certain functions provided by the Digital Library system and certain objects stored in the system. For example, you can give a user read-only access to an object through an *access list*.

access list. A list consisting of one or more individual user IDs or user groups and the *privilege set* associated with each user ID or user group. You use access lists to control user access to objects in the Digital Library system. The objects that can be associated with access lists are the data objects stored by users, *index classes* and subsets, *workbaskets*, and *workflows*.

ADSM. ADSTAR Distributed Storage Manager

ADSM management class. A logical area of storage that is managed by ADSM.

API. Application programming interface.

archive. Persistent storage used for long-term information retention, typically very inexpensive per unit stored and slow to access, and often in a different geographic location to protect against equipment failures and natural disasters.

attribute. The term used in the Digital Library APIs for *key field*. Not the same as the attribute in an *attribute/value pair*.

attribute/value pair. Data entered in HTML *form elements*, and other data associated with the HTML form elements. For example, the attribute can be the name of an HTML form element, such as a text entry field. The value is then what the user enters into that form element, such as a text term. Not the same as an attribute for a Digital Library index class.

B

base attributes. A set of indexes that is assigned to each object. All Digital Library objects have base attributes, also known as *key values*.

binary large object (BLOB). A large stream of binary data that comprises an entity, such as the objects stored and cataloged in a digital library.

BLOB. Binary large object.

C

cache. An area of storage used to temporarily store objects on the client machine. See also *object server cache* and *LAN cache*.

catalog. An index used by Digital Library applications to insert and remove objects that are not yet assigned to a list.

category. Another term for *index class*. In the Dynamic Page Builder sample application, it is the name of a list box that contains index classes for collections of magazine articles.

CGI. Common gateway interface.

CGI script. A program, usually written in C or PERL, that collects data and delivers it to the server. Digital Library provides a CGI script in the form of a C++ sample application program that routes information between Web pages and the *cliette.*

cliette. In the Dynamic Page Builder, the interface between the Web server and the library client search engines.

client application. An application written with the Digital Library APIs to customize a user interface.

Client Application for Windows. A complete object management system provided with Digital Library and written with Digital Library APIs. It supports document and folder creation, storage, presentation, processing, and access control. It can be customized with user exits and can be partially invoked with APIs.

collection. A group of objects with a similar set of management rules and contained within a *storage group*. Every object is in a collection, and all objects in a collection are stored in the same *storage group*.

Connection Manager. A Digital Library component that helps maintain connections to the library server, rather than starting a new connection for each query. The Connection Manager has an application programming interface.

container. In the *folder manager,* an object that can contain other folders or documents.

content class. The term used in the Digital Library APIs for *data format*.

custom system. A tailored configuration of a Digital Library system that includes two or more nodes installed on a local area network (LAN).

D

data format. A logical name assigned to a file type, valid only within the Digital Library system. Digital Library provides a large range of predefined data formats. You can also define your own data format. For example, you can define your own data format called MYGIF and use it for GIF files.

destager. A function of the object server that moves objects from the *staging area* to the first step in the object's *migration policy*. You can start and stop the destager from the object server interface or the command utility. You specify in the Object Server Configuration notebook how often the destager should be run.

document. An object that can be stored, retrieved, and exchanged among Digital Library systems and users as a separate unit. Also called an *object*, it can be any multimedia digital object. A single document can contain many different types of base parts, including text, images, and objects such as spreadsheet files.

dynamic page builder. An API in the *Internet application development toolkit* that is used to create applications that dynamically format the results of queries and display those results on a Web page.

E

element. An *object* that the list manager allocates for an application.

F

feature. The visual content information that is stored in the image search server. Also, the visual traits that image search applications use to determine matches. The four *QBIC* features are average color, histogram color, positional color, and texture.

file system. In AIX, the method of partitioning a hard drive for storage.

folder. An object that can contain other folders or documents. A folder can be indexed by key fields.

folder manager. The Digital Library data model for managing digital objects as online documents and folders. You can use the folder manager APIs as the primary interface between your applications and the Digital Library data store.

form element. In the Dynamic Page Builder, an element of the interface that can be created with HTML tags. The Dynamic Page Builder supports several HTML form elements.

H

handle. A character string that represents an object, and is used to retrieve the object.

history log. A file that keeps a record of activities for a workflow.

HTML. Hypertext markup language.

I

index class. A category for storing and retrieving objects, consisting of a named set of attributes. When you create an object in the Digital Library system, your application must assign an index class and supply the *key field* values required by that class. An index class identifies the key fields, automatic processing requirements, and storage requirements for an object.

index class subset. A view of an *index class* that an application uses to store, retrieve, and display folders and objects.

index class view. The term used in the APIs for *index class subset*.

Internet application development toolkit. A set of object-oriented APIs for building query applications for the World Wide Web. These applications can accept input from a Web browser for searching the Digital Library data store, and display the results in the browser.

item. The smallest unit of information that the library server administers. An item can be a folder, document, workbasket, or workflow. Referred to as an *object* outside of library server functions.

K

key field. An attribute of an object that represents a type of information about that object. For example, a video object might have key fields for the title and length of the video.

L

LAN cache. An area of temporary storage on a local *object server* that contains a copy of objects stored on a remote object server.

library client. The component of a Digital Library system that provides a low-level programming interface for the library system. The library client includes APIs that are part of the software developer's kit.

library object. In the Digital Library system, a *folder*, *document*, or *workbasket*. Library objects are stored on the *library server*.

library server. The component of a Digital Library system that contains index information for the objects stored on one or more *object servers*.

list. A data structure, or electronic queue, that your application uses to temporarily insert or remove objects for work. The *list manager* allocates lists in its file space for your application.

list manager. An OS/2 component in a Digital Library system that controls *cache*. The list manager server manages electronic queues, or *lists*, for the clients that link to it.

The list manager server includes a server process and a dynamic link library (DLL) that contains the APIs for the list manager. The *configuration server* is a type of list manager server.

M

management class. The term used in the APIs for *migration policy*.

media archiver. A physical device that is used for storing stream data (audio and video data). The VideoCharger is a type of media archiver.

media server. An AIX-based component of the Digital Library system that is used for storing and accessing video files.

migration policy. A user-defined schedule for moving objects from one *storage class* to the next. It describes the retention and class transition characteristics for a group of objects in a storage hierarchy.

migrator. A function of the object server that checks *migration policies* and moves objects to the next *storage class* when they are scheduled to move. You can start and stop the migrator from the object server interface or the command utility. You specify in the Object Server Configuration notebook how often the migrator should run.

multi-search query. A query that combines parametric and text searches.

N

network table file. A text file that contains the system-specific configuration information for each node in a Digital Library system. Each node in the system must have a network table file that identifies the node and lists the nodes that it needs to connect to.

The name of a network table is always FRNOLNT.TBL.

O

object. Any data entity stored on an object server in digital form.

In the *folder manager* data model, *object* specifically refers to a document's contents or base parts.

object server. The component of a Digital Library system that physically stores the objects or information accessed by your client applications.

object server cache. The working storage area for the *object server*. Also called the *staging area*.

P

part. A subelement of an *item* that corresponds to a data object or *BLOB* that is stored on an object server. An item can contain parts with diverse binary formats. For example, a user can create an object that contains an image, a word processing file, and a spreadsheet.

patron. The term used in the Digital Library APIs for *user*.

privilege. An authorization for a user to either access or perform certain tasks on objects stored in the system. Privileges are assigned by the Digital Library system administrator.

privilege set. A collection of *privileges* for working with system components and functions. The administrator assigns privilege sets to users (user IDs) and user groups.

property. A characteristic of an object that can be changed or modified. The properties of an object describe the object. Type style is an example of a property.

purger. A function of the *object server* that removes objects from the system. You can start and stop the purger from the object server interface or the command utility. You specify in the Object Server Configuration notebook when the purger should run.

S

SMS server. System-managed storage server.

Software developers toolkit. A set of application programming interfaces that you can use to build custom Digital Library applications. The toolkit includes APIs for the folder manager and the library client.

staging. The process of moving a stored object from an offline or low-priority device back to an online or higher priority device, usually on demand of the system or on request of a user. When a user requests an object stored in permanent storage, a working copy is written to the *staging area*.

staging area. The working storage area for the *object server*. Also referred to as *object server cache*.

stand-alone system. A preconfigured Digital Library system that installs all of the components of a Digital Library system on a single personal computer.

storage class. A named list of storage attributes. The list of attributes identifies a storage service level provided for data that is associated with the storage class. No physical storage is directly associated with a given storage class name. However, device drivers are associated with the storage class, and *storage groups* are used with a storage class to limit the physical storage.

Storage classes are defined during the initial installation of a Digital Library system or during customization.

storage class identifier. The identifier associated with a storage class. This identifier is assigned by the Digital Library system, not by the user.

storage group. A named collection of physical devices to be managed as a single object storage area. You use a storage group to define a category in which to store groups of objects. A storage group is used to limit a *storage class*.

Storage groups are defined during initial installation of a Digital Library system, during customization, or dynamically by a run-time function.

storage group identifier. The identifier associated with a storage group. This identifier is assigned by the Digital Library system, not by the user.

storage system. A generic term for storage in the Digital Library system. See *ADSM management class*, *media archiver*, *volume*

streamed data. Multimedia data sent over a network connection at a specified rate. A stream can be video only, audio only, or a combination of both video and audio in one stream. Data rates, which are expressed in bits per second, vary for different types of streams and networks.

U

user. A person who requires the services of Digital Library. This term generally refers to users of client applications, rather than the developers of applications, who use the Digital Library APIs.

user group. A group consisting of one or more individual user IDs defined under a single group name.

utility server. A Digital Library component that is used by the database utilities for scheduling purposes. You configure a utility server when you configure an *object server* or *library server*. There is one utility server for each object server and each library server.

V

volume. A representation of an actual physical storage device or unit that the objects in your system are stored on.

W

workbasket. A collection of *objects* and *folders* that are either in process or waiting to be processed. A workbasket definition includes the rules that govern the presentation, status, and security of its contents.

workflow. A sequence of *workbaskets* that a document or folder travels through while it is being processed.

Index

A

Application Programming Interface (API)
  ActiveX 157
    Installation 158
    Multi-search 157
  C++ 13
  Dynamic Page Builder 201
    configuring DPB with Net.Data 201
    connection manager 218
    Environment (DTW_DLDPB) 202
    HTML_PATH 202
    INCLUDE_PATH 202
    MACRO_PATH 202
    parameters 203
    performance 216
    reserved terms 202
    sample macro 1 206
    sample macro 2 210
    sample startup 217
    server configuration 217
    wizard 216
  Java 77
    Architecture 79
    Differences from C++ 80
    Multi-search 14, 78
    Packaging 79

C

capture subsystem, used with Digital Library 2
Collections and Iterators
  ActiveX 163
    Adding 163
    attribute, DKFOLDER 168
    attribute, DKPARTS 167
    data_item values 164
    Deleting 169
    Displaying 165
    Information, Digital Library 166
    Information, Text Search Engine 167
    Pid 163
    properties 164
  C++ 21, 24
    Adding 25
    attribute, DKFOLDER 31
    attribute, DKPARTS 30
    Creating 25
    data_item values 26
    deleting 32
    Displaying 27
    Information, Digital Library 28
    Information, Text Search Engine 29
    Memory Management 23
    Pid 25
    Programming Tips 24
    properties 27
    Sequential Collection 21
Collections and Iterators *(continued)*
  C++ 163, 24 *(continued)*
    Sequential Iterator 25
    Sorting 24
  Java 84, 85
    Adding 86
    attribute, DKFOLDER 92
    attribute, DKPARTS 91
    Creating 86
    data_item values 87
    Displaying 88
    Information, Digital Library 89
    Information, Text Search Engine 90
    Pid 86
    properties 88
    Sequential Collection 84
    Sequential Iterator 84
    Sorting 85
Combined query
  ActiveX 190
    parametric with text 190
    Programming tips 192
    ranking 191
    using a scope 191
  C++ 67
    parametric with text 68
    Programming tips 69
    ranking 69
    using a scope 68
  Java 123
    parametric with text 123
    Programming tips 124
    ranking 124
    using a scope 124

D

data concepts, persistent 8
Datastore DL
  ActiveX 158
    Connecting 158
    Datastore DL Options 159
    List Schema and Schema Attributes 159
    List Servers 159
  C++ 15
    Connecting 15
    Datastore DL Options 15
    List Schema and Schema Attributes 16
    List Servers 15
    Programming Tips 17
  Java 80
    Connecting 80
    Datastore DL Options 80
    List Schema and Schema Attributes 81
    List Servers 81
Datastore TS
  ActiveX 160
    Connecting 160

Datastore TS *(continued)*
Datastore TS Options 160
List Schema 161
List Servers 161
Assignment from 20
Assignment to 20
C++ 17
Connecting 17
Datastore TS Options 18
List Schema 18
List Servers 18
Destroying 21
Display of 20
Java 82
Connecting 82
Datastore TS Options 83
List Schema 83
List Servers 83
Memory Management 19
Programming Tips 21
type_code, Getting 19
Typecode 19
Using Type Constructors 19
Digital Library solution 2

I

image search server, description 3
Internet gateway, used with Digital Library 2

N

new features
in Version 2 4
in Version 2.4 5
Notices 221

O

object management
ActiveX 185
creating 185
deleting 188
updating 186
C++ 61
creating 61
deleting 65
updating 63
Java 117
creating 117
deleting 121
updating 119

P

persistent data concepts 8

Q

Query
ActiveX 175
Query *(continued)*
parametric type 175
text type 177
C++ 49
dkResultSetCursor vs DKResults 49
parametric type 50
query object types 49
text type 52
Java 107
dkResultSetCursor vs DKResults 107
parametric type 107
query object types 107
text type 109
Queryable collection
ActiveX 189
evaluating 189
getting results 189
Programming tips 190
queryable vs refined 190
C++
evaluating 66
getting results 66
Programming tips 67
queryable vs refined 67
Java 121
evaluating 122
getting results 121
Programming tips 123
queryable vs refined 122

R

Result set cursor
ActiveX 182
C++ 57
creating a collection 58
open and close 57
set and get 58
Java 114
creating a collection 115
open and close 114
set and get 115
Retrieval
ActiveX 183
documents 183
folders 184
parts 184
C++ 59
folders 60
parts 59
Java 116
folders 117
parts 116

S

scenarios
Digital Library solution 1
Settings
C++ 71
Building on AIX 72

Settings *(continued)*
Building on NT 71
On AIX 71
On NT 72
Java 127
Client connect and disconnect 128
Client/Server 128
On AIX 127
On NT 128
Programming tips 131
Setup environment 129
Using sample Java applets and servlet 137
Connect 137
Dynamic Page Builder 154
Java application on DL client 142
Local access 143
Remote access 144
Retrieve servlet 142
View 141
Stand-alone code examples
Java
annotation type 97
from buffer 94
from file 95
search indexed by Text Search Engine 96
storage subsystem 2
subsystems
capture, used with Digital Library 2
Internet gateway, used with Digital Library 2
storage 2

T

Text Search Engine
ActiveX 192
Boolean query 192
Client/Server mode 194
Free text query 193
GTR query 193
Hybrid query 193
Load and index data 193
MAXPIECE 194
Programming tip 192
Proximity query 193
C++ 69
Boolean query 70
Exceptions 14
Free text query 70
GTR query 71
Hybrid query 70
Load and index data 71
MAXPIECE 13
Programming tip 69
Proximity query 70
Java 125
Boolean query 125
Exceptions 78
Free text query 125
GTR query 126
Hybrid query 126
Load and index data 126
Text Search Engine *(continued)*
Java 192 *(continued)*
MAXPIECE 125
Programming tip 125
Proximity query 126
Setting heap size 78
triangle architecture, description 3

X

XDO
ActiveX 169
annotation type 37
data members 170
DDO, part of 169
from buffer 35
from file 35
in datastore 175
indexing 170
Programming Tips 170
search indexed by Text Search Engine 36
stand-alone 171
C++ 32
data members 33
DDO, part of 34
indexing 33
Pid 32
Programming Tips 33
stand-alone 35
Java 92
data members 93
DDO, part of 94
indexing 93
Pid 93
Programming tips 93
stand-alone 94

Program Number: 5765–C86

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-8868-02