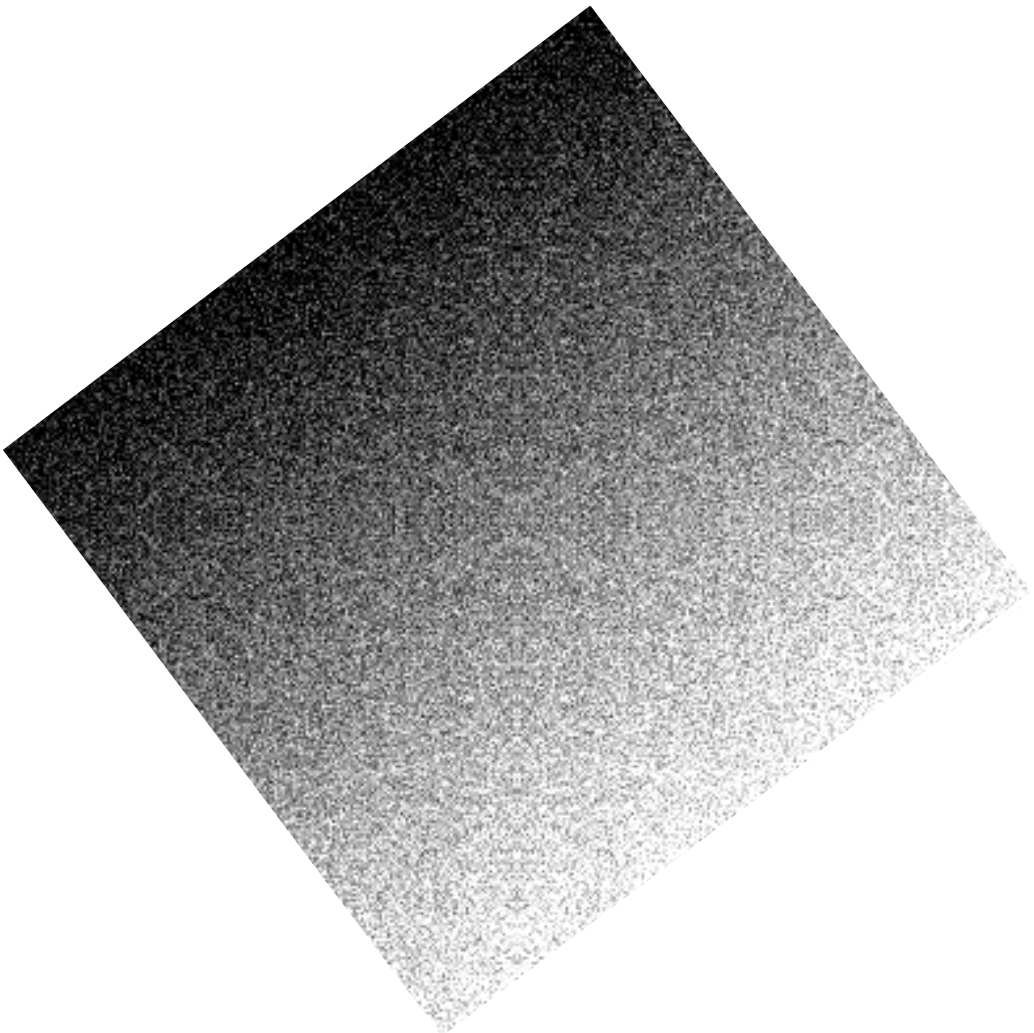
Intelligent Decision Server

IBM

Developing Applications with Open Data Access Service

Version 1



SC26-8888-00

Intelligent Decision Server

IBM

Developing Applications with Open Data Access Service

Version 1

SC26-8888-00

Note

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

First Edition (October 1996)

This edition applies to Version 1 of Intelligent Decision Server, 5639–IDS, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1996. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices ix
Trademarks ix

About This Book xi
Who Should Read This Publication xi
Prerequisite Reading xi
We'd Like Your Comments xii

Chapter 1. Introducing Open Data Access Service 1
Functionality 1
Connection Methods 2
Execution of ODAS Requests 3
  Capsule Services 3
  Conversion Services 4
ODAS Architecture 5
  ODAS Server 5
  Client Workstation 6

Chapter 2. Programming ODAS Clients 8
Locating Header Files Required for C Applications 8
Compiling C Applications 8
  Including Preprocessor Statements 8
  Compiling and Linking 9
Software for Visual Basic Applications 10
Identifying Desktop Objects 10
  Containers 10
  Objects with Non-Unique Names 11
  Wildcard Characters 12
Flags and File Types 12
  Defined Client File Types 12
  Defined Desktop Object Types 13
  Common Bit Flags 14
  Capsule Job Parameter Flags 16

Chapter 3. Using the C Language API Functions 18
Starting ODAS Application 18
Preventing Command Clashes 18
Connecting to a Desktop 18
Disconnecting from a Desktop 19
Viewing the Mail Tray 19
  Listing the Mail 19
  Retrieving an Envelope from the Mail Tray 22
  Retrieving Enclosures of an Envelope 22
  Sending Mail 23
Moving Files to the Desktop 25

© Copyright IBM Corp. 1996

Retrieving Files from the Desktop 26
Using Capsule Applications 27
Setting Capsule Variables 27
Running Capsule Applications 28
Checking a Capsule Application's Progress 29
Advanced ODAS Programming 30
Allocation of Output Buffers 30
Object Identifiers 30
Asynchronous Processing 31
Maximum Application Performance 32

Chapter 4. API Functions for C-Language Programming 37
Functional Lists of API Functions 37
Defined Data Types 39
Functions Reference Section 40
DisAFD 40
DisAFDList 41
DisCancelRequest 41
DisConnect 42
DisCopyObject 43
DisDeleteObject 44
DisDisconnect 44
DisExecuteAFD 45
DisGetCapsuleVars 46
DisGetErrorInfo 47
DisGetMultObjects 48
DisGetObject 51
DisGetPerfInfo 53
DisGetSysMessage 55
DisInitAFD 56
DisInitAFDList 57
DisInitBatch 58
DisInitClient 59
DisLogoff 59
DisLogon 60
DisMoveObject 61
DisPutMultObjects 62
DisPutObject 63
DisQueryCapsuleInfo 64
DisQueryObjectInfo 67
DisQueryResult 69
DisQuerySessionInfo 70
DisSetCapsuleStatus 72
DisSetCapsuleVars 74
DisSetConversionTable 75
DisSetCurrentContainer 77
DisSetEventHandler 77
DisSetPassword 81

DisSetSessionInfo . . . 81
DisSubmitBatch . . . 83
DisSubmitCapsule . . . 84
DisSubmitCapsuleAndSetVars . . . 86
DisTermAFD . . . 88
DisTermBatch . . . 88
DisTermClient . . . 89
DisWaitCapsuleStatus . . . 90

Chapter 5. Using the Visual Basic API Functions . . . 91
Starting ODAS . . . 91
Preventing Command Clashes . . . 91
Connecting to a Desktop . . . 91
Disconnecting from a Desktop . . . 92
Viewing the Mail Tray . . . 93
Listing the Mail . . . 93
Retrieving an Envelope from the Mail Tray . . . 95
Retrieving all Enclosures of an Envelope . . . 95
Sending Mail . . . 96
Moving Files to the Desktop . . . 98
Retrieving Files from the Desktop . . . 100
Using Capsule Applications . . . 100
Setting Capsule Variables . . . 100
Running Capsule Applications . . . 102
Checking a Capsule Application's Progress . . . 104
Advanced ODAS Programming . . . 105
Object Identifiers . . . 105
Asynchronous Processing . . . 105
Maximum Application Performance . . . 107

Chapter 6. API Functions for Visual Basic Programming . . . 112
Functional Lists of API Functions . . . 112
Special Functions Specific to Visual Basic . . . 115
Data Transfer . . . 115
Format Strings for Defined Types . . . 117
Error Codes for Visual Basic Functions . . . 118
Functions Reference Section . . . 118
DisAFD . . . 119
DisAFDList . . . 119
DisBufferToVBArray . . . 120
DisBufferToVBType . . . 121
DisCancelRequest . . . 123
DisCFormatSize . . . 123
DisConnect . . . 124
DisCopyObject . . . 125
DisCreateBuffer . . . 127
DisDeleteObject . . . 127
DisDisconnect . . . 128

DisFreeBuffer . . . 129
DisGetCapsuleVars . . . 129
DisGetErrorInfo . . . 131
DisGetMultObjects . . . 132
DisGetObject . . . 135
DisGetPerfInfo . . . 137
DisGetSysMessage . . . 139
DisInitBatch . . . 140
DisInitClient . . . 141
DisLogoff . . . 141
DisLogon . . . 142
DisMkTime . . . 143
DisMoveObject . . . 144
DisPutMultObjects . . . 145
DisPutObject . . . 147
DisQueryCapsuleInfo . . . 148
DisQueryObjectInfo . . . 151
DisQueryResult . . . 154
DisQuerySessionInfo . . . 155
DisSetCapsuleStatus . . . 157
DisSetCapsuleVars . . . 159
DisSetConversionTable . . . 162
DisSetCurrentContainer . . . 164
DisSetEventHandler . . . 164
DisSetPassword . . . 165
DisSetSessionInfo . . . 165
DisSubmitBatch . . . 167
DisSubmitCapsule . . . 168
DisSubmitCapsuleAndSetVars . . . 170
DisTermBatch . . . 172
DisTermClient . . . 173
DisVBArrayToBuffer . . . 173
DisVBFormatSize . . . 175
DisVBTypeToBuffer . . . 175
DisWaitCapsuleStatus . . . 176

Chapter 7. Custom Visual Basic Controls for ODAS . . . 178
Hardware and Software Requirements . . . 178
Installing the Controls . . . 178
Adding a Control to Your Application . . . 179
About the Controls . . . 179
The Browser . . . 180
DisCapQ . . . 181
DisCom and ODisCom . . . 182
DisConv . . . 182
DisDir . . . 183
DisSetVs . . . 183
DisSubCp . . . 183

DisTxfer and ODisTxfr . . . 183
DisXfr and ODisXfr . . . 184

Appendix A. C API Type Definitions . . . 185

Appendix B. Program Code with C-Language API Functions . . . 188

Appendix C. Visual Basic API Type Definitions . . . 192

Appendix D. Sample Program Code with Visual Basic API Functions . . . 195
SETUP.FRM . . . 195
LOGON.FRM . . . 197
GETOBJ.FRM . . . 199
SUBCAP.FRM . . . 202

Appendix E. Custom Visual Basic Controls Reference . . . 206
Browser . . . 206
Error Codes . . . 206
Methods . . . 208
Events . . . 208
Event Details . . . 209
Properties . . . 211
Property Details . . . 214
DisCapQ . . . 228
Limitations . . . 228
Assumptions . . . 228
Error Codes . . . 228
Methods . . . 228
Events . . . 229
Properties . . . 230
Property Details . . . 232
DisCom and ODisCom . . . 237
Limitations . . . 237
Assumptions . . . 238
Error Codes . . . 238
Events . . . 238
Event Details . . . 240
Properties . . . 242
Property Details . . . 243
DisConv . . . 248
Limitations . . . 248
Assumptions . . . 248
Error Codes . . . 248
Methods . . . 249
Events . . . 249
Properties . . . 249
Property Details . . . 250
DisDir . . . 252

Limitations . . . 252
Assumptions . . . 252
Error Codes . . . 252
Methods . . . 253
Events . . . 253
Properties . . . 255
Property Details . . . 258
DisSetVs . . . 263
Limitations . . . 263
Assumptions . . . 263
Error Codes . . . 264
Methods . . . 264
Events . . . 264
Properties . . . 265
Property Details . . . 266
DisSubCp . . . 268
Limitations . . . 268
Assumptions . . . 268
Error Codes . . . 269
Events . . . 269
Properties . . . 269
Property Details . . . 271
DisTxfer and ODisTxfr . . . 273
Limitations . . . 273
Assumptions . . . 274
Error Codes . . . 274
Methods . . . 275
Events . . . 275
Event Details . . . 275
Properties . . . 276
Property Details . . . 278
DisXfr and ODisXfr . . . 281
Limitations . . . 281
Assumptions . . . 281
Error Codes . . . 281
Events . . . 283
Event Details . . . 284
Properties . . . 286
Property Details . . . 287

Glossary . . . 290

Bibliography . . . 292
Referenced Publications . . . 292
IDS Publications . . . 292

Index . . . 294

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
555 Bailey Avenue, W92/H3
P.O. Box 49023
San Jose, CA 95161–9023

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of IBM Corporation in the United States or other countries or both:

ACF/VTAM	DXT	PROFS
AIX	Enterprise System/9000	PS/2
Application System/400	ES/9000	QMF
APPN	IBM	RACF
AS/400	MVS/ESA	RISC System/6000
Capsule	MVS/XA	RS/6000
DATABASE 2	OfficeVision	S/390

© Copyright IBM Corp. 1996

DataJoiner	OfficeVision/VM	SQL/DS
DB2	Operating System/2	System/360
DB2 for OS/2	Operating System/400	System/370
DB2/6000	OS/2	Virtual Machine/Enterprise Systems Architecture
DFSORT	OS/400	VM/ESA
Distributed Relational Database Architecture	Personal System/2	VTAM
DRDA	Presentation Manager	Win-OS/2

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About This Book

This book describes how to build applications on personal computers for access to IDS. This book includes information on:

- how Open Data Access Service (ODAS), Capsule Services, and Conversion Services work together to provide access to IDS from LAN-based or remote personal computers.
- reference descriptions of API functions for construction of your application for the C or Visual Basic programing languages.
- task-oriented descriptions of how to make your application perform various actions using the API calls.

The tasks described in this manual are primarily concerned with the ODAS and Capsule Services products, running on with on the following platforms:

- OS/2 2.1 or later
- AIX
- Microsoft Windows Version 3.1
- Windows NT
- Windows 95
- UNIX Solaris 2

Who Should Read This Publication

This book is directed at programmers who want to use C or Visual Basic programming languages to develop applications for use with IDS Open Data Access Service. To use the information in this book, readers must be have the following skills.

- You must be skilled in C or Visual Basic programming.
- You must be skilled in programming techniques associated with the IDS Open Data Access Service Application Programming Interface (ODAS API).

Prerequisite Reading

For additional information about configuring and administering ODAS or installing ODAS open clients, see the following books:

- *Installing IDS LAN Components* (GC26-8877)
- *Installing and Configuring IDS Open Clients* (GC66-8889)

© Copyright IBM Corp. 1996

We'd Like Your Comments

Your feedback is important to us in providing you with the most accurate and high-quality information. If you have any comments about this book or any other IDS documentation, please visit our Website at:

http://www.dss.ibm.com

There you'll find the Feedback Connection page where you can enter comments and send them to us.

Chapter 1. Introducing Open Data Access Service

This chapter describes the Open Data Access Service (ODAS) product and lists the major duties of developing applications for it. Such applications provide access to Intelligent Decision Server (IDS) services from remote or LAN-connected workstation.

Functionality

The components of ODAS, as illustrated in Figure 1 on page 2, are:

- The ODAS server, which resides on the IDS local area network (LAN) and interprets requests from clients, workstations that are not running IDS developer's desktop or service software)
- Capsule Services, which lets users run previously constructed capsule applications
- Conversion services, which convert transferred files to the proper formats
- Network-management programs on the LAN, to help system administrators manage ODAS resources
- Application programs on clients, which are typically in remote locations and communicate with the LAN through asynchronous modems and a variety of other interconnection methods

The application programs can be written in the C or Visual Basic language, using the functions described in this manual. To the end user, such a program can present a simple, push-button user interface for:

- Listing the contents of any desktop container (a desktop, file drawer, envelope, folder or capsule application)
- Copying or moving a desktop object from one container to another
- Deleting or renaming a desktop object
- Determining the existence of a desktop object
- Converting the format of a desktop object
- Exchanging mail or other files between the desktop and the client, with or without translation of text, spreadsheet, and other formats
- Examining or changing capsule variables
- Executing previously assembled capsule applications
- Querying the status of capsule applications
- Cancelling capsule applications that are in progress

This manual provides instruction and examples of how to make these tasks available to the end user.

© Copyright IBM Corp. 1996

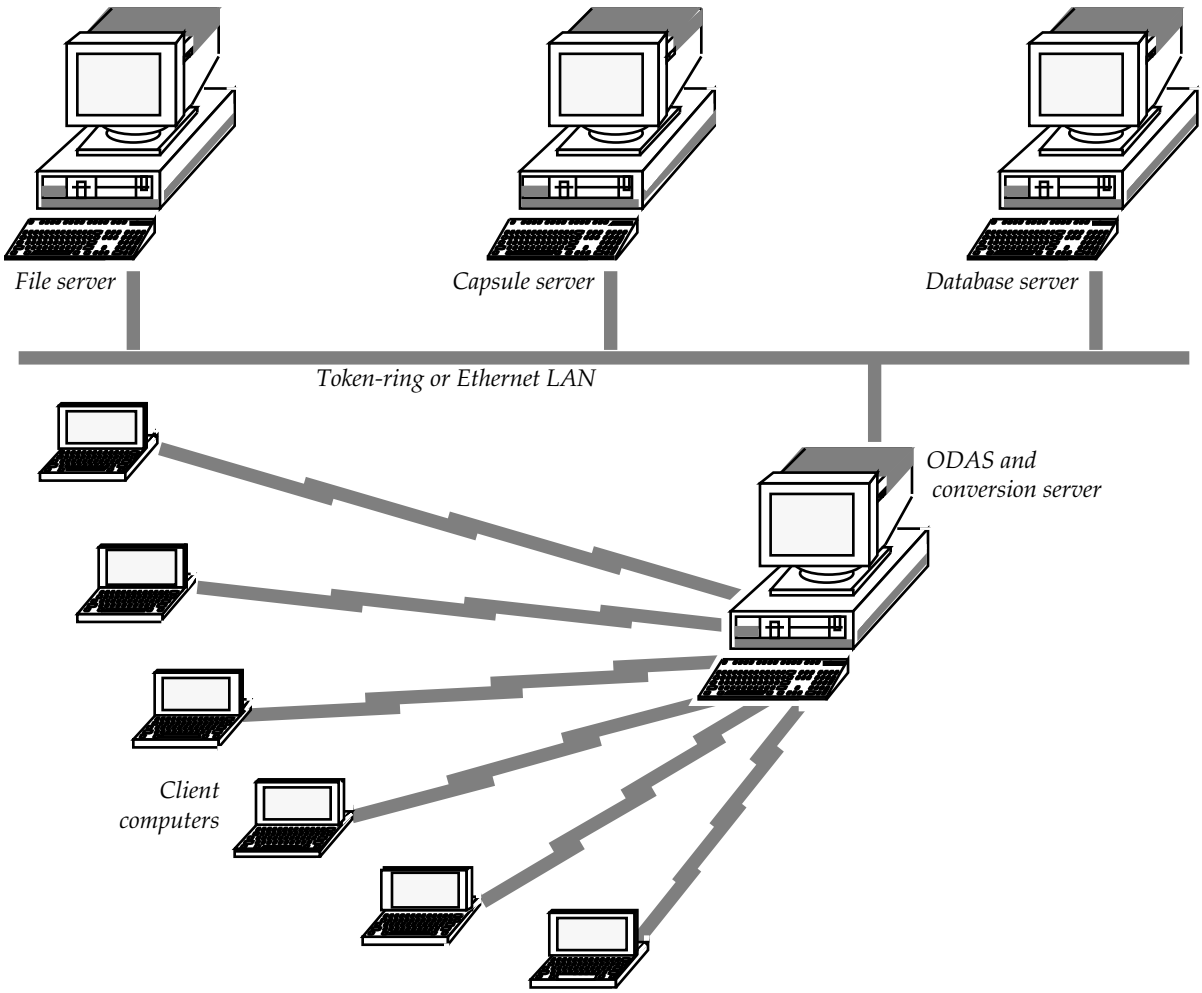


Figure 1. Open Data Access Service components

Connection Methods

You can connect a client with the ODAS server through any of a variety of software products that provide TCP/IP connections compliant with the Windows Sockets** (WinSock) 1.1 specification. With such software, connection is physically accomplished through either modems or direct cabling.

Whichever program you use, you should read its installation instructions carefully before installing it. The *Installing and Administering Open Data Access Service*manual offers some additional help.

Execution of ODAS Requests

ODAS works closely with Capsule Services, conversion services, and other LAN services, as shown in Figure 2 on page 4.

Capsule Services

The Capsule Services product allows controlled execution of capsule applications. Its service manager starts one of the available execution engines for each request from ODAS. That execution engine sees that the necessary tasks are performed, sends results to the ODAS server's session manager, then terminates.

Execution engines are grouped into clusters, each controlled by a cluster manager. Each cluster manager, and all of its execution engines, are in the same workstation. A service manager can control several cluster managers in different workstations, to distribute the work.

The Capsule Services scheduling capabilities let the client user run a capsule application:

- Immediately or as soon as an execution engine is available
- One time, at a specified date and time
- Periodically (for example, monthly, weekly, or daily)
- In batch mode, during a predefined time range (for example, the system administrator could define OVERNIGHT as an alias for some time between midnight and 5 a.m.)

User-defined aliases are available to client users through ODAS, but they are not available to workstation users of the Capsule Services tool.

Capsule Services also lets the user:

- List all jobs in the capsule queue, in the order of execution
- Specify the maximum starting-time delay (how long, beyond the scheduled starting time, a capsule job can wait to run before being canceled automatically by Capsule Services)
- Update the scheduled run time, maximum run time, and starting-time delay of any capsule application submitted by that user
- Place on hold, release from hold, or cancel any capsule job submitted by that user

The system administrator configures and controls Capsule Services through commands in the system administration console.

A set of commands is provided for administration of Capsule Services. These commands let the administrator control logging, queueing, status reporting, and aliases for job-execution times. The IDS *System Administration Guide and Reference* describe how to use these commands.

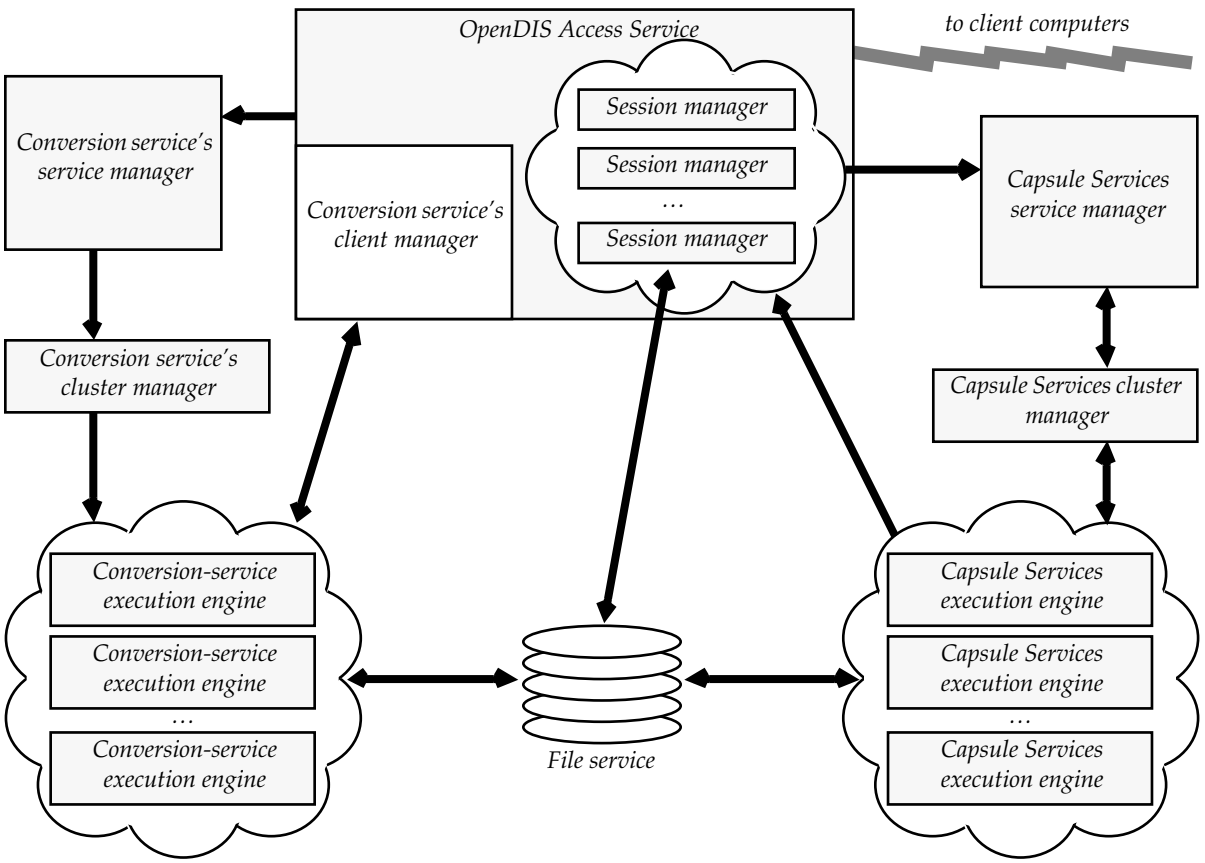


Figure 2. Services associated with ODAS

Conversion Services

Conversion services convert files between IDS formats and personal-computer formats. They work somewhat differently from Capsule Services: the conversion service manager starts a preconfigured number of execution engines, but then ends its involvement. The client manager, in the ODAS server, schedules conversion jobs to be performed by those execution engines. Each execution engine returns results to the client manager, then waits to be assigned a new job.

The System Administration tool (SA tool) provides a command console for administration of conversion services. These commands let the administrator control logging, conversion queueing, and status reporting. The IDS *System Administration Guide and Reference* describe how to use these commands.

ODAS Architecture

ODAS includes a server on the LAN and other components on each client.

ODAS Server

A server coordinates requests from clients and ensures that the appropriate LAN services execute those requests. Figure 3 on page 6 shows the major components of this server.

Administration Console

The administration console gives the ODAS administrator the ability to perform tasks including validating new clients and monitoring server activity. It communicates with the service manager and the network communication manager to provide configuration, monitoring, and other management needs for the client units and all components of the server. The IDS *System Administration Guide and Reference* describes how to use this console.

Session Managers

Whenever a client requests connection to the server, the service manager creates a session manager to interpret the client's requests and store information related to that specific session. Each session manager accepts requests from the corresponding client unit, interacts with the other IDS services to carry out those requests, and gathers the results to be returned to the client unit.

Service Manager

The center of a ODAS server is the service manager, which starts and stops session managers on behalf of the clients, distributes client requests to the appropriate sessions, gathers results for return to the clients, and directs all other communications within the server.

Logging Module

The logging module keeps a record of the activities of the service manager and the client manager, and makes that information available through the administration console. It tracks session events (connection, IDSconnection, and logging on and off), request events (such as file transfers, capsule execution, and other remote desktop activities), and server events (startup and shutdown).

Client Manager

The client manager handles the interface between the network communication manager and the rest of the server.

Network Communication Manager

The network communication manager interacts directly with the client's communication manager to transmit and receive data, commands, and other information. Within the server, it communicates with the client manager only.

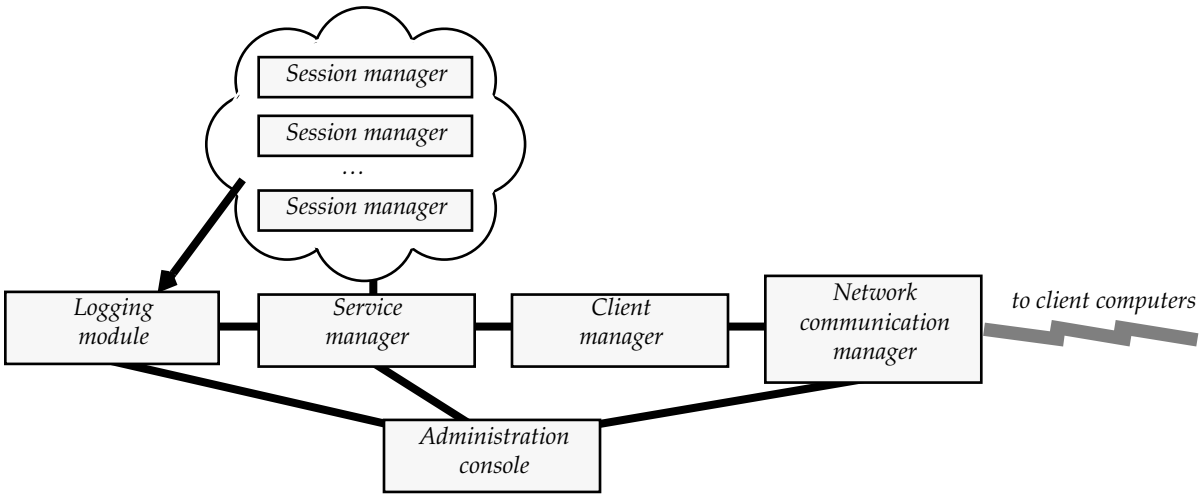


Figure 3. Architecture for the server

Client Workstation

To run an ODAS application program, a client must run one of these operating systems:

OS/2 2.1 or later

IBM DOS or MS-DOS, Version 3.3 or later (This manual uses the term “DOS” to mean any such operating system.)

Microsoft Windows Version 3.0 or later, with or without Microsoft Visual Basic Version 3.0 or later

SunOs V4.1.3 or Sun Solaris V2.4

AIX

The workstation must also be validated by the ODAS server. The system administrator is responsible for generating a diskette that contains this validation and the ODAS software for the client.

Figure 4 on page 7 shows the major ODAS components for a workstation.

Communication Manager

The communication manager interacts directly with the server's network communication manager to transmit and receive data, commands, and other information. Within the workstation, it translates application programming interface (API) functions and parameters into an internal format for transmission to the server, and translates the results into instructions and data for the API to act upon.

Application Programming Interface

The API module interprets the function calls from your application program, then sends the corresponding requests to the communication manager for transmission to the

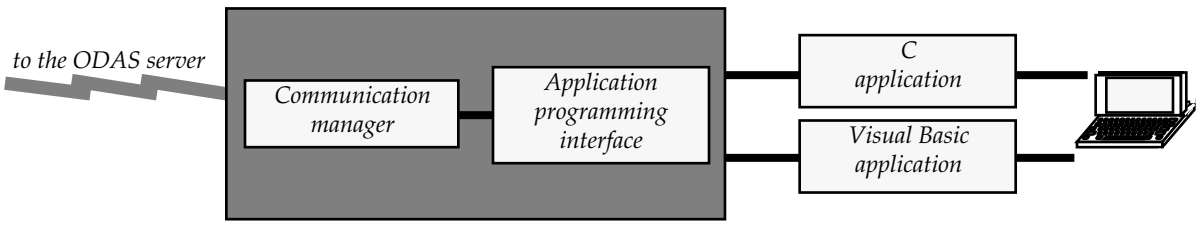


Figure 4. Architecture for the client

server. It also receives files and server information from the communication manager; it stores any fetched files and passes other information back to the application program.

C or Visual Basic Applications

Custom application programs, written in the C or Visual Basic language, include functions that enable many ODAS tasks, such as IDS developer's desktop operations or file transfers. On the IDS Installation CD-ROM, you have been provided with a wide range of tools, including sample code, Custom Visual Basic Controls, and other applications to help you in programming applications to work with IDS

Chapter 2. Programming ODAS Clients

This chapter describes:

- Locating header files for C applications.
- Compiling C applications.
- Software required to develop Visual Basic applications.
- How to identify IDS developers desktop objects.
- Using flags and file types.

Locating Header Files Required for C Applications

In the TOOLKIT directory of the ODAS server, find the following subdirectories for the Windows toolkit files:

- INCLUDE, which contains Windows C header files
- LIB, which contains Windows C files for importing
- VB, which contains Visual Basic header files

The OS/2 and UNIX toolkit files are only provided on diskettes. The UNIX support includes SunOs V4.1.3 and Sun Solaris V2.4.

Compiling C Applications

You can use any C compiler that supports the calling convention of the platform (for example, Windows, OS/2, SunOs, AIX).

Including Preprocessor Statements

ODAS provides header files for you to include in applications you write in C or C++. At the beginning of any source-code file that uses ODAS functions, data structures, or symbolic constants, include the header file disapi.h, as in this example:

```
#include <disapi.h>
```

To define the symbolic error-code constants for use by your application, include a #define statement for INCL_DIS_ERRORS before the #include statement, as in this example:

```
#define INCL_DIS_ERRORS
#include <disapi.h>
```

Each error-code constant has the form ERROR_MET_*description*, as in ERROR_MET_FILE_NOT_FOUND. The constant ERROR_MET_BASE defines the starting point of the error-code range.

© Copyright IBM Corp. 1996

Be sure to add the directory that contains the ODAS header files to the client's INCLUDE environment variable. For example:

```
SET INCLUDE=D:\C6  \INCLUDE;E:\GAPC\INCLUDE
```

Compiling and Linking

For Windows applications, the ODAS client library for Windows must be run in a protected mode; that is, Windows must be running in standard or enhanced mode.

The ODAS header files contain definitions for C or C++ compilers. However, any compiler can call the API functions if:

The code includes definitions for the necessary functions and data structures.

The compiler supports the system's calling convention

Using a typical C programming large memory model simplifies your programming.

Be sure to define a sufficiently large stack for any thread that calls ODAS function. Some ODAS functions can require as many as 2000 bytes of stack space.

For Windows 3.1

In the Windows environment, the import library is METRAPIW.LIB and the DLL is METRAPIW.DLL. The import library contains the bindings necessary to resolve the ODAS function calls; you must link it with the application. Execution of the ODAS application requires thelibraries. For example, the following command compiles and links a Windows application defined in the source file SALES.C:

dynamic-link

```
C:\> CL AL /W3 /F  X1    SALES.C METRAPIW.LIB
```

For OS/2

The import library is METRAPI2.LIBand the DLL is METRAPI2.DLL. For example, to compile and link sales.c on this platform:

```
C:\> CL icc /W3 /Ge+ SALES.C METRAPI2.LIB
```

For SunOs V4.1.3

The import library is libmetrapisun4.so.2.1. For example, to compile and link sales.c on this platform:

```
$ cc -DUNIX -o sales sales.c -lmetrapisun4
```

For Sun Solaris V2.4

The import library is libmetrapisun5.so.2.1. For example to compile and link sales.c on this platform:

```
$ cc -DUNIX -o sales sales.c -lmetrapisun
```

For AIX

The import library is libmetrapiaix.a. For example, to compile and link sales.c on this platform:

```
$ cc -DUNIX -o sales sales.c -lmetrapiaix
```

Software for Visual Basic Applications

If you want to use the Visual Basic API to write a ODAS application, use the Microsoft Visual Basic product, version 2.0 or later, for compatibility with the API libraries.

ODAS provides header files for you to include in applications you write in Visual Basic. You need to import three files:

METRERR.BAS, which defines the symbolic error-code constants

DISAPI.BAS, which defines the regular ODAS API functions

DISVB.BAS, which defines the five special ODAS API functions

To import these files use the Visual Basic project application:

1. In the menu bar, pull down the **File** menu and select **Load Text**.
2. In the Load Text window, find and select the directory containing the header files, then select the file **METRERR.BAS** and click on **New**.
3. Select the file DISAPI.BAS and click on **New**.
4. Select the file DISVB.BAS and click on **New**.

..

Identifying Desktop Objects

Several of the ODAS functions require that you specify a container or other object that is on the IDS desktop.

Containers

If you want to perform several actions in a specific container or on the desktop, you might want to make it the current container. See “DisSetCurrentContainer” on page 77 or “DisSetCurrentContainer” on page 164. Once you do this, subsequent functions apply to that container, unless you explicitly specify otherwise or set a new current container.

The syntax for specifying a name and location is:

*container**container*\...*name*

This discussion assumes that you have not used DisSetSessionInfoto alter the separator.

You can nest container specifications as deeply as you need, separating the container names with the backslash character (\) as you would for nested DOS directories. For example, Sales\Northeast specifies a container named Northeast that is contained within another container named Sales.

To specify a container that is at the top level of the current container, use the lower-level container name without the backslash character. For example, Sales specifies a container that is in the current container.

To specify a container that is at the top level of the desktop, use a backslash character (\ ) with the container name. For example, \Sales specifies a container named Sales that is not contained within any other container (except the desktop).

To specify the top level of the desktop as a container, use a backslash character by itself. If you have not used the DisSetCurrentContainer function, the desktop is the current container by default.

Objects with Non-Unique Names

A desktop can have more than one object with the same name. To indicate a specific object, you can use the DisQueryObjectInfo function to return information on all objects with that name, including a unique identifier for each such object.

Let the user choose one object from this list, and note its unique identifier. Then, in any function parameter where you would specify an object name (such as DisGetObject), specify instead MET@number. The function parameter then applies to only that object, ignoring any others of the same name.

In C or Visual Basic, specify MET@*number* in place of the object's name (*objectName*), where number is a string created from the hexadecimal representation of the unique object identifier.

To generate this representation in C, you could use the statement:

sprintf (*pszObjectName*, 'MET@%08LX', *number*)

where *number* is the ID member of the DISOBJECTINFO structure. To generate this representation in Visual Basic, you can use the statement:

objectName = 'MET@' & Hex$(*number*)

For example, suppose your Visual Basic program finds that the identifier for the third envelope in the Mail Tray (smith:HQ / Hello!) is &H10D4578C. Instead of referring to the envelope as \Mail Tray\smith:HQ / Hello!, you can specify \Mail Tray\MET@10D4578C. This path uniquely identifies that envelope, even if there are many envelopes in the Mail Tray named smith:HQ / Hello!

Wildcard Characters

In some functions, where you specify an object on the IDS developers desktop or a file on the client, the string can include wild-card asterisks or question marks.

An asterisk matches any string. For example, if you specify June* for a desktop object, the function applies to all objects whose names begin with June. If you specified June*ts, the function would apply to objects named June Accounts or June Reports, because both of them begin with June and end with ts. The function would not apply to objects named July Accounts or June Report.

Flags and File Types

When calling various functions in Visual Basic or C, you use predefined constants for client file types and IDS desktop object types, and with flags to specify desired actions.

Defined Client File Types

Table 1 lists the defined types of files on the workstation, as used in the functions DisGetMultObjects, DisGetObject, DisSetObject, and DisSetConversionTable.

Table 1. File types on the workstation

Description	File-type constant
Default file format	DIS_FILETYPE_DEFAULT
Unformatted ASCII text with no column spacing, paragraph formatting, or tabs	DIS_FILETYPE_TEXT
Unformatted ASCII text with spaces simulating the column spacing, paragraph formatting, and tabs of the desktop object. The default line length is 77 characters, but can be as long as 160 characters	DIS_FILETYPE_SHOWTEXT
Spreadsheet formats used by Lotus 1-2-3	DIS_FILETYPE_WK1 and DIS_FILETYPE_WKS
Data Interchange Format (DIF), a format for transferring data between spreadsheets	DIS_FILETYPE_DIF
Revisable Format Text (RFT)	DIS_FILETYPE_RFT
Rich Text Format (RTF)	DIS_FILETYPE_RTF
Windows bitmap (graphics)	DIS_FILETYPE_WINBMP
OS/2 bitmap (graphics)	DIS_FILETYPE_OS2BMP
PC Paintbrush format (PCX graphics)	DIS_FILETYPE_PCX
Tag Image File Format (TIFF graphics)	DIS_FILETYPE_TIFF
Binary (no conversion)	DIS_FILETYPE_BINARY
IDSformat (no conversion)	DIS_FILETYPE_DIS

Defined Desktop Object Types

Table 2 lists the defined types of objects on the IDS desktop, as used in the functions DisCopyObject, DisGetMultObjects, DisPutObject, DisSetConversionTable, and DisQueryObjectInfo.

Table 2 (Page 1 of 2). Desktop object types

Description	Object-type constant
Default object format	DIS_OBJTYPE_DEFAULT
Unknown object format	DIS_OBJTYPE_UNKNOWN
Reserved system object	DIS_OBJTYPE_SYSTEM
3270 SNA tool for PS/2	DIS_OBJTYPE_3270_SNA_PS2
3270 SNA tool for old workstations	DIS_OBJTYPE_3270_SNA
3270 bisynchronous tool	DIS_OBJTYPE_3270_BSC
Basic tool	DIS_OBJTYPE_BASIC
Browser tool	DIS_OBJTYPE_BROWSER
Calculator tool	DIS_OBJTYPE_CALCULATOR
Capsule Service tool	DIS_OBJTYPE_CAPSULE_SERVICES
Capsule tool	DIS_OBJTYPE_CAPSULE
Conversion table	DIS_OBJTYPE_CONVERSION_TABLE
DBA tool	DIS_OBJTYPE_DBA
DCA Translator	DIS_OBJTYPE_DCA_TRANSLATOR
Data Entry tool	DIS_OBJTYPE_DATAENTRY
Desktop	DIS_OBJTYPE_DESKTOP
Desktop colors icon	DIS_OBJTYPE_DESKTOP_COLORS
Dictionary	DIS_OBJTYPE_DICTIONARY
Discard folder	DIS_OBJTYPE_DISCARD
Disk on system desktop	DIS_OBJTYPE_DISK
Drawer	DIS_OBJTYPE_DRAWER
Envelope	DIS_OBJTYPE_ENVELOPE
Folder	DIS_OBJTYPE_FOLDER
Format tool	DIS_OBJTYPE_FORMAT
Forms Design tool	DIS_OBJTYPE_FORMS_DESIGN
Help tool	DIS_OBJTYPE_HELP
In transformer	DIS_OBJTYPE_IN_CONNECTOR
Layout tool	DIS_OBJTYPE_LAYOUT
Log icon	DIS_OBJTYPE_LOG
Mail Tray	DIS_OBJTYPE_MAILTRAY
OS/2 application	DIS_OBJTYPE_OS2_APPLICATION

Table 2 (Page 2 of 2). Desktop object types

Description	Object-type constant
OS/2 clipboard	DIS_OBJTYPE_OS2_CLIPBOARD
Option sheet design	DIS_OBJTYPE_OPTION_SHEET_DESIGN
Out transformer	DIS_OBJTYPE_OUT_CONNECTOR
PC BIN icon	DIS_OBJTYPE_PC_BIN
PC DIF icon	DIS_OBJTYPE_PC_DIF
PC Directory tool	DIS_OBJTYPE_PC_DIR
PC GFX icon	DIS_OBJTYPE_PC_GFX
PC RFT icon	DIS_OBJTYPE_PC_RFT
PC RTF icon	DIS_OBJTYPE_PC_RTF
PC TXT icon	DIS_OBJTYPE_PC_TXT
PC WKS icon	DIS_OBJTYPE_PC_WKS
PC folder (within PC Directory tool)	DIS_OBJTYPE_PC_FOLDER
PIF Formatter	DIS_OBJTYPE_PIF_FORMATTER
PIF data	DIS_OBJTYPE_PIF_DATA
Plot tool	DIS_OBJTYPE_PLOT
Printer	DIS_OBJTYPE_PRINTER
Query tool	DIS_OBJTYPE_QUERY
Reference capsule	DIS_OBJTYPE_REF_CAPSULE
Reference folder	DIS_OBJTYPE_REF_FOLDER
Report Builder tool	DIS_OBJTYPE_REPORTER2
Report tool	DIS_OBJTYPE_REPORT
Reporter tool	DIS_OBJTYPE_REPORTER
SA tool	DIS_OBJTYPE_SYSADMIN
Screen Capture tool	DIS_OBJTYPE_SCREEN_CAPTURE
Spreadsheet tool	DIS_OBJTYPE_SPREADSHEET2
SQL Entry tool	DIS_OBJTYPE_SQL
Spreadsheet Translator	DIS_OBJTYPE_SS_TRANSLATOR
Spreadsheet tool	DIS_OBJTYPE_SPREADSHEET
TTY/2622 tool	DIS_OBJTYPE_HP2622
Text document	DIS_OBJTYPE_DOCUMENT
Transformer tool	DIS_OBJTYPE_TRANSFORMER

Common Bit Flags

Table 3 on page 15 lists the common bit flags used in the fFlags parameter of each API function.

Table 3. Common bit flags

Flag name	Hexadecimal value	Meaning
DIS_WAIT	0000	Wait for the request to complete.
DIS_PRESERVE_TIME	0080	When copying a file, keep the timestamp of when it was last modified.
DIS_DELETE_IFEMPTY	0200	After the operation, delete only empty containers and noncontainers.
DIS_OVERWRITE	0400	Overwrite the target.
DIS_DISCARD_OUTPUTS	0800	Do not return output parameters.
DIS_IGNORE_ERROR	1000	Continue batch requests on error.
DIS_DELETE_EXPUNGE	2000	Bypass the normal Discard folder.
DIS_DELETE_SOURCE	4000	Delete sources after the operation.
DIS_NOWAIT	8000	Execute the request asynchronously.

To use two or more flags together, use the Or (in Visual Basic) or | (in C) operator. For example:

```
fFlags = DIS_DELETE_EXPUNGE Or DIS_NOWAIT
```

DIS_NOWAIT

By default, the API functions execute synchronously. A function does not return until the request is complete. If you specify the DIS_NOWAIT flag, however, the function returns as soon as the request is sent to the ODAS server.

You can call DisQueryResultto determine whether a request is executed and, if so, what the completion code was. You can use DIS_NOWAIT to make the function return immediately and then use repeat calls to DisQueryResult to determine when the original function call completes.

DIS_DELETE_IFEMPTY

The DIS_DELETE_SOURCE flag specifies that the desktop objects or workstation files are to be deleted after the current request is executed. For example, if you specify this flag in a DisGetObject call, the function transfers the object to the specified file on the workstation and then deletes the source object from the desktop. If you call DisGetObject without specifying DIS_DELETE_SOURCE, the source object remains on the desktop. If the deleted object is a container, such as an envelope, DIS_DELETE_SOURCE deletes the container and its enclosures.

The DIS_DELETE_IFEMPTY flag is the same as DIS_DELETE_SOURCE, except that it does not delete any container that is not empty. DIS_DELETE_IFEMPTY deletes objects that are not containers or workstation files exactly as DIS_DELETE_SOURCE does.

If you do not specify DIS_DELETE_EXPUNGE,you can still retrieve a deleted desktop object from the Discardfolder.

DIS_DELETE_EXPUNGE
The DIS_DELETE_EXPUNGE flag specifies that desktop objects are to be immediately deleted instead of being moved to the Discard container.

DIS_IGNORE_ERROR
The DIS_IGNORE_ERROR flag specifies that execution of a batch request should continue even if an error occurs. You can specify this flag in individual function calls within a batch (between DisInitBatch and DisSubmitBatch), so the flag applies to the individual call only; or, you can specify it on the DisSubmitBatch call, to apply to all calls within the batch.

DIS_DISCARD_OUTPUTS
The DIS_DISCARD_OUTPUTS flag specifies that the output parameters of the function are not to be returned. This flag is typically used with the DIS_NOWAIT flag, so the caller does not have to provide output variables that can remain usable until completion of the asynchronous request.

If you omit the DIS_DISCARD_OUTPUTSflag while specifying DIS_NOWAITor submitting batch commands, be sure that any output buffers passed to the function remain valid until the function is completed. One way to ensure this continued validity is to declare the buffers as static or global, rather than automatic (stack-based), variables. You can also use the DisQueryResultfunction to check for completion.

DIS_OVERWRITE
The DIS_OVERWRITE flag, in a call to DisCopyObject, DisMoveObject, DisGetObject, DisGetMultObjects, DisPutObject, or DisPutMultObjects, specifies that the target file or object is to be overwritten. You can also use it with DisSetConversionTable to replace the current conversion table instead of appending the new entries. If you use it with DisSetCapsuleVars, all of the capsule's variables are overwritten; otherwise, the supplied variable values are merged with the existing values.

DIS_PRESERVE_TIME
The DIS_PRESERVE_TIME flag, in a call to DisCopyObject, DisMoveObject, DisGetObject, DisGetMultObjects, DisPutObject, or DisPutMultObjects, specifies that the copying function is to keep the timestamp of when the file was last modified.

Capsule Job Parameter Flags
Table 4 and Table 5 on page 17 list bit flags used in the DisSubmitCapsule and DisSetCapsuleStatus functions. These flags affect the fbFlags and flMask members of DISCAPSULEJOBPARMS structures.

Table 4 (Page 1 of 2). Capsule job parameter flags for flMask

File type	Hexadecimal value	Description
DIS_CJP_PRIORITY	0001	The *bPriority* parameter is specified.

Table 4 (Page 2 of 2). Capsule job parameter flags for flMask

File type	Hexadecimal value	Description
DIS_CJP_DESCRIPTION	0002	The *pszDescription* parameter is specified.
DIS_CJP_ALIASNAME	0004	The *pszAliasName* parameter is specified.
DIS_CJP_SCHEDULED	0008	The *timeScheduled* parameter is specified.
DIS_CJP_MAXSTARTDELAY	0010	The *secsMaxStartDelay* parameter is specified.
DIS_CJP_MAXEXECUTION	0020	The *secsMaxExecution* parameter is specified.

Table 5. Capsule job parameter flags for fbFlags

File type	Hexadecimal value	Description
DIS_CJP_HOLD	01	Delay the job until further notice.
DIS_CJP_TIME_RELATIVE	02	Specified times are relative to now.

Chapter 3. Using the C Language API Functions

To provide the end user with the most robust access to the Intelligent Decision Server, your application program must perform a variety of tasks. This chapter shows how to use the functions described in Chapter 2, “Programming ODAS Clients” on page 8.

The code fragments in this chapter are abridged; Appendix B, “Program Code with C-Language API Functions” on page 188, shows a short sample application program.

Starting ODAS Application

Before calling any of the ODAS C functions, you must start the TCP/IP communications software, as described in the TCP/IP documentation. You can either require the user to start this software interactively or automatically with a command in your application.

Preventing Command Clashes

In a Windows application, if the user clicks a button before the previous activity finishes, the second ODAS call can interfere with the first one, due to the asynchronous Windows message system. One way to prevent this is to disable all the buttons that call ODAS functions until the currently active function returns its results.

Connecting to a Desktop

To connect to a desktop, use the DisInitClient, DisConnect, and DisLogon functions. Your program needs to:

Call the DisInitClient function to initialize the end user's computer

Call the DisConnect function, specifying the name of the ODAS server

Call the DisLogon function to access an IDS desktop

The following code samples demonstrate how to connect to a desktop. In this example, DISLANCOMM is a structure type determined by DIS_COMM_LAN, as described in “DisConnect” on page 42.

1. Declare the DISLANCOMM parameters structure and any other necessary variables.

```
INT                result;
DISLANCOMM         lan;
HDIS               sessionHandle;
```

2. Initialize the client computer on which the application is to run.

```
result = DisInitClient( );
```

3. In the parameters structure, set the size of the structure and, optionally, the desktop name and password. This example assumes most of these parameters from the default values that are set on the validation diskette by the server's administrator.

© Copyright IBM Corp. 1996

```
memset(&lan,  , sizeof (lan));
lan.cb = sizeof (lan);
lan.pszServerName = myOdasServerName;
```

4. Call DisConnect to connect to the ODAS server.

```
result = DisConnect(DIS_COMM_LAN, &lan,  ,
    &sessionHandle);
```

5. Call DisLogon to access the desktop.

```
result = DisLogon(pszDesktop, pszPassword,  ,
    sessionHandle);
```

Disconnecting from a Desktop

To end a desktop connection, use the DisLogoff, DisDisconnect, and DisTermClient functions. Your program needs to:

Log off the desktop

End the connection to the LAN

Release any resources

The following code samples demonstrate how to terminate access to a desktop:

1. Call DisLogoff, to leave the desktop.

```
result = DisLogoff( , sessionHandle);
```

2. Call DisDisconnect to end the connection to the network.

```
result = DisDisconnect( , sessionHandle);
```

3. Call DisTermClient to clean up loose ends.

```
DisTermClient( );
```

Viewing the Mail Tray

One use of ODAS is to let users exchange mail. Your program should:

Display the list of mail items to let the user choose which items to download

For each selected mail item, download that item and any enclosures it might have

Listing the Mail

To check the contents of the Mail Tray, use the DisQueryObjectInfo function. For each mail item, your program should call the DisQueryObjectInfo function to retrieve information such as the object's name, timestamp, and size, and show that information to the user.

The following code samples demonstrate how to list the contents of the Mail Tray. In this example, pObjBuffer is a pointer to a structure of the type DISOBJECTINFO, as described in “DisQueryObjectInfo” on page 67.

1. Declare the DISOBJECTINFO parameters structure and other necessary variables.

```
INT                 result;
UINT                 i, iStart, cEntries, cbBuf;
PVOID                pBuffer;
PDISOBJECTINFO       pObjBuffer;
```

2. Allocate a buffer to receive the list of mail items. This list shows the same information that a workstation user can see in the Mail Tray container.

```
pBuffer = malloc(BUFSIZE);
/  Allocate a buffer  /
iStart =  ;              /  Initialize the starting index  /
```

3. Cycle through the contents of the Mail Tray. In each iteration, call DisQueryObjectInfo to retrieve standard information about as many envelopes as will fit in the buffer, then display the number of enclosures and title of each envelope. Finally, break the loop when DisQueryObjectInfo sets the starting index (iStart) to zero, indicating that the program has reached the end of the container.

   When the following example checks for error conditions, note that BUFFER_TOO_SMALL is "normal"; it indicates that there are more entries to retrieve. FILE_NOT_FOUND means there are no more entries, and is treated as an error only if it occurs the first time through the loop.

```
for (;;) {
    cEntries = BUFCOUNT;
                        /  Ask for a large number of items  /
    cbBuf = BUFSIZE;        /  Initialize the buffer size  /
    result = DisQueryObjectInfo
              ('\\Mail Tray\\ ',     /  Match all items.  /
               DIS_QOIL_STANDARD,
                                    /  Return standard info.  /
               &iStart,              /   starting  index   /
               &cEntries,             /  item count  /
               pBuffer,              /   buffer  pointer   /
               &cbBuf,                 /  buffer size  /
               DIS_QOI_COUNT_CONTENTS,
                                          /  count enclosures  /
               sessionHandle);

    /  Check for error.  /

    if (result != ERROR_MET_BUFFER_TOO_SMALL) {
        if ((result == ERROR_MET_FILE_NOT_FOUND)
                && !fHeader) {
            break;
        } else if (!GAOK(result, 'DisQueryObjectInfo')) {
            break;
        }
    }

    /  Display the column headers only once, in the
        first time through the loop.  /

    if (fHeader) {
        puts('Encl    Name');
        fHeader = FALSE;
    }

    /  Loop through the returned items, displaying the
        number of enclosures and names.  /

    pObjBuffer = (PDISOBJECTINFO)pBuffer;
    for (i =  ; i < cEntries; i++) {
        printf('%3lu%s\n',  pObjBuffer->cbObject,
                pObjBuffer->szName);
        pObjBuffer = PNEXTDISOBJECTINFO(pObjBuffer);
    }
    if (iStart ==  ) {
            /  Starting index reset means no more entries.  /
        break;
    }
}
free(pBuffer);     /  Free the buffer  /
```

Retrieving an Envelope from the Mail Tray

To copy a selected mail item to the remote personal computer, the most important function is DisGetObject.

The following code samples demonstrate how to retrieve the mail items from the desktop to the remote personal computer, where the user can read them.

1. Declare the necessary constants and variables.

```
/ Parameters supplied to this function: /
PSZ         pszEnvelopeName;
                    / name of the envelope to retrieve /
PSZ          pszLocalPath;          / target file on the client
                      computer,such as '\mail\env1.txt') /
BOOL          fDelete;
             / Set TRUE to delete envelope after retrieval /

/ constants and variables for this function /
#define  LINEWIDTH  7  / Format lines to this width. /
#define  PATHLEN      (DIS_OBJECT_NAME_LEN+1 +1)
ULONG       cbFile;                  / file size /
CHAR        szPath[PATHLEN];
                    / Construct the object path here. /
```

2. Construct the Envelope's name as a string.

```
strcpy(szPath, '\\Mail Tray\\');
strcat(szPath, pszEnvelopeName);
```

3. Call DisGetObject to retrieve the Envelope, converting it into TEXT format (unformatted ASCII text).

```
cbFile = ;        / Specify no limit on file size. /
result = DisGetObject(szPath, pszLocalPath,
    DIS_FILETYPE_TEXT, &cbFile, LINEWIDTH,
    fDelete ? DIS_DELETE_SOURCE :  ), sessionHandle);
```

Retrieving Enclosures of an Envelope

To retrieve enclosures, use the DisSetConversionTable and DisGetMultObjects functions.

The following code samples demonstrate how to retrieve the actual mail items from the desktop to the remote personal computer, where the user can read them. In this example, convTableIn is an array of structures of the type DISCONVERTENTRY, as described in “DisSetConversionTable” on page 75.

1. Declare pointers to the DISCONVERTENTRY parameters structure, with other necessary constants and variables.

```
/ Parameters supplied to this function: /
PSZ          pszEnvelopeName;
                    / name of the Envelope to retrieve /
PSZ        pszLocalPath;
                  / basis of target file names on the
                  client computer (such as '\mail\enc1.') /

/ constants and variables for this function /
#define  LINEWIDTH  7  / Format lines to this width. /
#define  PATHLEN      (DIS_OBJECT_NAME_LEN+12+1)
DISCONVERTENTRY convTableIn[2];   / conversion table /
CHAR       szPath[PATHLEN];
                    / Construct the object path here. /
```

2. Call DisSetConversionTable to set the conversion table so it will convert all spreadsheets to Data Interchange Format (DIF) and all text objects to TEXT format.

```
memset(convTableIn,  , sizeof (convTableIn));
convTableIn[ ].bFileType = DIS_FILETYPE_DIF;
convTableIn[ ].bObjType = DIS_OBJTYPE_SPREADSHEET2;
convTableIn[1].bFileType = DIS_FILETYPE_TEXT;
convTableIn[1].bObjType = DIS_OBJTYPE_DOCUMENT;
convTableIn[1].cchLineWidth = LINEWIDTH;
result = DisSetConversionTable(convTableIn, 2,
       DIS_FILETYPE_DEFAULT,  DIS_OVERWRITE,
       sessionHandle);
```

3. Construct a string, using the wildcard (*), to specify all of the Envelope's enclosures.

```
strcpy(szPath, '\\Mail Tray\\');
strcat(szPath, pszEnvelopeName);
strcat(szPath, '\\ ');
```

4. Call DisGetMultObjects to retrieve the enclosures, converting each enclosure as specified in the conversion table.

```
cObjects = 1 ;       / Retrieve as many as 1
      enclosures.  /
result = DisGetMultObjects(szPath, pszLocalPath,
          &cObjects, NULL, NULL,  , sessionHandle);
```

Sending Mail

To send mail to the Mail Tray, use the DisPutMultObjects or DisPutObject function. The code to send mail without enclosures is somewhat simpler than the code for enclosures. In each case, your program should:

Ensure that each mail item exists as a properly formatted text file on the remote personal computer

Copy each message file to the Mail Tray

One Item, without Enclosures

To send one mail item that has no enclosures, use the DisPutObject function. The following code samples demonstrate how to send mail files from the remote personal computer to the desktop Mail Tray.

1. Declare the necessary constants and variables.

```
/ Parameters supplied to this function: /
PSZ    pszLocalFile;    / file name on the client computer
                              (such as '\newmail\env') /
BOOL   fDelete;
          / Set TRUE to delete file after mailing it /
```

2. Call DisPutObject to send the message file to the Mail Tray.

```
result = DisPutObject(pszLocalFile, DIS_FILETYPE_TEXT,
        , '\\Mail Tray\\', DIS_OBJTYPE_ENVELOPE,
       (fDelete ? DIS_DELETE_SOURCE :  ), sessionHandle);
```

Two or More Items, without Enclosures

To send two or more mail items that have no enclosures, use the DisPutMultObjects function. The following code samples demonstrate how to send mail files from the remote personal computer to the desktop Mail Tray. In this example, convTableOut is an array of structures of the type DISCONVERTENTRY, as described in "DisSetConversionTable" on page 75.

1. Declare pointers to the DISCONVERTENTRY parameters structure, with other necessary variables.

```
/ Parameters supplied to this function: /
PSZ    pszLocalSpec;    / basis for file names on the
                    client computer (such as '\newmail\ ') /
BOOL   fDelete;
         / Set TRUE to delete files after mailing them /

/ variables for this function /
DISCONVERTENTRY convTableOut;     / conversion table /
```

2. Call DisSetConversionTable to set the conversion table so it will convert text files to Envelopes.

```
convTableOut.pszFileExt = NULL;
convTableOut.bFileType = DIS_FILETYPE_TEXT;
convTableOut.bObjType = DIS_OBJTYPE_ENVELOPE;
convTableOut.cchLineWidth =  ;
result = DisSetConversionTable(&convTableOut, 1,
        DIS_FILETYPE_TEXT, DIS_OVERWRITE, sessionHandle);
```

3. Call DisPutMultObjects to send the message files to the Mail Tray.

```
result = DisPutMultObjects(pszLocalSpec, '\\Mail
        Tray',
         (fDelete ? DIS_DELETE_SOURCE :  ), sessionHandle);
```

One Item, with Enclosures

To send a mail message and enclose other messages, spreadsheets, or other files in the message, use the DisPutMultObjects function. The following code samples demonstrate how to send a mail file with enclosures from the remote personal computer to the desktop Mail Tray. In this example, convTableOut is an array of structures of the type DISCONVERTENTRY, as described in “DisSetConversionTable” on page 75.

1. Declare pointers to the DISCONVERTENTRY parameters structure, with other necessary variables.

```
/ Parameters supplied to this function: /
PSZ     pszLocalFile;   / file name of the message file on
                the client computer (such as '\newmail\env') /
PSZ    pszEnclSpec;    / basis for names of the enclosure
                            files (such as '\newmail\encl') /
BOOL    fDelete;            / Set TRUE to delete files after
                              mailing them /

/ variables for this function /
DISCONVERTENTRY convTableOut;     / conversion table /
PSZ pszEnvelope = '\\New PC Mail';
```

2. Call DisPutObject to send the message file to the developer's desktop.

```
result = DisPutObject(pszLocalFile, DIS_FILETYPE_TEXT,
          , pszEnvelope, DIS_OBJTYPE_ENVELOPE,
         (fDelete ? DIS_DELETE_SOURCE :  ), sessionHandle);
```

3. Call DisSetConversionTable to set the conversion table so it will convert text files to Envelopes.

```
convTableOut.pszFileExt = NULL;
convTableOut.bFileType = DIS_FILETYPE_TEXT;
convTableOut.bObjType = DIS_OBJTYPE_ENVELOPE;
convTableOut.cchLineWidth =  ;
result = DisSetConversionTable(&convTableOut, 1,
         DIS_FILETYPE_TEXT, DIS_OVERWRITE, sessionHandle);
```

4. Call DisPutMultObjects to send the enclosure files to the new Envelope.

```
result = DisPutMultObjects(pszEnclSpec, pszEnvelope,
         (fDelete ? DIS_DELETE_SOURCE :  ), sessionHandle);
```

5. Call DisMoveObject to mail the Envelope.

```
result = DisMoveObject(pszEnvelope, 'Mail Tray\\”,  ,
        sessionHandle);
```

Moving Files to the Desktop

To move one or more files to the LAN, use the DisPutMultObjects or DisPutObject function. Your program should:

Ensure that each file exists, properly formatted, on the client computer

Specify the appropriate conversion

Call DisPutObject (or DisPutMultObjects) to copy the file or files into the container

The following code samples demonstrate how to copy one WKS-format spreadsheet file to a developer's desktop, automatically converting it to a Spreadsheet II object.

1. Declare the necessary constants and variables.

```
/ Parameters supplied to this function: /
PSZ     pszLocalFile;          / file name of the spreadsheet
      on the client computer (such as '\data\budget.wks') /
PSZ     pszContainer;          / desktop container path /
PSZ     pszName;                 / name of icon on desktop /
BOOL    fDelete;                 / Set TRUE to delete file after
                                     sending it /

#define PATHLEN    ((DIS_OBJECT_NAME_LEN 2)+1)
CHAR   szPath[PATHLEN];
```

2. Construct the target path to specify the desired desktop container.

```
szPath[ ] = '\\';
if (pszContainer == NULL) {
    szPath[1] =  ;
} else {
   strcpy(szPath+1,  pszContainer);
}
strcat(szPath, pszName);
```

3. Call DisPutObject to send the local file to the developer's desktop.

```
result = DisPutObject(pszLocalFile,
       DIS_FILETYPE_WKS,   ,
         szPath,  DIS_OBJTYPE_SPREADSHEET2,
          (fDelete ? DIS_DELETE_SOURCE :  ), sessionHandle);
```

Retrieving Files from the Desktop

To retrieve one or more files from the LAN, use the DisGetMultObjects or DisGetObject function. Your program should:

Specify the appropriate conversion

Call the DisGetObject (or DisGetMultObjects) function to copy the file or files into the container

The following code samples demonstrate how to copy one Spreadsheet II object from a developer's desktop, automatically converting it to a WKS-format spreadsheet file.

1. Declare the necessary constants and variables.

```
/ Parameters supplied to this function: /
PSZ     pszObjectName;
                / name of the desktop object to retrieve /
PSZ     pszLocalPath;
                        / target file on the client computer
                          (such as '\data\budget.wks') /
BOOL    fDelete;
              / Set TRUE to delete object after retrieval /

/ constants and variables for this function /
#define   PATHLEN     (DIS_OBJECT_NAME_LEN+1 +1)
ULONG       cbFile;                     / file size /
CHAR        szPath[PATHLEN];
                      / Construct the object path here. /
```

2. Call DisGetObject to retrieve the spreadsheet, converting it into DIF format.

```
cbFile =  ;             / Specify no limit on file
      size.  /
result = DisGetObject(pszObjectName, pszLocalPath,
        DIS_FILETYPE_WKS, &cbFile,  ,
        (fDelete ? DIS_DELETE_SOURCE :  ), sessionHandle);
```

Using Capsule Applications

For effective remote use of Capsule applications, the end user should be able to list the available applications, set @-variables in the selected application, run the application, check its progress, and alter its queue status.

Setting Capsule Variables

To assign values to a Capsule application's @-variables, use the DisSetCapsuleVars function. The following code samples demonstrate how to do this. In this example, aCapsuleVars is an array of structures of the type DISCAPSULEVARINFO, as described in “DisSetCapsuleVars” on page 74.

Although the at-sign (@) is an essential part of the variable name in the normal workstation environment, you cannot use it in the ODAS programming; specify "A" for @A, and so forth.

1. Declare pointers to the DISCAPSULEVARINFO parameters structure, with other necessary variables.

```
/ Parameters supplied to this function: /
PSZ pszCapsuleName;             / Capsule path (such as
                                  '\Budgeting\Q492 Budget') /
PSZ pszDepartment;
                    / value for Department variable (@A) /
PSZ pszLineItem;   / value for LineItem variable (@B) /

/ variables for this function /
DISCAPSULEVARINFO aCapsuleVars[2];
                      / Structure for Capsule variables /
```

2. Initialize the information about Capsule variables.

```
strcpy(aCapsuleVars[ ].szName, 'A');
                                        / Change variable @A. /
aCapsuleVars[ ].ulReserved = ;
                                      / Must be set to zero. /
aCapsuleVars[ ].pszDescription = NULL;
      / Specify no change to the variable's
                                        description. /
aCapsuleVars[ ].pszValue = pszDepartment;
                                        / new value for the @A /
strcpy(aCapsuleVars[1].szName, 'B');
                                        / Change variable @B. /
aCapsuleVars[1].ulReserved = ;
                                      / Must be set to zero. /
aCapsuleVars[1].pszDescription = NULL;
      / again, no change to the variable's description. /
aCapsuleVars[1].pszValue = pszLineItem;
                                        / new value for the @B /
```

3. Call DisSetCapsuleVars to send the new values to the specified Capsule application on the developer's desktop.

```
result = DisSetCapsuleVars(pszCapsuleName,
       aCapsuleVars,
         2, , sessionHandle);
```

Running Capsule Applications

To run an existing Capsule application, use the DisSubmitCapsule function.

The following code samples demonstrate how to run a Capsule application. In this example, JobParms is an array of structures of the type DISCAPSULEJOBPARMS, as described in “DisSetCapsuleVars” on page 74.

1. Declare pointers to the DISCAPSULEJOBPARMS parameters structure, with other necessary variables.

```
/ parameters supplied to this function: /
PSZ pszCapsuleName;
      / Capsule path (such as '\Budgeting\1993 Budget') /
PSZ pszOutFolder;
          / output folder (such as '\Budgeting\Results') /
BOOL fOvernight;              / Set TRUE to run the Capsule
  application at night, or FALSE to run it immediately. /

/ variables for this function /
DISCAPSULEJOBPARMS jobParms, pjobParms;
                                    / to modify job parameters /
ULONG idJob;       / to receive the Capsule job number. /
```

2. Initialize the Capsule job parameters in the DISCAPSULEJOBPARMS structure.

```
if (fOvernight) {
            /  Set the flag for overnight or immediate use.  /
    memset(&jobParms,  , sizeof (jobParms));
    jobParms.cb = sizeof (jobParms);
    jobParms.pszAliasName = 'OVERNIGHT';
    pjobParms = &jobParms;
} else {
    pjobParms = NULL;
}
```

3. If necessary, set the Capsule application's @-variables as described above in “Setting Capsule Variables” on page 27.”
4. Call DisSubmitCapsule to submit the Capsule application to the capsule server.

```
result = DisSubmitCapsule(pszCapsuleName, pszOutFolder,
        pjobParms, &idJob,  , sessionHandle);
```

Checking a Capsule Application's Progress

To check the status of a Capsule application in the queue, use the DisQueryCapsuleInfo function.

The following code samples demonstrate how to check a Capsule application's status. In this example, capsuleJobInfo is a structure of the type DISCAPSULEJOBINFO, as described in “DisQueryCapsuleInfo” on page 64.

1. Declare pointers to the DISCAPSULEJOBINFO parameters structure, with other necessary variables, and get the session handle.

```
DISCAPSULEJOBINFO    capsuleJobInfo;
HDIS                 handle;
long                 completionCode =  ;
```

2. Wait for the completion code for the current Capsule application.

```
result
= DisWaitCapsuleStatus (jobId, DIS_WCS_OUTPUT,
        TIME_OUT_LENGTH, &completionCode, DIS_WAIT,
       handle);
```

3. If the completion code is 0 (zero), the Capsule application is not in the queue. Make sure that it has been run and that it exists in the execution folder.

```
strcpy
(capsuleName, CAPSULE_EXECUTION_FOLDER);
strcat (capsuleName, '\\');
strcat (capsuleName, argv[CAPSULE_NAME]);
if (result = DisQueryObjectInfo (capsuleName,
        DIS_QOIL_TEST, &index, &numb,
        &buffer, &bufSize, DIS_WAIT, handle))
    exit (result – ERROR_MET_BASE);
```

Advanced ODAS Programming

This section describes how to allocate output buffers, process functions asynchronously, and maximize your application's performance.

Allocation of Output Buffers

Many of the structures returned by the query-type function calls contain both fixed-length and variable-length data. An example is the DISCAPSULEJOBINFO structure:

```
typedef struct   _DISCAPSULEJOBINFO_L1 {
    USHORT cb;                    / size of this entry, in bytes /
    BYTE bPriority;               / job priority; specify 1 for
     the lowest priority through 31 for the highest priority /
    BYTE bState;                  / job state /
    ULONG id;                     / job identifier /
    PSZ pszOwner;                 / desktop that submitted the job /
    PSZ pszDescription;           / job's description /
    PSZ pszAliasName;             / scheduler alias name /
    ULONG timeSubmitted;          / time when job was submitted /
    ULONG timeScheduled;          / job's scheduled starting time /
   ULONG  secsMaxStartDelay;
              / maximum delay past scheduled time, in seconds /
   ULONG  secsMaxExecution;
                          / maximum execution time, in seconds /
} DISCAPSULEJOBINFO,   PDISCAPSULEJOBINFO;
```

Much of the job information is returned in the fixed-length structure (such as the job identifier and submission time), but several pieces of information are returned as variable-length data (such as the job owner and description). The fixed-length portion contains pointers to the variable-length data—null-terminated strings in this case.

When allocating space for these data structures, you must allow room for both the fixed-length data and the variable-length data in the buffer you supply to the function. For example, if you supply to DisQueryCapsuleInfo a buffer of size equal to sizeof (DISCAPSULEJOBINFO), no job data is returned because you did not provide any additional space for the variable-length data.

In general, avoid attempting to allocate a buffer of precisely the correct size, but instead allocate a buffer of reasonably large size. For example, if you expect to retrieve no more than 10 Capsule job entries, a 2000-byte buffer would be slightly larger than required.

Object Identifiers

Unlike the Windows, DOS, and OS/2 file systems, the IDS file system permits more than one object of the same name to exist in a container. When you refer to an object by name, ODAS always selects the object that appears first in the container directory. The IDS directory is not sorted, so you cannot depend on which of the like-named objects will be selected. (The IDS workstation sorts the directory before displaying it.)

ODAS lets applications distinguish between several like-named objects by using a specially formatted, unique object identifier. The DisQueryCapsuleInfo function returns this identifier in the ID field of the DISOBJECTINFO structure. (See “DisQueryObjectInfo” on page 67.)

To use the object identifier, substitute a name of the form “MET@number”, where number is a string created from the hexadecimal representation of the unique object identifier. (See “Objects with Non-Unique Names” on page 11.

You must still specify any container navigation required to locate the object. The object identifier name must only appear wherever the actual object name can appear.

Asynchronous Processing

Sometimes you might want to initiate an operation and not wait for the operation to complete. ODAS allows this through use of the DIS_NOWAIT flag, as described in “DIS_NOWAIT” on page 15.When you specify this flag in a function call, the function returns when the request is sent to the server.

Later, your application can use DisQueryResult to query the result of the request. For example:

```
if (!DisInitBatch( , sessionHandle)) {
    /  Copy three spreadsheets and total their return codes.  /
   if  (!DisCopyObject('\\Budgets\\FY93',
              '\\PC Budgets\\FY93.WKS', DIS_OBJTYPE_WKS,
             sessionHandle)  &&
        !DisCopyObject('\\Budgets\\FY92',
              '\\PC Budgets\\FY92.WKS', DIS_OBJTYPE_WKS,  ,
             sessionHandle)  &&
        !DisCopyObject('\\Budgets\\FY91',
              '\\PC Budgets\\FY91.WKS', DIS_OBJTYPE_WKS,  ,
             sessionHandle))
   {
       /  if all three were error-free  /
       if (!DisSubmitBatch(NULL,  , DIS_NOWAIT,
              sessionHandle))  {
           for (;;) {
               if (NothingMoreToDo()) {
                   err = DisQueryResult( ,  , sessionHandle);
                   break;
               }
               GoDoSomethingElse();
               err = DisQueryResult( , DIS_NOWAIT,
                   sessionHandle);
               if (err != ERROR_MET_REQUEST_PENDING) {
                   break;
               }
           }
           if (err ==  ) {
               fSuccess = TRUE;
           }
       }
   }
}
```

You can callDisQueryResult either with or without the DIS_NOWAIT flag. Without DIS_NOWAIT, DisQueryResult waits until the request completes. If you use DisQueryResult on a batch request, the iSubRequest parameter determines whether you want the overall result of the batch request or the result of an individual subrequest. Because the example above specifies a zero iSubRequest, DisQueryResult examines the overall state of the batch request.

You cannot use the DIS_NOWAIT flag on DisInitClient, DisTermClient, DisConnect, or DisDisconnect.

Maximum Application Performance

The most effective way to ensure the highest performance and least response time in ODAS applications is to reduce the number of requests and responses that must pass between the client and the server.

Two methods that can reduce the request-and-response requirements are:

Structure the application logic to require fewer ODAS function calls and resources for the desired operation, include eliminating unnecessary function calls, supplying large buffers when calling query-type functions, and using path names to refer to objects in various containers on the desktop.

Take advantage of the batch-request capability to bundle or piggyback requests.

Unnecessary Functions

The easiest function call to eliminate is DisLogoff. The DisDisconnect and DisLogon functions automatically log off the current desktop if DisLogoff has not been called. Because the typical ODAS application disconnects the client from the server when it has completed its desktop tasks, you can simply use the DisDisconnect function to both log off the desktop and end the ODAS session.

Avoid introducing too much robustness into the application at the expense of good application performance. You might want to call DisQueryObjectInfo to ensure existence of an object before retrieving it, but you can gain better performance by omitting that call and checking the return code from DisGetObject to determine whether the object was present as expected. Alternatively, if your user will have to type the object name, instead of just selecting it from your menu, you might find the overhead of DisQueryObjectInfo worthwhile by providing an up-to-date menu for the user's selection.

Large Buffers

When you call the query-type functions (such as DisQueryObjectInfo or DisQueryCapsuleInfo), you can greatly improve application response time by using larger buffers and retrieving many entries at once, rather than retrieving only one entry per function call.

For example, this method for listing the contents of the Mail Tray performs poorly because it retrieves only one object's information in each DisQueryObjectInfo call:

```
iStart =  ;
for (;;) {
    cEntries = 1;
    cbBuf = sizeof (obji);
    result = DisQueryObjectInfo('Mail Tray\\ ',
            DIS_QOIL_STANDARD,  &iStart,
&cEntries, &obji,
            &cbBuf,  DIS_QOI_COUNT_CONTENTS,
sessionHandle);
    if (result !=  ) {
        break;
    }
   printf('%3lu       %s\n',
obji.cbObject, obji.szName);
    if (iStart ==  ) {
        break;
    }
}
```

A more efficient application uses this method to store many objects' information in a single, large buffer. PDISOBJECTINFO is a structure type described in “DisQueryObjectInfo” on page 67.

```
iStart =  ;
for (;;) {
    cEntries = BUFCOUNT;                          /  == 1    /
    cbBuf = BUFSIZE;                              /  ==  x4    /
    result = DisQueryObjectInfo('Mail Tray\\ ',
             DIS_QOIL_STANDARD, &iStart, &cEntries,
             pBuffer, &cbBuf, DIS_QOI_COUNT_CONTENTS,
            sessionHandle);
    if (result !=   && result !=
            ERROR_MET_BUFFER_TOO_SMALL
) {          break;
    }
    pObjBuffer = (PDISOBJECTINFO)pBuffer;
    for (i =  ; i < cEntries; i++) {
        printf('%3lu       %s\n',  pObjBuffer->cbObject,
                pObjBuffer->szName);
         pObjBuffer = PNEXTDISOBJECTINFO(pObjBuffer);
    }
    if (iStart ==  ) {
        break;
    }
}
```

Path Names

The DisSetCurrentContainer function sets the name of the default container to which unqualified paths are directed. An unqualified path is one that does not begin with the path separator character (which, by default, is the backslash character). For example, if you use DisSetCurrentContainer to set the folder Budgets as the current container, the path FY93 Expenses refers to the object named FY93 Expenses in the Budgets folder; compare this with the path \FY93 Expenses, which refers to the object FY93 Expenses at the desktop level.

You can also use the path separator character to construct paths that navigate through multiple levels of containers. For example, \New Icons*Blank Icons\Folder refers to the blank Folder icon. If the current container is Budgets, then FY92 Budgets\Q4 Expenses refers to the Q4 Expenses object in the FY92 Budgets folder which, in turn, is in the Budgets folder.

As a general rule, avoid use of the DisSetCurrentContainer function and use qualified and unqualified paths to refer to objects in different containers on the desktop. This increases the complexity of the application somewhat, but it improves the application response time, even over an application that bundles DisSetCurrentContainer into a batch request.

Deleting Desktop Objects

ODAS deletes objects in a manner similar to the way the IDS workstation deletes objects. Deleted objects are first moved into the Discard folder, from which they can be retrieved until subsequent deletions push the earlier objects out of the Discard folder, purging them from the desktop. This feature lets users recover inadvertently deleted icons. However, it is typically useful only in interactive applications, in which the user can manipulate objects directly.

If your application is non-interactive, or does not let users directly manipulate icons, you can streamline it by using the DIS_DELETE_EXPUNGE flag on function calls that delete desktop objects. This flag deletes the objects instead of moving them into the Discard folder; hence, the ODAS server and file server have fewer operations to perform.

Multiple-Transfer Functions

Whenever possible, your application should use the multiple-transfer functions (DisGetMultObjects and DisPutMultObjects) to send and receive a set of objects, rather than calling DisGetObject or DisPutObject for each individual object.

ODAS can convert and transfer several objects more quickly with the multiple-transfer functions because:

- Fewer requests and responses pass between the client and server
- The overhead of searching for objects and setting up for conversions is shared among all the object transferals
- If enough Conversion Services engines are available, they can convert several objects simultaneously, thereby reducing the overall conversion time

Batch Requests

The ODAS batch facility lets you bundle several client requests and send them to the server as a unit. This reduces both transmission time between the client and server and the transaction time on the server.

To use the batch facility, enclose the functions between the ODAS functions DisInitBatch and DisSubmitBatch. The DisInitBatch function initiates batch processing. All subsequent ODAS function calls executed by the application are not sent to the server but are queued locally as “subrequests.”

When the application is ready to send the batch request to the server, it executes the DisSubmitBatch function. If the application supplies a suitable array when it calls DisSubmitBatch, ODAS will return the individual results of the batch subrequests in that array.

The following example transfers a mail message to the desktop, adds several enclosures, and moves the resulting envelope to the Mail Tray to deliver the message to the Post Office. The example assumes that the text of the message is in the file \NEWMAIL\ENV, the enclosures are in the directory \NEWMAIL\ENCL, and the conver-

sion table is appropriately configured to perform the desired conversions on the enclosures.

```
INT result;
INT aerrorResults[3];

result = DisInitBatch( , sessionHandle);
if (result ==  ) {
    result = DisPutObject('\\newmail\\env',
DIS_FILETYPE_TEXT,
            '\\NewEnv',  DIS_OBJTYPE_ENVELOPE,
            DIS_OVERWRITE|DIS_DELETE_EXPUNGE,
sessionHandle);
    if (result ==  ) {
        result = DisPutMultObjects('\\newmail\\encl\\ ',
                '\\NewEnv',  , sessionHandle);
        if (result ==  ) {
            result = DisMoveObject('\\NewEnv',
'\\Mail Tray\\',
                    ,  sessionHandle);
        }
    }
    if (result =  ) {
        result = DisSubmitBatch(aerrorResults, 3,  ,
                sessionHandle);
    } else {
        result = DisTermBatch( , sessionHandle);
    }
}
```

Notice the call to DisTermBatch if one of the subrequest calls returns an error code. This function terminates batch processing and discards the remaining, queued subrequests without sending them to the server.

The return code from DisSubmitBatch provides an overall indication of the success or failure of the subrequests. You can examine the elements of the aerrorResults array to obtain the return codes from the individual subrequests.

Whenever possible, an application should combine requests using the batch facility. Another common batch request is to combine DisGetObject with DisQueryObjectInfo to retrieve the text of an envelope and at the same time determine the names, types, and sizes of the envelope's enclosures. Still another is to combine DisSetConversionTable with DisGetMultObjects to configure the appropriate conversions and to convert and retrieve the desired objects.

The DisInitClient, DisTermClient, DisConnect, and DisDisconnect functions cannot be executed as batch subrequests.

Chapter 4. API Functions for C-Language Programming

Each of the application programming interface (API) functions described in this chapter performs a specific action. In most cases, this action is analogous to an action frequently performed by Intelligent Decision Server (IDS) workstation users. For example, the DisLogoff function is equivalent to the workstation user selecting the desktop's Log Off button.

The functions require a library specifically designed for the Windows operating system. “Software for Visual Basic Applications” on page 10 describes how to obtain the appropriate library.

Functional Lists of API Functions

Table 6 through Table 11 on page 39 list the API functions, grouped by their uses. Later in this chapter, the detailed API descriptions are sorted in alphabetical order for easy reference.

Table 6. API functions for session and desktop controlFm Variable:Table Sheet

Function	Action
DisInitClient ()	Initializes the ODAS application.
DisTermClient ()	Releases resources allocated for the ODAS application.
DisConnect ()	Establishes a connection to the ODAS server.
DisDisconnect ()	Terminates the application's connection with the ODAS server.
DisLogoff ()	Logs off a developer's desktop.
DisLogon ()	Logs on a developer's desktop.
DisQueryObjectInfo ()	Checks the type, size, creation time, and other information of one or more desktop objects.
DisSetPassword ()	Changes the password of the connected desktop.
DisSetSessionInfo ()	Changes the separator character to be used in path names on the desktop.

Table 7 (Page 1 of 2). API functions for transfer of electronic mail andfilesFm Variable:Table Sheet

Function	Action
DisGetSysMessage ()	Retrieves a message from the system administrator.
DisGetObject ()	Fetches an object from the developer's desktop to a personal computer.
DisGetMultObjects ()	Fetches objects from the developer's desktop to a personal computer.

© Copyright IBM Corp. 1996

Table 7 (Page 2 of 2). API functions for transfer of electronic mail andfilesFm Variable:Table Sheet

Function	Action
DisPutObject ()	Sends one personal computer file to the developer's desktop.
DisPutMultObjects ()	Sends one or more personal computer files to the developer's desktop.
DisQuerySessionInfo ()	Checks either the current container path or the existence of pending system messages or mail.
DisSetConversionTable ()	Establishes the type-conversion table to be used when copying a personal computer file or desktop object.

Table 8. API functions for manipulating desktop objects

Function	Action
DisCopyObject ()	Duplicates a desktop object, placing the new copy in a specified container.
DisDeleteObject ()	Deletes the specified object from the desktop.
DisMoveObject ()	Relocates a desktop object, renames the object, or performs both actions.
DisSetCurrentContainer ()	Specifies the current (default) desktop container on which a subsequent function is to be performed.

for using

Table 9 (Page 1 of 2). API functions Capsule applications

Function	Action
DisGetCapsuleVars ()	Obtains a copy of the current values of the Capsule application's @-variables.
DisSetCapsuleVars ()	Assigns values to the Capsule application's @-variables.
DisSubmitCapsule ()	Submits the specified Capsule application to the Capsule Services for execution.
DisSubmitCapsuleAndSetVars ()	Submits the specified Capsule application to the Capsule Services for execution, and assigns values to the Capsule application's @-variables.
DisQueryCapsuleInfo ()	Checks the current status of jobs in the Capsule Services queue.
DisSetCapsuleStatus ()	Specifies whether to cancel, purge, hold, release, or modify the specified Capsule application in the Capsule Services queue, or to set various job parameters.

Table 9 (Page 2 of 2). API functions Capsule applications

Function	Action
DisWaitCapsuleStatus ()	Pauses the personal-computer application until the specified Capsule application achieves the specified status.

Table 10. API functions for automated file distribution (AFD)

Function	Action
DisAFD ()	Performs a complete AFD session, but only for AFD lists that are set for automatic execution.
DisAFDList ()	Performs a complete AFD session, for all AFD lists, whether for manual or automatic execution.
DisInitAFD ()	Initializes an AFD session, collecting only those AFD lists that are set for automatic execution.
DisInitAFDList ()	Initializes an AFD session, collecting all AFD lists, whether for manual or automatic execution.
DisExecuteAFD ()	Executes an AFD session.
DisTermAFD ()	Terminates an AFD session.

Table 11. Other API functions

Function	Action
DisCancelRequest ()	Cancels the operation that the server is currently processing.
DisGetErrorInfo ()	Returns detailed results information for a specified function call or batch submission.
DisGetPerfInfo ()	Retrieves statistics monitoring the system's response time.
DisInitBatch ()	Begins a queue of functions for batch execution.
DisQueryResult ()	Checks the result of an asynchronous operation on the developer's network; determines whether a prior request has been executed and, if so, returns the completion code.
DisSetEventHandler ()	Notifies your event-handling procedure when various events occur.
DisSubmitBatch ()	Submits a queue of functions for batch execution.
DisTermBatch ()	Terminates a batch submission, discarding any calls that have not yet been submitted to the ODAS server.

Defined Data Types

Table 12 on page 40 lists the defined data types used in the C API functions.

Table 12. Data types

Type	Definition	Pointer type	Definition
ULONG	unsigned long	PULONG	unsigned long*
HDIS	ULONG	PHDIS	HDIS*
UINT	unsigned int	PUINT	unsigned int*
BYTE	unsigned char	CHAR FAR*	unsigned char*
USHORT	unsigned short	PCHAR	char*
		PVOID	void*

PSZ is a pointer to a null-terminated string.

Functions Reference Section

The remainder of this chapter presents the C-language API functions in alphabetical order, with full details including the calling syntax and examples of usage.

This chapter uses several notational conventions, described in the beginning of this manual, to indicate the correct syntax of each function.

DisAFD

This function performs a complete AFD session, but only for AFD lists that are set for automatic execution.

If your application must execute AFD operations conditionally, or if you want to specify how much data is to be transferred before actually confirming the action, you should use the individual functions DisInitAFD, then DisExecuteAFD, then DisTermAFD. DisAFDcalls those three functions.

Desktop Equivalent

There is no equivalent action for a user at a developer's workstation.

Syntax

INT DISENTRY DisAFD (UINT *fFlags*, HDIS *hdis*);

where:

fFlags	Must be 0 (zero).
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example performs a complete AFD session.

```
HDIS        sessionHandle;
. . .result = DisAFD (    , sessionHandle );
```

DisAFDList

This function performs a complete AFD session for all matching AFD lists, whether for manual or automatic execution.

If your application must execute AFD operations conditionally, or if you want to specify how much data is to be transferred before actually confirming the action, you should use the individual functions DisInitAFDList, then DisExecuteAFD, then DisTermAFD. DisAFDListcalls those three functions.

Desktop Equivalent

There is no equivalent action for a user at a developer's workstation.

Syntax

INT DISENTRY DisAFDList (PSZ *pszName*, PDISAFDINFO *pafdi*, UINT *fFlags*, HDIS *hdis*);

where:

pszName	Is a pointer to the name pattern to be compared with names of AFD lists assigned to the user or to the user's groups. The name pattern can contain a wildcard asterisk (*).
fFlags	Must be 0 (zero).
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example performs a complete AFD session for all matching AFD lists.

```
HDIS        sessionHandle;
. . .result = DisAFDList ( " ",  , sessionHandle );
```

DisCancelRequest

This function cancels the operation that the ODAS server is currently processing.

If the server is already at a certain point in processing the request you want to cancel, the server may not be immediately available for a new request.

If a batch request is being processed, some subrequests may already have been processed. Check the result codes of the individual subrequests to find which, if any, are complete.

Desktop Equivalent

There is no equivalent action for a user at a developer's workstation.

Syntax

INT DISENTRY DisCancelRequest (HDIS *hdis*);

where:

hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example cancels the current operation being processed on the ODAS server.

```
HDIS        sessionHandle;
. . .result = DisCancelRequest ( sessionHandle );
```

DisConnect

This function establishes a connection to the ODAS server. Call this function before calling any other ODAS API function (other than DisInitClient, which must be called first).

Desktop Equivalent

The equivalent action, if the user were at an Intelligent Decision Server workstation, would be to start the workstation from the OS/2 desktop.

Syntax

INT DISENTRY DisConnect (UINT *uCommType*, VOID FAR **pCommParms*, UINT *fFlags*, HDIS FAR **phdis*);

where:

uCommType	Must be 2or the constant DIS_COMM_LAN.
pCommParms	Is a pointer to the communication-parameters structure defined for the communications type specified in uCommType. The client application must supply the name of the server, as defined by the network software (such as the TCP/IP domain-name service, or DNS). The DISLANCOMM structure is:

```
typedef struct   _DISLANCOMM {
    USHORT cb;                          / Size of the structure, in bytes /
    USHORT usTimeout;             / Connection timeout; how many seconds to wait
                                                for a completed connection /
    CHAR FAR  pszServerName;     / Server's name /
} DISLANCOMM;
```

fFlags	Must be 0 (zero).
phdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Remarks

The ODAS administrator can configure the server to automatically update the ODAS software whenever a client computer with outdated software is connected. This updating can cause DisConnect to appear to take an abnormally long time to complete.

Example

This example specifies a connection with the communications parameters specified in the structure lanComm, then waits for completion.

```
HDIS          sessionHandle;
DISLANCOMM    lanComm;
. . ./ Reset all members of the structure to zero. /
memset (&lanComm,  , sizeof (lanComm));
lanComm.cb == sizeof (lanComm);
                                    / Fill the lanComm structure. /
lanComm.pszServerName == "MyODAS";
result == DisConnect ( DIS_COMM_LAN, &lanComm,  ,      &sessionHandle );
```

DisCopyObject

This function duplicates a desktop object. The new copy can be in the same location as the original or it can be in a specified container. (See “Containers” on page 10.)It can have the same name or a different name. If convertible, it can be in the same format or a different format.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to select an object and use the [*COPY*] function key to duplicate it.

Syntax

INT DISENTRY DisCopyObject (CHAR FAR **pszObjectName*, CHAR FAR **pszNewObjectName*, UINT *uNewObjectType*, UINT *fFlags*, HDIS *hdis*);

where:

pszObjectName	Specifies the original name of the desktop object to be copied.
pszNewObjectName	Specifies the name of the new copy.
uNewObjectType	Specifies a numerical code representing the type of desktop object that the new copy is to be. Choose the code from Table 2 on page 13.To copy without conversion, use DIS_OBJTYPE_DEFAULT.
fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_DELETE_SOURCE flag to move the original object into the desktop's Discard folder after the operation. Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove the original object from the desktop.

hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example copies the Next Report spreadsheet, then returns without waiting for completion:

```
result = DisCopyObject ("Next Report",
    "Copy of Next Report",      DIS_OBJTYPE_DEFAULT, DIS_NOWAIT, sessionHandle );
```

DisDeleteObject

This function deletes the specified object from the desktop.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to select an object and use the [DELETE] function key to remove it from the desktop.

Syntax

INT DISENTRY DisDeleteObject (CHAR FAR **pszObjectName*, UINT *fFlags*, HDIS *hdis*);

where:

pszObjectName	Specifies the name of the object to be deleted.
fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove the object from the desktop.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example completely deletes the Next Report spreadsheet from the desktop, then returns without waiting for completion:

```
result = DisDeleteObject ("Next Report",
     DIS_DELETE_EXPUNGE | DIS_NOWAIT, sessionHandle );
```

DisDisconnect

This function terminates the desktop session instance and the application's connection with the ODAS server.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to select *Exit Data Interpretation System*.

Syntax

INT DISENTRY DisDisconnect (UINT *fFlags*, HDIS *hdis*);

where:

fFlags	Must be 0 (zero).
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example ends the current desktop session:

```
result = DisDisconnect ( , sessionHandle );
```

DisExecuteAFD

This function executes an AFD session. DisExecuteAFD must be called after DisInitAFD (or DisInitAFDList), which initializes the AFD session.

If your application must execute AFD operations conditionally, or if you want to specify how much data is to be transferred before actually confirming the action, you should use the individual functions DisInitAFD (or DisInitAFDList), then DisExecuteAFD, then DisTermAFD. Otherwise, you can use DisAFD or DisAFDList, each of which calls those three functions.

If you do not want to perform the AFD operations identified by *DisInitAFD* (or *DisInitAFDList*), you should use *DisTermAFD* without calling *DisExecuteAFD*.

Desktop Equivalent

There is no equivalent action for a user at a IDSworkstation.

Syntax

INT DISENTRY DisExecuteAFD (UINT *fFlags*, HDIS *hdis*);

where:

fFlags	Must be 0 (zero).
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example begins the execution phase of the current desktop session:

```
result = DisExecuteAFD ( , sessionHandle );
```

DisGetCapsuleVars

This function obtains a copy of the current values of the Capsule application's @-variables. The values are returned in a *DISCAPSULEVARINFO* structure.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to select the {Show Controls} button in a Capsule application to open its User Input Controlwindow and look at the currently set @-variables.

Syntax

INT DISENTRY DisGetCapsuleVars (CHAR FAR **pszCapsuleName*, CHAR FAR **pszStartVar*, UINT FAR **pcVars*, VOID FAR **pBuf*, UINT FAR **pcbBuf*, UINT *fFlags*, HDIS *hdis*);

where:

pszCapsuleName	Specifies the name of the Capsule application whose variables are to be obtained.
pszStartVar	Specifies the first of the Capsule variables that are to be obtained. Do not include the @ character; for example, specify B for the @B variable.
pcVars	Is a pointer to the number of variables that are to be obtained. Upon completion, the function returns the number of variables that are actually obtained.
pBuf	Is a pointer to an array of one or more structures that receive the Capsule application's @-variables. The DISCAPSULEVARINFO structure is:

```
#define DIS_CAPSULEVAR_NAME_LEN       4
#define DIS_CAPSULEVAR_DESC_LEN       5     / Maximum description length /
#define DIS_CAPSULEVAR_VALUE_LEN 255    / Maximum value length /
#define DIS_CAPSULEVAR_TYPE_DEFAULT == / Preserve existing type and
                                         set new type /
typedef struct   _DISCAPSULEVARINFO {
     CHAR szName[DIS_CAPSULEVAR_NAME_LEN]                  / Variable name /
     ulType As Long;                          / Variable type /
     CHAR FAR  pszDescription;           / Variable's description /
     CHAR FAR  pszValue;                    / Variable's value /
} DISCAPSULEVARINFO;
```

The pszDescription and pszValue members are pointers to null-terminated strings.

pcbBuf	Is a pointer to the size of the pBuf buffer, in bytes. Upon completion, the function returns the actual length of the data it obtained.

	The buffer must have enough space to accept the description and value of the Capsule variable, each of which can vary in length. Therefore, specifying sizeof (DISCAPSULEVARINFO) is insufficient.
fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Remarks

While the at-sign (@) is an essential part of the variable name in the normal workstation environment, you cannot use it in the ODAS programming; specify “A” for @A, and so forth.

Example

This example retrieves the current @-variables of the Budget Capsule application, beginning with the @B variable, then returns without waiting for completion:

```
#define   BUFSIZE   1
PVOID                    pBuffer;
UINT                     numVariables,  bufferSize;
PDISCAPSULEVARINFO atVariables;
. . .
atVariables = (PDISCAPSULEVARINFO)malloc (BUFSIZE);
bufferSize = BUFSIZE;
numVariables = 1 ;
result = DisGetCapsuleVars ( "Budget Capsule", "B",
      &numVariables, atVariables, &bufferSize, DIS_NOWAIT,
     sessionHandle  );
```

DisGetErrorInfo

This function returns detailed results information for a specified function call or batch submission.

Desktop Equivalent

There is no equivalent action for a user at a IDSworkstation.

Syntax

INT DISENTRY DisGetErrorInfo (UINT *iRequest*, DISERRORINFO FAR **perri*, HDIS *hdis*);

where:

iRequest	Specifies the index of the function call for which results are desired. Use 0(zero)to get information about a function call that was not in a batch, or to get summary information about the most recent batch submission. Use a positive integer to get information about a specific function call within the most recent batch submission
perri	Is a pointer to a structure that receives the requested information. The DISERRORINFO structure is:

```
typedef struct   _DISERRORINFO {
     UINT cb;          /  Size of structure (must be set on entry)  /
     INT err;          /  Primary error code  /
     INT errSystem; /  System error code  /
     UINT uErrLoc;    /  Component in which error occurred  /
     UINT uErrType; /  Type of error  /
     INT errBaseSys;/  Untranslated, low-level error code  /
} DISERRORINFO,   PDISERRORINFO;
```

hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example returns detailed results about the third function call in the most recent batch submission:

```
DISERRORINFO    resultsBuffer;
. . .result == DisGetErrorInfo ( 3, &resultsBuffer, sessionHandle );
```

DisGetMultObjects

This function fetches one or more objects from the developer's desktop to a personal computer file, automatically converting their formats as needed. The names and types of the objects and the resultant files are returned in a DISGETOBJECTINFO structure.

The system tries to determine the type of each new file by matching the type of the original desktop object against the current entries in the conversion table. If no match is found, the system uses the default type specified in the table. To modify the conversion table for the current session, use the DisSetConversionTable function.

If you use the DIS_NOWAIT flag with DisGetMultObjects, no files are transferred until you also call the DisQueryResult function.

Desktop Equivalent

The most equivalent action, if the user were at an IDS workstation, would be to use the PC Directory tool to move files from the desktop to the workstation's OS/2 or DOS file system. Transferring files between the developer's desktop and a different personal computer is not possible without using ODAS.

Syntax

INT DISENTRY DisGetMultObjects (CHAR FAR **pszObjectSpec*, CHAR FAR **pszFileSpec*, UINT FAR **pcObjects*, VOID FAR **pBuf*, UINT FAR **pcbBuf*, UINT *fFlags*, HDIS *hdis*);

where:

pszObjectSpec — Is a string specifying the desired desktop objects. (See "Identifying Desktop Objects" on page 10.)

pszFileSpec — Specifies the base file name and path for all copies on the destination personal computer. The system appends a three-digit number to pszFileSpec for each copy made. For example, specifying "C:\HQ\FOO." causes the copies to be named C:\HQ\FOO.001, C:\HQ\FOO.002, and so forth. However, the period (.) is not provided automatically. In the example, specifying "C:\HQ\FOO" (without the period) causes the copies to be named C:\HQ\FOO001, C:\HQ\FOO002, and so forth.

Some flags affect this appendage. See fFlags, below.

pcObjects — Is a pointer to the maximum number of desktop objects to be retrieved. Upon completion, the function returns the actual number of desktop objects that were retrieved.

pBuf — Is a pointer to an optional buffer that receives information about the source objects and destination files.

If you do not specify the DIS_GMO_OBJECT_INFO flag in the fFlags parameter, the function returns information in the DISGETOBJECTINFO structure:

```
typedef struct    _DISGETOBJECTINFO {
    USHORT cb;                  / Size of this entry, in bytes /
    BYTE bObjType;              / Numerical code indicating the original type of
                                      desktop object, as shown in
                                    Table 2 on page 13 /
    BYTE bFileType;             / Numerical code indicating the destination file
                                      type, as shown in Table 1 on page 12 /
    ULONG cbFile;               / Size of destination file, in bytes /
    CHAR szName [DIS_OBJECT_NAME_LEN];
                                / Original desktop-object name /
} DISGETOBJECTINFO;
```

If you do specify the DIS_GMO_OBJECT_INFO flag in the fFlags parameter, the function returns information in the DISOBJECTINFO structure:

```
#define DIS_OBJECT_NAME_LEN   8      / Maximum length of a desktop
                                                              object's name  /
typedef struct       _DISOBJECTINFO {
      USHORT cb;                    /  Size of this entry, in bytes  /
      BYTE bObjType;                /  Numerical code indicating the type of desktop
                                          object, as shown in
                                          Table 2 on page 13  /
      BYTE fbFlags;                 /  Reserved for future use;
                                          specify   (zero)  /
      ULONG id;                     /  Unique identifier for each desktop object, to
                                          distinguish between objects with identical names  /
      ULONG timeCreation;           /  Time and date when the object was created /
      ULONG timeLastRead;           /  Time and date when it was last read  /
      ULONG timeLastWrite;          /  Time and date when it was last written  /
      ULONG cbObject;               /  Size of the object, in bytes  /
      CHAR szName [DIS_OBJECT_NAME_LEN];     /  Object's name  /
} DISOBJECTINFO;
```

Specify NULL if you do not want DisGetMultObjects to return this information.

pcbBuf Is a pointer to the size of the pBuf buffer, in bytes. Upon completion, the function returns the actual length of the buffer that it retrieved. Specify NULL if you specify NULL for pBuf.

fFlags Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

Specify the DIS_DELETE_SOURCE flag to move the objects into the desktop's Discard folder after the operation.

Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove the objects from the desktop after the operation.

Specify the DIS_OVERWRITE flag to overwrite any existing files that have the same name as retrieved files. If the client computer has an existing file with the same name, and you do not specify DIS_OVERWRITE, the function returns an error.

Specify the DIS_GMO_ID_NAMEflag to alter the text appended to the name in the pszFileSpec parameter. With DIS_GMO_ID_NAME, the unique file-identifier number of the source object is appended instead of the three-digit number that occurs with no flag.

Specify the DIS_GMO_SOURCE_NAMEflag to alter the text appended to the name in the pszFileSpec parameter. With DIS_GMO_SOURCE_NAME, the original IDS name of the source object is appended instead of the three-digit number that occurs with no flag.

Specify the DIS_GMO_OBJECT_INFOflag to return pBuf information in the DISOBJECTINFO structure instead of the default DISGETOBJECTINFO structure.

hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Examples

This example copies all objects from the desktop Mail Tray into a DOS or OS/2 directory DISMAIL as files named ITEM001, ITEM002, and so forth, and then completely deletes the original objects from the desktop. The function returns data in the DISGETOBJECTINFO structure, and waits for completion.

```
#define BUFSIZE 1
UINT    numMessages = 32;       /  maximum number to get  /
PDISGETOBJECTINFO       objectsBuffer;
UINT       bufferSize;
. . .objectsBuffer = (PDISGETOBJECTINFO)malloc (BUFSIZE);
bufferSize = BUFSIZE;
result = DisGetMultObjects ( "Mail Tray\ ", "DISMAIL\ITEM",
       &numMessages, objectsBuffer, &bufferSize,
       DIS_DELETE_EXPUNGE | DIS_WAIT, sessionHandle );
```

The next example copies the same objects but does not return the DISGETOBJECTINFO structure:

```
UINT    numMessages = 32;
. . .result = DisGetMultObjects ( "Mail Tray\ ",
        "DISMAIL\ITEM",
       &numMessages, NULL, NULL,
         DIS_DELETE_EXPUNGE | DIS_WAIT,
      sessionHandle  );
```

DisGetObject

This function fetches an object from the developer's desktop to a personal computer file, automatically converting its format as needed. You can specify the destination file format to override the conversion table that is currently defined for the session. To modify the conversion table for the current session, use the DisSetConversionTable function.

If you use the *DIS_NOWAIT* flag with *DisGetObject*, the file is not transferred until you also call the *DisQueryResult* function.

Desktop Equivalent

The most equivalent action, if the user were at an IDS workstation, would be to use the PC Directory tool to move a file from the desktop to the OS/2 file system. Transferring a file between the developer's desktop and a different personal computer is not possible without using ODAS.

Syntax

INT DISENTRY DisGetObject (CHAR FAR **pszObjectName*, CHAR FAR **pszFileName*, UINT *uFileType*, ULONG FAR **pcbFile*, UINT *cchLineWidth*, UINT *fFlags*, HDIS *hdis*);

where:

pszObjectName	Specifies the name of the desktop object to be retrieved. A desktop can have more than one object with the same name. To indicate a specific object, you can specify "MET@number", where number is a string created from the hexadecimal representation of the unique object identifier returned by the DisQueryObjectInfo function. (See "Objects with Non-Unique Names" on page 11.)
pszFileName	Specifies the file name on the destination personal computer.
uFileType	Specifies a numerical code representing the type of file conversion. Choose the code, from Table 1 on page 12, corresponding to the destination file type. You can specify DIS_FILETYPE_DIS to bypass conversion or DIS_FILETYPE_DEFAULT to select the default conversion.
pcbFile	Specifies the maximum number of bytes to fetch. If you specify 0 (zero), the function fetches the entire object, regardless of size. Upon completion, the function returns the actual number of bytes that were retrieved. Specify NULL if you want DisGetObject to ignore this parameter.
cchLineWidth	Specifies the maximum line width for a destination text file. If the destination file is not a text file, or you want to specify no maximum, specify 0 (zero).
fFlags	Specifies optional flags, as described in "Common Bit Flags" on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_DELETE_SOURCE flag to move the object into the desktop's Discard folder after the operation. Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove the object from the desktop after the operation. Specify the DIS_OVERWRITE flag to overwrite any existing file that has the same name as the retrieved file. If the client computer has an existing file with the same name, and you do not specify DIS_OVERWRITE, the function returns an error.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example copies the Next Report spreadsheet from the desktop into a Lotus 1-2-3 spreadsheet named NEXTRPT.WKS, then completely deletes the document from the desktop. The function call waits for completion.

```
result = DisGetObject ("Next Report", "NEXTRPT.WKS",
    DIS_FILETYPE_WKS, NULL,  , DIS_DELETE_SOURCE | DIS_WAIT,
   sessionHandle  );
```

DisGetPerfInfo

This function retrieves statistics monitoring the system's response time for the current session. This response-time monitoring (RTM) data is gathered only after the DIS_INIT_RTM flag has been passed to DisInitClient.

Desktop Equivalent

There is no equivalent action for a user at a developer's workstation.

Syntax

INT DISENTRY DisGetPerfInfo (VOID FAR *pBuf, UINT FAR *pcbBuf, UINT *fFlags*, HDIS *hdis*);

where:

pBuf — Points to a buffer that receives the DISAPIPERF structure. All response-time statistics are given in milliseconds. The DISAPIPERFSERVICE structure is:

```
typedef struct   _DISAPIPERFSERVICE {
     ULONG msec;              / Service's response time /
     ULONG msecAccum;         / Service's accumulated response time since
                                the last time the accumulators were reset /
     ULONG msecTotal;         / Total accumulated response time /
} _DISAPIPERFSERVICE;
```

```
typedef struct    _DISAPIPERF {
      USHORT cRequests;        /  Number of requests processed in this session  /
      BYTE cServices;          /  Number of services for which response-time
                                        statistics are available in the Services field of
                                        the DISAPIPERF structure  /
      BYTE bReserved1;           /  (Reserved for future use)  /
      ULONG msecIdle;            /  Time spent outside any API functions, that is,
                                        in the user application or interactive delay
                                         (waiting for user input)  /
      ULONG msecActive;          /  Time the API is active; "response time" for the
                                        function or for a complete batch request  /
      ULONG msecCallActive;      /  Individual response time for the most recent
                                          batch subrequest completed  /
      ULONG msecTotalIdle;         /  Accumulated idle time for the application  /
      ULONG msecTotalActive     /  Accumulated active time for the application  /
      ULONG msecAccumInterval; /  Time elapsed since the accumulators were last
                                           reset (length of the current accumulation)  /
      ULONG msecAccumActive;   /  Accumulated active time since the accumulators
                                        were last reset  /
       DISAPIPERFSERVICE Services [DIS_PERF_SERVICE_MAX];
} DISAPIPERF,  PDISAPIPERF;
                                                    /  Services [DIS_PERF_SERVICE_PORT] gives
                                          response-time statistics for ODAS service.
                                          Services [DIS_PERF_SERVICE_DISK] gives
                                           response-time statistics for IDS file servers.
                                           Services [DIS_PERF_SERVICE_DIR] gives
                                           response-time statistics for IDS directory
                                          service.
                                           Services [DIS_PERF_SERVICE_CAPS] gives
                                           response-time statistics for IDS Capsule
                                            Service.
                                             Services [DIS_PERF_SERVICE_CONV] gives
                                             response-time statistics for IDS Conversion
                                            Service.   /
```

pcbBuf	Points to the size of the pBuf buffer. Before returning, the function sets the value to the size of the structure actually placed in the pBuf buffer.
fFlags	Specifies optional flags. One of the following resets the accumulators after reading them: DIS_GPI_RESET_ACCUM 0x0001
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example retrieves the latest RTM data for the session:

```
UINT cb;
DISAPIPERF perf;
cb==sizeof (perf);
result == DisGetPerfInfo ( &perf, &cb,  , sessionHandle );
```

DisGetSysMessage

This function retrieves one or more pending messages from the system administrator. Each message is returned in a DISSYSMESSAGE structure.

Desktop Equivalent

The equivalent action, if the user were at an Intelligent Decision Server workstation, would be to click on the desktop's *IMPORTANT MESSAGE*button and read the displayed message.

Syntax

INT DISENTRY DisGetSysMessage (UINT FAR **pcMessages*, VOID FAR **pBuf*, UINT FAR **pcbBuf*, UINT *fFlags*, HDIS *hdis*);

where:

pcMessages — Is a pointer to the maximum number of messages to be retrieved. The function sets this to the number of messages actually retrieved.

pBuf — Is a pointer to an array of one or more structures that receive the messages. The DISSYSMESSAGE structure is:

```
#define DIS_SYSMESSAGE_LEN    256    / Maximum message length /
typedef struct    _DISSYSMESSAGE {
     USHORT cb;                          / Size of structure, in bytes /
     USHORT usReserved;                  / Reserved for future use; specify   (zero) /
     ULONG time;                         / Time message was received /
     CHAR szMessage[DIS_SYSMESSAGE_LEN];     / Message text /
} DISSYSMESSAGE;
```

pcbBuf — Is a pointer to the size of the pBuf buffer, in bytes. Upon completion, the function returns the actual length of the data it obtained.

fFlags — Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

hdis — Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example retrieves from zero to five system messages, but returns without waiting for completion:

```
#define BUFSIZE 1
UINT        numMessages = 5;
PDISSYSMESSAGE      msgBuffer;
UINT      bufferSize;
. . .msgBuffer = ( PDISSYSMESSAGE ) malloc ( BUFSIZE );
buffersize = BUFSIZE;
result = DisGetSysMessage ( &numMessages, msgBuffer,
    &bufferSize, DIS_NOWAIT, sessionHandle );
```

DisInitAFD

This function initializes an AFD session. It collects only AFD lists that are set for automatic execution. (Another function, DisInitAFDList, collects all AFD lists, manual or automatic, that match the specified name pattern.)

During the initialization phase of AFD, the function:

Collects all lists that are assigned to the user and the user's groups and are set for automatic execution

Compares the last-written dates of the client and server files specified in the lists

Constructs a list of the needed AFD operations (sending, receiving, and deleting)

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

INT DISENTRY DisAFD (PDISAFDINFO *pafdi*, UINT *fFlags*, HDIS *hdis*);

where:

pafdi — Is a pointer to a structure that receives a summary of the operations that AFD intends to perform. Specify NULL if this information is not needed. You must initialize the cb field of the structure to the size of the buffer you are providing. The DISAFDINFO structure is:

```
typedef struct   _DISAFDINFO {
     USHORT cb;                    / Size of the structure, in bytes /
     USHORT cSend;                 / Number of files to send to the server /
     USHORT cReseive;              / Number of files to receive from the server /
     USHORT cAddClient;            / Number of files to add to the client /
     USHORT cAddServer;            / Number of files to add to the server /
     USHORT cDelClient;            / Number of files to delete from the client /
     USHORT cDelServer;            / Number of files to delete from the server /
     USHORT cbSend;                / Total number of bytes to send to the server /
     USHORT cbReceive;             / Total number of bytes to receive from the
                                                                  server /
} DISAFDINFO;
```

fFlags — Must be 0 (zero).

hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example initializes an AFD session.

```
HDIS      sessionHandle;
. . .result = DisAFD ( &myAfdInfo,  , sessionHandle );
```

DisInitAFDList

This function initializes an AFD session. It collects all AFD lists, whether they are set for manual or automatic execution.

During the initialization phase of AFD, the function:

Collects all lists that are assigned to the user and the user's groups, whether set for manual or automatic execution

Compares the last-written dates of the client and server files specified in the lists

Constructs a list of the AFD operations needed (sending, receiving, and deleting)

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

INT DISENTRY DisAFDList (PSZ *pszName*, PDISAFDINFO *pafdi*, UINT *fFlags*, HDIS *hdis*);

where:

pszName	Is a pointer to the name pattern to be compared with names of AFD lists assigned to the user or to the user's groups. The name pattern can contain a wildcard asterisk (*).
pafdi	Is a pointer to a structure that receives a summary of the operations that AFD intends to perform. Specify NULL if this information is not needed. You must initialize the cb field of the structure to the size of the buffer you are providing. The DISAFDINFO structure is:

```
typedef struct   _DISAFDINFO {
     USHORT cb;                          / Size of the structure, in bytes  /
     USHORT cSend;                     / Number of files to send to the server /
     USHORT cReseive;                / Number of files to receive from the server /
     USHORT cAddClient;             / Number of files to send to the client /
     USHORT cAddServer;            / Number of files to add to the server /
     USHORT cDelClient;             / Number of files to delete from the client /
     USHORT cDelServer;            / Number of files to delete from the server /
     USHORT cbSend;                  / Number of bytes to send to the server /
     USHORT cbReceive;             / Number of bytes to receive from the server /
} DISAFDINFO;
```

fFlags	Must be 0 (zero).
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example initializes an AFD session for all lists starting with field.

```
HDIS      sessionHandle;
DISAFDINFO myAfdInfo;
myAfdInfo.cb == sizeof ( myAfdInfo );
. . .result == DisInitAFDList ( "field ", &myAfdInfo,  , sessionHandle );
```

DisInitBatch

This function starts a queue of submitted functions. All subsequent functions are queued until DisSubmitBatch is reached; at that point, the queued functions are submitted to the ODAS server for execution.

Each function in the batch can return errors due to local conditions (such as a missing local file in a DisPutObject call). In this situation, you can use the DisTermBatch command to terminate the batch instead of submitting it to the ODAS server. (See "DisTermBatch" on page 88.)

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

INT DISENTRY DisInitBatch (UINT *fFlags*, HDIS *hdis*);

where:

fFlags	Must be 0 (zero).
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example enables, creates, and submits a batch request comprising four desktop operations:

```
result = DisInitBatch ( , sessionHandle );
    DisGetObject ( "Next Report", "NEXTRPT.WKS",
        DIS_FILETYPE_WKS, NULL,  ,
        DIS_DELETE_SOURCE |
        DIS_WAIT,  sessionHandle);
    DisGetSysMessage ( &numMessages, msgBuffer, &bufferSize,
        ,  sessionHandle);
    DisCopyObject ( "Next Report", "Copy of Next Report",
        DIS_OBJTYPE_DEFAULT,  , sessionHandle);
    DisLogoff ( , sessionHandle);
result = DisSubmitBatch ( batchResults, 4, DIS_NOWAIT,
   sessionHandle  );
```

DisInitClient

This function initializes the ODAS application. You must call *DisInitClient* before calling any other ODAS API functions.

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

INT DISENTRY DisInitClient (UINT *fFlags*);

where:

fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_INIT_RTMflag if you plan to call DisGetPerfInfofor performance monitoring.

Example

This example initializes the client computer before any other API functions are called:

```
result = DisInitClient (  );
```

DisLogoff

This function logs off a developer's desktop. The client application can access several different desktops during a single session, but not simultaneously.

Special flags, described below, start or suppress automated file distribution (AFD). AFD lets the ODAS server update your application software as needed.

Desktop Equivalent

The equivalent action, if the user were at an IDS workstation, would be to select the desktop's Log Offbutton.

Syntax

INT DISENTRY DisLogoff (UINT *fFlags*, HDIS *hdis*);

where:

fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_LOFF_AFD_NOWflag to start an AFD session before logging off, regardless of the user's AFD mode setting in the user-profile database. Specify the DIS_LOFF_AFD_SUPPRESSflag to avoid starting an AFD session when the user's AFD mode in the user-profile database is set for Automatic at logoff. This flag also suppresses a deferred AFD session that would be started if DisLogon is called with the DIS_LON_AFD_DEFERflag.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example logs off the desktop that is currently logged on, then returns without waiting for completion:

```
result = DisLogoff (DIS_WAIT, sessionHandle);
```

DisLogon

This function accesses a developer's desktop. The client application can access several different desktops during a single session, but not simultaneously. If the application is already logged on, logging on again terminates the previous logon.

Special flags, described below, start or suppress automated file distribution (AFD). AFD lets the ODAS server update your application software as needed.

Desktop Equivalent

The equivalent action, if the user were at an IDS workstation, would be to enter a desktop name and password.

Syntax

INT DISENTRY DisLogon (CHAR FAR **pszDesktopName*, CHAR FAR **pszDesktopPassword*, UINT *fFlags*, HDIS *hdis*);

where:

pszDesktopName	Specifies the desktop to be accessed.
pszDesktopPassword	Specifies the password for that desktop.

fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_LON_AFD_NOWflag to start an AFD session after logging on, regardless of the user's AFD mode setting in the user-profile database. Specify the DIS_LON_AFD_SUPPRESSflag to avoid starting an AFD session when the user's AFD mode in the user-profile database is set for Automatic at logon. Specify the DIS_LON_AFD_DEFERflag to defer starting an AFD session until the user logs off, when the user's AFD mode in the user-profile database is set for Automatic at logon.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example accesses the desktop named *George*, and does not return until the request is completed:

```
result = DisLogon ( "George", "widget", DIS_WAIT,       sessionHandle);
```

DisMoveObject

This function relocates a desktop object into a different container, renames the object, or performs both actions.

Desktop Equivalent

The equivalent action, if the user were at an IDS workstation, would be to select an object and use the [MOVE] function key to relocate it on the desktop. The equivalent to the optional renaming would be to open the object's Icon Options window, delete the old name, and type a new one.

Syntax

INT DISENTRY DisMoveObject (CHAR FAR **pszObjectName*, CHAR FAR **pszNewObjectName*, UINT *fFlags*, HDIS *hdis*);

where:

pszObjectName	Specifies one or both of the original name and location of the object. (See “Containers” on page 10.)
pszNewObjectName	Specifies one or both of the new name and location of the object. If you do not specify a location, the object remains in its current location. If you do not specify a new name, the object keeps its current name. If you specify a new name and no new location, the object is renamed.
fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

	Specify the DIS_OVERWRITE flag to replace a existing desktop object that has the same name as a moved object and to move the existing object into the Discard folder. Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove from the desktop an object that has the same name as a moved object.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example moves the Next Report document from the In Basket folder to the Out Basket folder and renames it Finished Report, but returns without waiting for completion:

```
result = DisMoveObject ("In Basket\Next Report",
     "Out Basket\Finished Report", DIS_NOWAIT,
    sessionHandle);
```

DisPutMultObjects

This function sends one or more personal computer files to the developer's desktop, automatically converting their formats as needed.

The system tries to determine the type of each new desktop object by matching the file extension of the original file against the current entries in the conversion table. If the file extension does not match any table entry, the system uses the default type specified in the table. To modify the conversion table for the current session, use the DisSetConversionTable function.

When using DisPutMultObjects with the DIS_DELETE_SOURCE flag, you should either call it synchronously by avoiding the DIS_NOWAIT flag, or call the DisQueryResult function to wait for completion of the DisPutMultObjects call. This caveat is necessary because DisPutMultObjects with DIS_DELETE_SOURCE requires some cleanup processing on the client computer.

Desktop Equivalent

The most equivalent action, if the user were at an IDS workstation, would be to use the PC Directory tool to move files to the desktop from the OS/2 file system. Transferring files between the developer's desktop and a different personal computer is not possible without using ODAS.

Syntax

INT DISENTRY DisPutMultObjects (CHAR FAR **pszFileSpec*, CHAR FAR **pszContainerName*, UINT *fFlags*, HDIS *hdis*);

where:

pszFileSpec	Is a string specifying the personal computer files to be sent. (See “Identifying Desktop Objects” on page 10.)

pszContainerName	Is a string specifying the name of the desktop container where the copies are to be placed. (See “Containers” on page 10.)
fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_DELETE_SOURCE flag to delete the files from the personal computer's directory after the operation. Specify the DIS_OVERWRITE flag to replace a existing desktop object that has the same name as a copied object and to move the existing object into the Discard folder. Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove from the desktop an object that has the same name as a copied object.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example sends files MAIL.TXT and MAIL.WKS to the desktop's My Mail folder, then deletes them from the personal computer. The function call waits for completion.

```
result = DisPutMultObjects ( "MAIL. ", "My Mail",
        DIS_DELETE_SOURCE | DIS_WAIT, sessionHandle);
```

DisPutObject

This function sends one personal computer file to the developer's desktop, automatically converting its format as needed.

When using DisPutObject with the DIS_DELETE_SOURCE flag, you should either call it synchronously by avoiding the DIS_NOWAIT flag, or call the DisQueryResult function to wait for completion of the DisPutObject call. This caveat is necessary because DisPutObject with DIS_DELETE_SOURCE requires some cleanup processing on the client computer.

Desktop Equivalent

The most equivalent action, if the user were at an IDS workstation, would be to use the PC Directory tool to move files to the desktop from the OS/2 file system. Transferring files between the developer's desktop and a different personal computer is not possible without using ODAS.

Syntax

INT DISENTRY DisPutObject (CHAR FAR **pszFileName*, UINT *uFileType*, ULONG *cbMax*, PSZ *pszObjectName*, UINT *uObjectType*, UINT *fFlags*, HDIS *hdis*);

where:

pszFileName	Is a pointer to the file identifier of the personal computer file to be sent.

uFileType	Specifies a numerical code representing the type of file to be sent. Choose the code, from Table 1 on page 12, corresponding to the type of source file. You can specify DIS_FILETYPE_DIS to bypass conversion or DIS_FILETYPE_DEFAULT to select the default file type, based on the suffix of the original file name.
cbMax	Specifies the maximum number of bytes to transfer. Specify 0 (zero) if you do not want to set any limit.
pszObjectName	Is a pointer to the name you want to give the new desktop object.
uObjectType	Specifies a numerical code representing the type of desktop object that the new copy is to be. Choose the code from Table 2 on page 13.Specify DIS_OBJTYPE_DEFAULT to let the conversion table decide the object type, based on the value of uFileType or on the suffix of the original file name.
fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_DELETE_SOURCE flag to delete the file from the personal computer's directory after the operation. Specify the DIS_OVERWRITE flag to replace a existing desktop object that has the same name as a copied object and to move the existing object into the Discard folder. Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove from the desktop an object that has the same name as a copied object.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example sends the REPORT.TXT file from the personal computer to the developer's desktop, then deletes it from the personal computer. The function waits for completion.

```
PSZ    fileName = "REPORT.TXT",
       objectName == "Claybrook Report";
. . .
result == DisPutObject (fileName, DIS_FILETYPE_DEFAULT,  ,
     objectName,  DIS_OBJTYPE_DEFAULT,
      DIS_DELETE_SOURCE | DIS_WAIT, sessionHandle);
```

DisQueryCapsuleInfo

This function checks the current status of jobs in the Capsule Services queue. Each job's information is returned in a DISCAPSULEJOBINFO structure.

This function returns information about only those jobs that are waiting to run or are currently running. It does not return information about jobs that have finished running. You can use DisWaitCapsuleStatus to check on the jobs that might have already completed.

Desktop Equivalent

The equivalent action, if the user were at an IDS workstation, would be to open the Capsule Services icon, select Show Queue, and visually inspect the window.

Syntax

INT DISENTRY DisQueryCapsuleInfo (UINT *uMode*, ULONG *idJob*, UINT *uInfoLevel*, UINT FAR **pcJobs*, VOID FAR **pBuf*, UINT FAR **pcbBuf*, UINT *fFlags*, HDIS *hdis*);

where:

uMode — Is the information selection code, specifying which information you want to retrieve. Table 13 lists the valid values.

Table 13. Information selection codes for DisQueryCapsuleInfo ()

Selection code	Value	Description
DIS_QCI_JOB	1	Return information about the job that the idJob number identifies.
DIS_QCI_FIRST	2	Return information about the first n jobs in the execution queue, where n is the smaller of pcJobs or the number of jobs that fit in the pBuf buffer. The first job is defined as the one with the earliest scheduled starting time.
DIS_QCI_NEXT	3	Return information about the next n jobs in the execution queue, where n is as described for DIS_QCI_FIRST.

idJob — Is the identifying number of the first job whose information is to be retrieved. The *DisSubmitCapsule* function provides this number. If *uMode* is not *DIS_QCI_JOB* (*1*), then *idJob* is ignored.

uInfoLevel — Specifies the amount of information to be retrieved. Table 14 lists the valid values.

Table 14. Detail selection codes for DisQueryCapsuleInfo ()

Selection code	Value	Description
DIS_QCIL_TEST	0	Check for the job's existence and return no other information.
DIS_QCIL_STANDARD	1	Return information in the DISCAPSULEJOBINFO structure.

pcJobs — Is a pointer to the number of jobs whose information is to be retrieved. The function sets this to the actual number of jobs for which it retrieves information.

pBuf — Is a pointer to an array of one or more structures that receive the requested information. The DISCAPSULEJOBINFO structure is:

```
typedef struct   _DISCAPSULEJOBINFO_L1 {
    USHORT cb;                  / Size of this entry, in bytes /
    BYTE bPriority;             / Job priority; specify 1 for the
                                                    lowest priority to
                                                    31 for the highest priority /
    BYTE bState;                / Job state /
    ULONG id;                   / Job identifier /
    CHAR FAR  pszOwner;         / Desktop that submitted the job /
    CHAR FAR  pszDescription;   / Job's description /
    CHAR FAR  pszAliasName;     / Scheduler alias name /
    ULONG timeSubmitted;        / Time when job was submitted /
    ULONG timeScheduled;        / Job's scheduled starting time /
    ULONG secsMaxStartDelay;    / Maximum delay past scheduled time,
                                                                in seconds /
    ULONG secsMaxExecution;     / Maximum execution time, in seconds /
} DISCAPSULEJOBINFO;
```

The bState member returns one of the numerical codes in Table 15 to indicate current state of execution.

Table 15. Capsule job-state codes

Job-state code	Value	Description
DIS_CJI_WAITING	1	The job is waiting for its scheduled time.
DIS_CJI_READY	2	The job is waiting for an execution resource.
DIS_CJI_HELD	3	The job is being held.
DIS_CJI_RUNNING	4	The job is currently executing.
DIS_CJI_ABENDED	5	The job has been cancelled or has been abnormally terminated due to a Capsule execution error.

The pszDescription and pszAliasName members are pointers to null-terminated strings.

pcbBuf — Is a pointer to the size of the pBuf buffer, in bytes. Upon completion, the function returns the actual length of the data it obtained.

fFlags — Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

Specify the DIS_QCI_OWNEDflag to return information about only those jobs that were submitted by the user issuing the request. Otherwise, the returned information includes jobs submitted by all users.

hdis — Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example retrieves information about the first 10 jobs in the execution queue, then returns that information in the capsuleStatus buffer without waiting for completion.

```
#define BUFSIZE 1
ULONG                    jobNumber;
UINT                     numJobs = 1 , bufferSize;
PDISCAPSULEJOBINFO   capsuleStatus
. . .
result = DisSubmitCapsule ( "Budget Capsule",
      "Capsule Execution Drawer",      jobParameters, &jobNumber,
     DIS_NOWAIT,  sessionHandle);
      /  DisQueryCapsuleInfo needs the job number that
      DisSubmitCapsule sets.  /
. . .
capsuleStatus = (PDISCAPSULEJOBINFO) malloc (BUFSIZE);
bufferSize = BUFSIZE;
result = DisQueryCapsuleInfo (DIS_QCI_JOB, jobNumber,
      DIS_QCIL_STANDARD, &numJobs, capsuleStatus, &bufferSize,
     DIS_NOWAIT,  sessionHandle);
```

DisQueryObjectInfo

This function checks the type, size, creation time, and other information of one or more desktop objects. The function returns the information in one or more DISOBJECTINFO structures.

Desktop Equivalent

The most equivalent action, if the user were at an IDS workstation, would be to open a container holding the objects and show the container's contents with their sizes and dates.

Syntax

INT DISENTRY DisQueryObjectInfo (CHAR FAR **pszObjectSpec*, UINT *uInfoLevel*, UINT FAR **piStart*, UINT FAR **pcEntries*, VOID FAR **pBuf*, UINT FAR **pcbBuf*, UINT *fFlags*, HDIS *hdis*);

where:

pszObjectSpec	Is a string specifying the desired desktop objects. (See “Identifying Desktop Objects” on page 10.)
uInfoLevel	Specifies the amount of information to be retrieved. Table 16 lists the valid values.

Table 16 (Page 1 of 2). Detailselection codes for DisQueryObjectInfo ()

Selection code	Value	Description
DIS_QOIL_TEST	0	Check for the object's existence and return no other information.

Table 16 (Page 2 of 2). Detailselection codes for DisQueryObjectInfo ()

Selection code	Value	Description
DIS_QOIL_STANDARD	1	Return information in the DISOBJECTINFO structure.
DIS_QOIL_FULLPATH	2	Return the full path name of the object. With this value, you cannot use wildcard asterisks (*) in the function.

piStart — Is a pointer to an integer specifying the object to be queried. Use 0 to begin the search with the first object in a desktop container, 1 to begin with the second object, and so on. If you store the index value between calls, a series of calls can search the container sequentially. The function sets this parameter to 0 when it has investigated the last object in the container.

pcEntries — Is a pointer to an integer specifying the quantity of desktop objects whose information is to be retrieved. The function sets this to the actual number of objects for which it retrieves information.

pBuf — Is a pointer to an array of one or more structures that receive the requested information. The DISOBJECTINFO structure is:

```
#define DIS_OBJECT_NAME_LEN   8      / Maximum length of a desktop
                                                          object's name  /
typedef struct   _DISOBJECTINFO {
     USHORT cb;                      / Size of this entry, in bytes  /
     BYTE bObjType;                  / Numerical code indicating the type of desktop
                                           object, as shown in Table 2 on page 13  /
     BYTE fbFlags;                  / Reserved for future use; specify   (zero)  /
     ULONG id;                       / Unique identifier for each desktop object, to
                                             distinguish between objects with identical
                                                                  names   /
     ULONG timeCreation;            / Time and date when the object was created /
     ULONG timeLastRead;            / Time and date when it was last read  /
     ULONG timeLastWrite;           / Time and date when it was last written  /
     ULONG cbObject;                 / Size of the object, in bytes  /
     CHAR szName[DIS_OBJECT_NAME_LEN];     / Object's name  /
} DISOBJECTINFO;
```

The timeCreation, timeLastRead, and timeLastWrite members are integers measuring the number of seconds since the start of January 1, 1900.

pcbBuf — Is a pointer to the size of the pBuf buffer, in bytes. Upon completion, the function returns the actual length of the data it obtained.

fFlags — Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

Specify the DIS_DELETE_SOURCE or DIS_DELETE_IFEMPTY flag to move the objects into the desktop's Discard folder after the operation.

Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove the objects from the desktop.

You can also specify the flags listed in Table 17 on page 69.

Table 17. Flags for DisQueryObjectInfo ()

Flag	Value	Description
DIS_QOI_COUNT_CONTENTS	1	Count the contents of the container and return them in the cbObject field. Without DIS_QOI_COUNT_CONTENTS, the function returns the container's size, in bytes, in the cbObject field.
DIS_QOI_MININFO	2	Speed the processing by not returning the time Last Read, timeLastWrite, cbFile, and attrFilefields.

hdis — Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example queries the Mail Tray, then waits for completion before it returns.

```
int                 index =  , bufferSize;
int                 maxObjects == MAX_OBJS_RETURNED;
DISOBJECTINFO    objectBuffer[MAX_OBJS_RETURNED];
. . .

bufferSize = sizeof (objectBuffer);
result == DisQueryObjectInfo ("Mail Tray\\",
    DIS_QOIL_STANDARD, &index, &maxObjects,
        objectBuffer,
         &bufferSize, DIS_WAIT, sessionHandle)
```

DisQueryResult

This function checks the result of an asynchronous operation on the IDS network. You can call DisQueryResult to determine whether a prior request has been executed and, if so, to return one of the completion codes shown in Table 18.

Table 18 (Page 1 of 2). Completion codes for DisQueryResult ()

Completion code	Description
ERROR_MET_REQUEST_PENDING	The specified operation is still awaiting completion.

Table 18 (Page 2 of 2). Completion codes for DisQueryResult ()

Completion code	Description
ERROR_MET_SUBREQUEST_ERROR	If you specified iSubRequest = 0, at least one operation within the batch returned an error. For more specific information about the error, specify the index of the specific operation or check the results buffer used in the DisSubmitBatch function.
ERROR_MET_REQUEST_NOT_FOUND	The index you specified was not within the number of operations in the most recent batch request.

If you use the *DIS_NOWAIT* flag with *DisGetObject* or *DisGetMultObjects*, no files are transferred until you also call the *DisQueryResult* function.

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

INT DISENTRY DisQueryResult (UINT *iSubRequest*, UINT *fFlags*, HDIS *hdis*);

where:

iSubRequest	Specifies an operation within a batch request. If the operation was not a batch request, or you want to check a global batch result, specify 0 (zero).
fFlags	Specifies optional flags, as described in "Common Bit Flags" on page 14.Specify the DIS_NOWAIT flag to force this function to return as soon as the request has been sent to the ODAS server.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example checks the result of the first function in a previous batch request, then returns without waiting for completion:

```
result = DisQueryResult (1, DIS_NOWAIT, sessionHandle);
```

DisQuerySessionInfo

This function checks one of: the current container path; the number of pending system messages, number of items in the mail tray, and number of new items in the mail tray since the previous query; or information about the ODAS service. The function returns the information as a string or structure, as described in Table 19 on page 71.

Desktop Equivalent

If the user were at an IDS workstation, the action equivalent to selection code 1 would be to look to see if the desktop is displaying the IMPORTANT MESSAGEbutton, and to

open and look at the mail tray. There is no workstation action equivalent to checking the number of new items in the mail tray since the previous query.

There is no workstation action equivalent to selection code 2.

Syntax

INT DISENTRY DisQuerySessionInfo (UINT *iInfo*, VOID FAR **pBuf*, UINT FAR **pcbBuf*, UINT *fFlags*, HDIS *hdis*);

where:

iInfo — Is the information selection code, specifying which information you want to retrieve. Table 19 lists the valid values.

Table 19. Information selection codes for DisQuerySessionInfo ()

Selection code	Value	Description
DIS_QSI_MSGCOUNT	*1*	Return a *DISMSGCOUNT* structure, containing various information about pending mail and system messages.
DIS_QSI_CURRENTPATH	*2*	Return the current container path as a null-terminated string.
DIS_QSI_SERVICEINFO	*3*	Return a *DISSERVICEINFO* structure, containing various information about the ODAS service.

pBuf — Is a pointer either to a string that receives the current container path or to a buffer for a DISMSGCOUNT or DISSERVICEINFO structure containing the other requested information, depending on iInfo.

The DISMSGCOUNT structure is:

```
typedef struct   _DISMSGCOUNT {
     USHORT cSysMessages; /  Number of pending system messages  /
     USHORT cMail;              /  Number of items in the mail tray  /
     USHORT cNewMail;          New items in mail tray since the prior query  /
} DISMSGCOUNT;
```

The DISSERVICEINFO structure is:

```
#define  DIS_SERVICENAME_LEN        64
typedef struct   _DISSERVICEINFO {
     ULONG idSession;                          /  Current session identifier  /
     CHAR szServiceName[DIS_SERVICENAME_LEN];     / Server name  /
CHAR sz ServiceName[DIS_SERVICENAME_LEN];     /  Network name / }
 DISSERVICEINFO,   PDISSERVICEINFO;
```

pcbBuf	Is a pointer to the size of the pBuf buffer, in bytes. Upon completion, the function returns the actual length of the data it obtained.
fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example retrieves the current container path, then returns without waiting for completion.

```
#define BUFSIZE 1
CHAR   pathBuffer[BUFSIZE];
UINT   bufferSize;
. . .

bufferSize = sizeof (pathBuffer);
result == DisQuerySessionInfo ( DIS_QSI_CURRENTPATH,
     pathBuffer, &bufferSize, DIS_NOWAIT,
    sessionHandle);
```

DisSetCapsuleStatus

This function specifies whether to cancel, purge, hold, release, or modify the specified Capsule application in the Capsule Services queue, or to set various job parameters in a DISCAPSULEJOBPARMS structure.

Desktop Equivalent

If the user were at an IDS workstation, the action equivalent to action code 1 would be to open the Capsule Services icon, select the job to be cancelled, and click on the Cancel button.

There is no workstation action equivalent to the other action codes.

Syntax

INT DISENTRY DisSetCapsuleStatus (UINT *uAction*, ULONG *idJob*, DISCAPSULEJOBPARMS FAR **pjobp*, UINT *fFlags*, HDIS *hdis*);

where:

uAction	Specifies a numerical code representing the action to be applied to the specified Capsule application. Table 20 on page 73 lists the valid codes.

Table 20. Action codes for DisSetCapsuleStatus ()

Action code	Value	Description
DIS_SCS_CANCEL	1	Cancel the Capsule job from the queue. If the job is periodic, and there are more instances of it in the queue, DIS_SCS_CANCEL cancels only the one instance with the earliest scheduled starting time.
DIS_SCS_PURGE	2	Purge a periodic Capsule job.
DIS_SCS_HOLD	3	Hold the job until it is released by the DIS_SCS_RELEASE code.
DIS_SCS_RELEASE	4	Release the held job.
DIS_SCS_MODIFY	5	Modify the Capsule job parameters.

idJob — Is the identifying number of the Capsule application that is to be affected. The DisSubmitCapsule function provides this number.

pjobp — Is a pointer to a buffer for the structure that specifies the job parameters. If you specify an action code other than DIS_SCS_MODIFY, specify NULL for pjobp. The DISCAPSULEJOBPARMS structure is:

```
typedef struct   _DISCAPSULEJOBPARMS {
    USHORT  cb;
     BYTE bPriority;          / Job priority; specify 1 for the lowest priority
                                to 31 for the highest priority  /
     BYTE fbFlags;            / Bit mask relating to when the job is to be run, as
                                listed in Table 5 on page 17  /
     ULONG flMask;            / Bit mask telling which parameters are set in
                                various structure members, as listed in
                                Table 4 on page 16  /
     CHAR FAR  pszDescription;   / Job description  /
     CHAR FAR  pszAliasName;   / Scheduler's alias name  /
     ULONG timeScheduled;       / Scheduled start time  /
     ULONG secsMaxStartDelay;   / Maximum delay past scheduled time,
                                  in seconds  /
     ULONG secsMaxExecution;    / Maximum execution time, in seconds  /
} DISCAPSULEJOBPARMS;
```

The pszDescription and pszAliasName members are pointers to null-terminated strings.

The timeScheduled member is an integer measuring the number of seconds since the start of January 1, 1900.

fFlags — Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

hdis — Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example cancels the previously submitted Budget Capsule application, then returns without waiting for completion:

```
ULONG   jobNumber;
. . .result == DisSubmitCapsule ( "Budget Capsule",
   "Capsule Execution Drawer", NULL, &jobNumber, DIS_NOWAIT,
    sessionHandle);
/  DisSetCapsuleStatus needs the job number that       DisSubmitCapsule sets. /
. . .
result == DisSetCapsuleStatus (DIS_SCS_CANCEL, jobNumber,
     NULL, DIS_NOWAIT, sessionHandle);
```

DisSetCapsuleVars

This function assigns values to the Capsule application's @-variables. You specify the job parameters in a DISCAPSULEVARINFO structure.

Desktop Equivalent

The equivalent action, if the user were at an IDS workstation, would be to select the {Show Controls} button in a Capsule application to open its User Input Controlwindow and type values for one or more @-variables.

Syntax

INT DISENTRY DisSetCapsuleVars (CHAR FAR **pszCapsuleName*, DISCAPSULEVARINFO FAR **pcapv*, UINT *cVars*, UINT *fFlags*, HDIS *hdis*);

where:

pszCapsuleName	Specifies the name of the Capsule application whose variables are to be set.
pcapv	Is a pointer to an array of one or more structures that specify the Capsule application's @-variables. The DISCAPSULEVARINFO structure is:

```
#define DIS_CAPSULEVAR_NAME_LEN       4
#define DIS_CAPSULEVAR_DESC_LEN       5      /  Maximum description length  /
#define DIS_CAPSULEVAR_VALUE_LEN 255              /  Maximum value length  /
#define DIS_CAPSULEVAR_TYPE_DEFAULT ==   / Preserve existing type and
                                                          set new type /
typedef struct    _DISCAPSULEVARINFO {
      CHAR szName[DIS_CAPSULEVAR_NAME_LEN]    /  Variable name  /
      ulType As Long;                                 /  Variable type /
      CHAR FAR  pszDescription;                 /  Variable's description  /
      CHAR FAR  pszValue;                          /  Variable's value  /
} DISCAPSULEVARINFO;
```

	The pszDescription and pszValue members are pointers to null-terminated strings. If either pointer is null, the function does not change the corresponding value or description for the @-variable. If you want to change the @-variable's value, but not change its description, specify pszDescription = NULL and point pszValue to the desired value. To delete either a value or a description, specify a pointer to a zero-length string (pszValue= "" orpszDescription= "").
cVars	Specifies the number of DISCAPSULEVARINFO structures that pcapv points to.
fFlags	Specifies optional flags, as described in "Common Bit Flags" on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_OVERWRITE or DIS_OVERWRITE_VARSflag to replace the Capsule application's current variables instead of merging the new and old lists of variables. For example, assume that the existing list has @A and @B and pcapv specifies only @B. With DIS_OVERWRITE, the resulting variable list has only @B. Without DIS_OVERWRITE, the resulting variable list has the old @A and the new @B.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example sets the @-variables of the Reports Capsule application, then returns without waiting for completion:

```
UINT numVars = 1;
DISCAPSULEVARINFO   capsuleParams
. . .
strcpy (capsuleParams.szName, "A");
capsuleParams.pszValue == "Northeast Region";
capsuleParams.pszDescription == NULL;
result == DisSetCapsuleVars ("Reports Capsule",
      &capsuleParams, numVars, DIS_NOWAIT,
       sessionHandle);
```

DisSetConversionTable

This function modifies the type-conversion table. The system uses this table when copying a personal computer file to a desktop or a desktop object to a personal computer.

You specify each conversion table entry in a *DISCONVERTENTRY* structure.

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

INT DISENTRY DisSetConversionTable (DISCONVERTENTRY FAR **pcnveTable*, UINT *ccnveTable*, UINT *uDefaultFileType*, UINT *fFlags*, HDIS *hdis*);

where:

pcnveTable — Is a pointer to an array of one or more structures that specify entries in the conversion table. The DISCONVERTENTRY structure is:

```
typedef struct   _DISCONVERTENTRY {
     CHAR FAR  pszFileExt;        / File type on the personal computer (characters
                                       after the last period in a file identifier) /
     BYTE bFileType;               / Personal-computer file type; choose a
                                       numerical code from Table 1 on page 12 /
     BYTE bObjType;                / Desktop-object type; choose a numerical code
                                       from Table 2 on page 13 /
     USHORT cchLineWidth;       / Maximum line width when bObjType is 1
                                     (DIS_FILETYPE_TEXT) /
} DISCONVERTENTRY;
```

ccnveTable — Specifies the number of entries in the new conversion table. If you do not specify DIS_OVERWRITE, ccnveTable is the number of entries added to the table.

uDefaultFileType — Specifies the default personal-computer file type to use if a DisPutObject or DisPutMultObjects call specifies a source file whose file-name extension does not correspond to any entry in the conversion table.

fFlags — Specifies optional flags, as described in "Common Bit Flags" on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

Specify the DIS_OVERWRITE flag to replace the current conversion table instead of merely appending the new entries.

hdis — Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example modifies the conversion table so personal-computer files with the .WRD suffix will, when copied to a desktop, be converted to Text documents. The function then returns without waiting for completion:

```
DISCONVERTENTRY   tableEntry;
UNIT                    entries = 1;
. . .tableEntry.pszFileExt = ".WRD";
tableEntry.bFileType = DIS_FILETYPE_TEXT;
tableEntry.bObjType = DIS_OBJTYPE_DOCUMENT;
tableEntry.cchLineWidth = 8 ;
result = DisSetConversionTable (&tableEntry, entries,
    DIS_FILETYPE_TEXT, DIS_NOWAIT, sessionHandle);
```

DisSetCurrentContainer

This function specifies the desktop container on which a future function is to be performed. This specification determines how relative object names are mapped. (Relative object names are names that do not begin with the current path separator, as set by the *DisSetSessionInfo* function or defaulted to the backslash (\) character.)

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

```
INT DISENTRY    DisSetCurrentContainer (CHAR FAR  pszContainerName  ,
    UINT fFlags , HDIS hdis  );
```

where:

pszContainerName	Specifies the name of the new current container. (See "Containers" on page 10.)
fFlags	Specifies optional flags, as described in "Common Bit Flags" on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example makes In Basket the current default container, then returns without waiting for completion:

```
result = DisSetCurrentContainer ( "In Basket", DIS_NOWAIT,       sessionHandle);
```

DisSetEventHandler

This function establishes an event-handling procedure for the specified session. This handler, a procedure in your application, is called by the API when one of the notification events occurs.

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

```
INT DISENTRY DisSetEventHandler (PFNDISEVENTPROC pfnEventProc, ULONG
flEventMask, PDISEVENTpevEventBuf,UINT fFlags, HDIS hdis);
```

where:

pfnEventProc	Is a pointer to the event-handling procedure.

flEventMask — Is some combination of the following flags, which determine the types of events to be passed to the event-handling procedure.

DIS_EVENT_M_FILE 0x0001 File-transfer events

DIS_EVENT_M_REQUEST 0x0002 Request events

DIS_EVENT_M_DATA -x0004 New-data events

pevEventBuf — Is a pointer to the structure to be passed to the event-handling procedure after being filled in by the API. You must initialize the cb field of the structure to the size of the structure. Any data you place in the aulUserParms field will be maintained and passed through to the event-handling procedure. The DISEVENT structure is:

```
typedef struct    DISEVENT {
      USHORT cb;                          / Size of the event, in bytes /
      USHORT usType;                      / Event type /
      USHORT fsFlags;                     / Event flags /
      SHORT err;                          / Event status /
      ULONG aulUserParms [ 4 ] ;
                                          / Space for user/application data /
                                                union {
                                                struct {
            ULONG cbFiles;                / Total size of all files /
            USHORT cFiles;                / Number of files /
                                                } ev_FileListBegin;
          struct {
            ULONG cbFiles;                / Total size of files transferred /
            ULONG msecElapsed;            / Time taken to transfer files /
            USHORT cFiles;                / Number of files transferred /
          } ev_FileListEnd;
          struct {
            ULONG cbFile;                 / Size of file to be transferred /
            USHORT iSequence;             / File sequence number for (multiple file) /
            CHAR szName [ 1 ] ;;
                                          / Name of file (on client) /
          } ev_FileBegin;
          struct {
            ULONG cbFile;                 / Size of file being transferred /
            ULONG cbDone;                 / Amount transferred so far /
            ULONG msecElapsed;            / Time taken to transfer so far /
            USHORT iSequence;             / File sequence number /
          } ev_FileProgress;
          struct {
            ULONG cbFile;                 / Size of file transferred /
            ULONG msecElapsed;            / Time taken to transfer file /
            USHORT iSequence;             / File sequence number /
          } ev_FileEnd;
          struct {
            ULONG idRequest;              / Request ID /
            USHORT usType;                / Request type /
          } ev_RequestBegin;
```

```
        struct  {
              USHORT iSequence;          / Request sequence number (in batch) /
              USHORT usType;             / Request type /
              ULONG idRequest;           / Request ID /
              USHORT usNextType;         / Request type (next batch subrequest) /
        }  ev_RequestEnd;
        struct  {
             DISAFDINFO  afdi;
             ULONG  msecElapsed;
         } ev_AFDInitEnd, ev_AFDExecuteBegin, ev_AFDExecuteEnd;
    }  Parms;
} DISEVENT,  PDISEVENT;
```

Table 21 shows the event codes and values for the usType field of the DISEVENT structure.

Table 21 (Page 1 of 2). Notification events for usType field of DISEVENT structure

Event code	Value	The event handler is called with this event:
DIS_EVENT_FILELIST_BEGIN	1	When a multiple-file transfer (DisGetMultObjects or DisPutMultObjects) begins. Provided: the number and total size of the files to be transferred.
DIS_EVENT_FILELIST_END	2	When a multiple-file transfer (DisGetMultObjects or DisPutMultObjects) ends. Provided: the number and total size of the files transferred, as well as the time that elapsed during file transfer.
DIS_EVENT_FILE_ BEGIN	3	When a file transfer begins, including the individual files in a multiple-file transfer. Provided: the file's size and name. If this is part of a multiple-file transfer, the file index is also provided.
DIS_EVENT_FILE_END	4	When a file transfer ends, including the individual files in a multiple-file transfer. Provided: the file's size and the time that elapsed during file transfer. If this is part of a multiple-file transfer, the file index is also provided.
DIS_EVENT_FILE_ PROGRESS	5	Periodically during a file transfer. Provided: total file size, amount transferred, and time elapsed since the transfer began. If this is part of a multiple-file transfer, the file index is also provided.
DIS_EVENT_NEW_ DATA	6	When a new mail or system message has arrived for the user, or a Capsule job that the user submitted has finished. However, this event occurs only while other requests are being processed. If the application is not making API function calls, this event does not occur until the result of the next API function call is returned by the service.

Table 21 (Page 2 of 2). Notification events for usType field of DISEVENT structure

Event code	Value	The event handler is called with this event:
DIS_EVENT_REQUEST_BEGIN	7	When an API request is sent to the service for processing. Provided: the ID number and type of the request.
DIS_EVENT_REQUEST_END	8	When the result of an API request is returned from the service. Provided: the ID number and type of the request. If the request is one of a batch of requests, the subrequest index and the next subrequest's type are also provided.
DIS_EVENT_AFD_INIT_BEGIN	9	When the AFD initialization phase begins.
DIS_EVENT_AFD_INIT_END	10	When the AFD initialization phase ends. Provided: AFD summary information (DISAFDINFO).
DIS_EVENT_AFD_ EXECUTE_BEGIN	11	When the AFD execution phase begins. Provided: AFD summary information (DISAFDINFO).
DIS_EVENT_AFD_ EXECUTE_END	12	When the AFD execution phase ends. Provided: AFD summary information (DISAFDINFO) and the time that elapsed during the execution phase.

Table 22 shows the event flags in the fsFlags field of the DISEVENT structure:

Table 22. Event flags for fsFlags field of the DISEVENT structure

Event flag		Description
DIS_EVENT_F_RECEIVE	0x0001	A file is being received.
DIS_EVENT_F_SEND	0x0002	A file is being sent.

Table 23 shows the event flags to be used for new-data events (DIS_EVENT_NEW_DATA):

Table 23. Event flags for new-data events

Event flag		Description
DIS_EVENT_F_MAIL	0x0001	New mail is available.
DIS_EVENT_F_MESSAGE	0x0002	A new system message is available.
DIS_EVENT_F_CAPSULE	0x0004	A Capsule job has finished.

fFlags — Must be 0 (zero).

hdis — Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example establishes an event handler to be called when file-transfer events occur.

```
BYTE Buf [ x1   ];
PDISEVENT pev;
pev == ( PDISEVENT ) Buf;
pev -> cb == sizeof (Buf);
result == DisSetEventHandler (EventProc, DIS_EVENT_M_FILE, pev,  ,
   sessionhandle);
```

DisSetPassword

This function changes the password of the connected desktop. The session remains logged on.

Desktop Equivalent

The equivalent action, if the user were at an IDS workstation, would be to select the desktop's Change Password button and enter a new password.

Syntax

INT DISENTRY DisSetPassword (CHAR FAR **pszOldPassword*, CHAR FAR **pszNewPassword*, UINT *fFlags*, HDIS *hdis*);

where:

pszOldPassword	Specifies the current password.
pszNewPassword	Specifies the new password.
fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example assumes that the program has queried the user for the values of oldPassword and newPassword. It returns without waiting for completion.

```
CHAR FAR  oldPassword, newPassword;
. . .
result == DisSetPassword ( oldPassword, newPassword,
       DIS_NOWAIT, sessionHandle );
```

DisSetSessionInfo

This function either changes the separator character to be used in path names on the desktop, or sets the IBM code-page number.

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

INT DISENTRY DisSetSessionInfo (UINT *iInfo*, VOID FAR **pBuf*, UINT *cbBuf*, UINT *fFlags*, HDIS *hdis*);

where:

iInfo — Is the selection code, specifying which information you want to change. Table 24 lists the valid values.

Table 24. Information selection codes for DisSetSessionInfo ()

Selection code	Value	Description
DIS_SSI_CODEPAGE	1	The function specifies the code-page number in use by the personal computer.
DIS_SSI_PATHSEPCHAR	2	The function specifies the separator character, to be used in path names, to any other printable ASCII character. The default character is a backslash (\ ).

pBuf — Is a pointer to a buffer containing either the code-page number or the separator character, depending on iInfo.

cbBuf — Is the size of the pBuf buffer, in bytes.

fFlags — Specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

hdis — Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Examples

This example sets the path separator to the vertical bar (|) then returns without waiting for completion.

```
char    separator = '|';
...
result == DisSetSessionInfo (DIS_SSI_PATHSEPCHAR, &separator,
      sizeof(separator),
       DIS_NOWAIT,  sessionHandle);
```

The next example is another acceptable way to set the path separator.

```
result = DisSetSessionInfo (DIS_SSI_PATHSEPCHAR, “|”, 1,
       DIS_NOWAIT, sessionHandle );
```

DisSubmitBatch

This function submits queued functions for execution. The DisInitBatch function starts a queue of submitted functions. All subsequent functions are queued until DisSubmitBatch is reached; at that point, the queued functions are submitted to the ODAS server for execution.

Each function in the batch can return errors due to local conditions (such as a missing local file in a DisPutObject call). In this situation, you can use the DisTermBatch command to terminate the batch instead of submitting it to the ODAS server. (See "DisTermBatch" on page 88.)

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

INT DISENTRY DisSubmitBatch (INT FAR **pjResults*, UINT *cResults*, UINT *fFlags*, HDIS *hdis*);

where:

pjResults	Is a pointer to the buffer that is to receive the results of the functions that this operation submits.
cResults	Specifies the number of entries in the array that is to be passed to pjResults.
fFlags	Specifies optional flags, as described in "Common Bit Flags" on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_SB_SUBRESULTSflag to retrieve the results of each subrequest as soon as the server has completed the request. Otherwise, the function retrieves results when all subrequests are completed.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example enables, creates, and submits a batch request comprising four desktop operations. The result codes of DisGetObject, DisGetSysMessage, DisCopyObject, and DisLogoff are returned in batchResults.

```
INT   batchResults[4];
. . .
result == DisInitBatch ( , sessionHandle);
    DisGetObject ( "Next Report", "NEXTRPT.WKS",
        DIS_FILETYPE_WKS, NULL,  ,
        DIS_DELETE_SOURCE | DIS_WAIT, sessionHandle);
    DisGetSysMessage (&numMessages, msgBuffer,
        &bufferSize,  , sessionHandle);
    DisCopyObject ( "Next Report", "Copy of Next Report",
        DIS_OBJTYPE_DEFAULT, DIS_NOWAIT, sessionHandle);
    DisLogoff ( , sessionHandle);
result == DisSubmitBatch ( batchResults, 4, DIS_NOWAIT,
     sessionHandle  );
```

DisSubmitCapsule

This function submits a copy of the specified Capsule application to the Capsule Services, which then runs the copy once or periodically, according to the schedule you give it. If you want the results to replace the original Capsule application, use the DIS_OVERWRITE flag.

You specify the DisSubmitCapsule job parameters in a DISCAPSULEJOBPARMS structure.This function returns as soon as the Capsule job has been queued for execution. It does not wait for the job to be executed. You can use DisWaitCapsuleStatusto wait for a Capsule job to complete execution.

Desktop Equivalent

The equivalent action, if the user were at an IDS workstation, would be to copy the Capsule icon onto the closed Capsule Services icon.

Syntax

INT DISENTRY DisSubmitCapsule (CHAR FAR **pszCapsuleName*, CHAR FAR **pszDestContainerName*, DISCAPSULEJOBPARMS FAR **pjobp*, ULONG FAR **pidJob*, UINT *fFlags*, HDIS *hdis*);

where:

pszCapsuleName	Is the name of the Capsule application to be submitted.
pszDestContainerName	Is the name of the container into which the Capsule application is to be copied after execution. (See “Containers” on page 10.)To discard the executed copy of the Capsule application after it completes execution, specify NULL.
pjobp	Is a pointer to a buffer for the structure that specifies the job parameters. The DISCAPSULEJOBPARMS structure is:

```
typedef struct    _DISCAPSULEJOBPARMS {
     USHORT  cb;
      BYTE bPriority;                / Job priority; specify 1 for the lowest priority
                                          to 31 for the highest priority  /
      BYTE fbFlags;                   / Bit mask relating to when the job is to be run, as
                                          listed in Table 5 on page 17  /
      ULONG flMask;                    / Bit mask telling which parameters are set in
                                          various structure members, as listed in
                                          Table 4 on page 16  /
      CHAR FAR  pszDescription;    / Job description  /
      CHAR FAR  pszAliasName;       / Scheduler's alias name  /
      ULONG timeScheduled;           / Scheduled start time  /
      ULONG secsMaxStartDelay;     / Maximum delay past scheduled time,
                                                    in seconds  /
    ULONG secsMaxExecution;         / Maximum execution time, in seconds  /
} DISCAPSULEJOBPARMS;
```

The pszDescription and pszAliasName members are pointers to null-terminated strings.

The timeScheduled member is an integer measuring the number of seconds since the start of January 1, 1900.

pidJob
Is a pointer to the variable that is to receive the job's identification number.

fFlags
Specifies optional flags, as described in “Common Bit Flags” on page 14.

You can specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. However, you should not use DIS_NOWAIT if your goal is only to prevent your application from waiting for the Capsule application to finish processing. DisSubmitCapsule returns as soon as the Capsule application has been queued for execution.

Specify DIS_OVERWRITE to cause the results to replace the original Capsule application.

hdis
Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example submits the Budget Capsule application to be copied to the Capsule Execution Drawer container and then executed; it then waits for completion.

```
ULONG                   jobNumber;
DISCAPSULEJOBPARMS   jobParams
. . .
/  Reset all members of the structure to zero.  /
memset (&jobParams,  , sizeof (jobParams));
jobParams.cb == sizeof (jobParams);
jobParams.pszAliasName == “OVERNIGHT”;
jobParams.flMask == DIS_CJP_ALIASNAME
result == DisSubmitCapsule ( “Budget Capsule”,
      "Capsule Execution Drawer", &jobParams, &jobNumber,
      DIS_WAIT, sessionHandle );
```

DisSubmitCapsuleAndSetVars

This function submits a Capsule job to the Capsule Services and sets the @-variables atomically.

The Submit Capsule/Set @-Variables operation is equivalent to the sequence of calls: DisCopyObject, DisSetCapsuleVars, DisSubmitCapsule,andDisDeleteObject, with the addition that the operation is performed atomically and does not require the extra copy/delete operations. Additionally, a new name may be specified for the completed Capsule.

This API provides a safe means by which shared Capsule applications may be submitted without the danger of @-variables being modified by another simultaneous *submit* operation. This API does not modify the @-variables in the original source Capsule. The @-variables are set in a unique copy of the Capsule submitted to the Capsule Service.

DisSubmitCapsuleAndSetVars must be used in cases where the source Capsules reside in a shared directory.

Desktop Equivalent

The equivalent action, if the user were at an IDS workstation, would be to select the Show Controls button in a Capsule application to open its User Input Controlwindow and type values for one or more @-variables, then close the Capsule icon and copy the Capsule icon onto a closed Capsule Services icon.

Syntax

*INT DISENTRY DisSubmitCapsuleAndSetVars (CHAR FAR *pszCapsuleName, CHAR FAR *pszDestContainerName, CHAR FAR *pszCapsuleNameOut, DISCAPSULEJOBPARMS FAR *pjobp, ULONG FAR *pidJob, DISCAPSULEVARINFO FAR *pcapv, UINT cVars, UINT fFlags, HDIS hdis);*

where:

pszCapsuleName	Is the name of the Capsule application to be submitted.

pszDestContainerName — Is the name of the container into which the Capsule application is to be copied after execution. (See “Containers” on page 10.)To discard the executed copy of the Capsule application after it completes execution, specify NULL.

pszCapsuleNameOut — Is the name of the completed Capsule application.

pcaps — Is a pointer to a Capsule application's parameters.

pidJob — Is a pointer to the variable that is to receive the job's identification number.

pcapv — Is a pointer to an array of one or more structures that specify the Capsule application's @-variables. The DISCAPSULEVARINFO structure is:

```
#define DIS_CAPSULEVAR_NAME_LEN      4
#define DIS_CAPSULEVAR_DESC_LEN   5     / Maximum description
length /
#define DIS_CAPSULEVAR_VALUE_LEN 255            / Maximum value
length /
#define DIS_CAPSULEVAR_TYPE_DEFAULT == / Preserve
existing type and
          set new type /
typedef struct   _DISCAPSULEVARINFO {
    CHAR  szName[DIS_CAPSULEVAR_NAME_LEN]            /
Variable name /
     ulType As Long;/ Variable type /
     CHAR FAR  pszDescription;    / Variable's description /
     CHAR FAR  pszValue;       / Variable's value /
} DISCAPSULEVARINFO;
```

cVars — Specifies number of @-variable entries.

fFlags — Specifies optional flags, as described in “Common Bit Flags” on page 14.

You can specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. However, you should not use DIS_NOWAIT if your goal is only to prevent your application from waiting for the Capsule application to finish processing. DisSubmitCapsule returns as soon as the Capsule application has been queued for execution.

Specify DIS_OVERWRITE to cause the results to replace the original Capsule application. Specify DIS_OVERWRITE_VARS to delete all existing @-variables and replace them with only those provided in this API call.

hdis — Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

DIS_SC_NOTIFY_MAIL

The DIS_SC_NOTIFY_MAIL flag causes Global Access to send a mail envelope to the desktop's mail tray when the Capsule completes execution.

Example

For examples of how to submit and set the Capsule variables, see “DisSetCapsuleVars” on page 74 and “DisSubmitCapsule” on page 84.

DisTermAFD

This function terminates an AFD session. DisTermAFD must be called to free the resources that DisInitAFD allocated.

If your application must execute AFD operations conditionally, or if you want to specify how much data is to be transferred before actually confirming the action, you should use the individual functions DisInitAFDList, then DisExecuteAFD, then DisTermAFD. Otherwise, you can use DisAFD or DisAFDList, each of which calls three functions: DisInitAFDList, then DisExecuteAFD, then DisTermAFD.

If you do not want to perform the AFD operations identified by DisInitAFD (or DisInitAFDList), you may call DisTermAFD to terminate AFD processing. If you do want those operations performed, call DisExecuteAFD to perform them, then call DisTermAFD to free the AFD processing resources.

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

INT DISENTRY DisTermAFD (UINT *fFlags*, HDIS *hdis*);

where:

fFlags	Must be 0 (zero).
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example terminates the specified session.

```
result = DisTermAFD ( , sessionHandle );
```

DisTermBatch

This function terminates a batch submission, discarding any calls that have not yet been submitted to the ODAS server. (See “DisInitBatch” on page 58,and “DisSubmitBatch” on page 83.)

Each function in the batch can return errors due to local conditions (such as a missing local file in a DisPutObject call). In this situation, you can use the DisTermBatch command to terminate the batch instead of submitting it to the ODAS server.

Desktop Equivalent
There is no equivalent action for a user at an IDSworkstation.

Syntax
INT DISENTRY DisTermBatch (UINT *fFlags*, HDIS *hdis*);

where:

fFlags	Must be 0 (zero).
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example
This example enables, creates, and either submits or terminates a small batch request, depending on the results of *DisInitBatch*

```
result = DisInitBatch (
_, sessionHandle );
    result == DisGetSysMessage ( &numMessages, msgBuffer,
          &bufferSize,  , sessionHandle );
    result == DisGetObject ( "Next Report", "NEXTRPT.WKS",
          DIS_FILETYPE_WKS, NULL,  ,
          DIS_DELETE_SOURCE | DIS_WAIT, sessionHandle );

if (result ==  ) {
    result == DisSubmitBatch ( batchResults, 2, DIS_NOWAIT,
        sessionHandle  );
} else {
    result == DisTermBatch (  , sessionHandle );
```

DisTermClient
This function releases all resources allocated for this ODAS application. Use it at the end of the application.

Desktop Equivalent
There is no equivalent action for a user at an IDSworkstation.

Syntax
INT DISENTRY DisTermClient (UINT *fFlags*);

where:

fFlags	Reserved for future use; specify 0 (zero).

Example
This example releases API resources:

```
result = DisTermClient (  );
```

DisWaitCapsuleStatus

This function pauses the personal-computer application until the specified Capsule application achieves the specified status.

Desktop Equivalent

There is no equivalent action for a user at an IDSworkstation.

Syntax

INT DISENTRY DisWaitCapsuleStatus (ULONG *idJob*, UINT *uStatus*, ULONG *ulTimeout*, LONG FAR **plCompCode*, UINT *fFlags*, HDIS *hdis*);

where:

idJob	Is the identifying number of the Capsule application whose status is to be checked. The DisSubmitCapsule function provides this number.
uStatus	Specifies a numerical code representing the status to be achieved by the specified Capsule application. Currently, the only valid value is DIS_WCS_OUTPUT, or 1, which waits until the Capsule job is completed or abnormally ended.
ulTimeout	Specifies the maximum number of seconds to wait for the Capsule application to achieve the specified status. Specify –1 to wait forever.
plCompCode	Is a pointer to a variable that receives the Capsule application's completion code (after that application has reached the output queue).
fFlags	Specifies optional flags, as described in “Common Bit Flags” on page 14.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example pauses execution until the Capsule job named *Budget Capsule* is completed or abnormally ended:

```
LONG   jobNumber;
LONG   completion
. . .
result == DisSubmitCapsule ( "Budget Capsule",
      "Capsule Execution Drawer", NULL, &jobNumber, DIS_WAIT,
     sessionHandle);
/  DisWaitCapsuleStatus needs the job number that       DisSubmitCapsule sets. /
. . .
result == DisWaitCapsuleStatus ( jobNumber, DIS_WCS_OUTPUT, –1,
     &completion,      DIS_WAIT,  sessionHandle  );
```

Chapter 5. Using the Visual Basic API Functions

To provide the end user with the most robust access to the Intelligent Decision Serverm, your application program must perform a variety of tasks. Of course, you can also write a specialized program to perform a specific function. This chapter shows how to use the functions described in Chapter 6, “API Functions for Visual Basic Programming” on page 112, to enable these tasks.

The code fragments in this chapter are abridged; Appendix D, “Sample Program Code with Visual Basic API Functions” on page 195, shows a short, sample application program.

Starting ODAS

Before calling any of the ODAS Visual Basic functions, you must start the TCP/IP communications software, as described in the TCP/IP documentation. You can either require the user to start this software interactively or automatically with a command in your application.

Preventing Command Clashes

In a Windows application, if the user clicks a button before the previous activity finishes, the second ODAS call can interfere with the first one, due to the asynchronous Windows message system. One way to prevent this is to disable all the buttons that call ODAS functions until the currently active function returns its results.

Connecting to a Desktop

To connect to a desktop, use the DisInitClient, DisConnect, and DisLogon functions. Your program should:

Call the DisInitClient function to initialize the end user's workstation

Call the DisConnect function, specifying the name of the ODAS server

Call the DisLogon function to access a developer's desktop

The following code samples demonstrate how to connect to a desktop. In this example, DISLANCOMM is a structure type determined by DIS_COMM_LAN, as described in “DisConnect” on page 124.

1. Declare the DISLANCOMM parameters structure and any other necessary variables.

   ```
   Dim lanComm As DISLANCOMM
   Dim retCode As Integer
   Dim bufferSize As Integer
   Dim disBuffer As Long
   ```

2. Initialize the client on which the application is to run.

© Copyright IBM Corp. 1996

```
retCode = DisInitClient( )
```

3. In the parameters structure, set the size of the structure and, optionally, the desktop name and password. You can assume most of these parameters from the default values that are set on the validation diskette by the server's administrator. However, this example assumes that the associated form contains ComboBox or TextBox items named commPort, baudRate, commTimeOut, phoneNumber, userName, and password, where the user can type or select the appropriate information for the ODAS server that is to be accessed.

```
Select Case Val(myOdasServerName.Text)
Case "Marketing"
    lan.pszServerName = "Sales"
Case 12
    lan.pszServerName = "Marketing"
...
```

4. Call DisCreateBuffer to allocate and lock a buffer to be used by DisConnect.

```
Screen.MousePointer = 11
disBuffer = DisCreateBuffer(DISLANCOMM_FORMAT, 1, bufferSize)
```

5. Call DisCFormatSize to find the size of the lanComm structure.

```
lanComm.cb = DisCFormatSize(DISLANCOMM_FORMAT,
DISUB_FIXED_LEN_SIZE)
```

6. Call DisVBTypeToBuffer to copy the data from the lanComm structure into the disBuffer structure.

```
retCode = DisVBTypeToBuffer(disBuffer, bufferSize,
lanComm, Len(lanComm), DISLANCOMM_FORMAT)
```

7. Call DisConnect to dial the telephone number and connect to the LAN.

```
retCode = DisConnect(DIS_COMM_LAN, ByVal disBuffer,
DIS_WAIT, sessionHandle)
```

8. Call DisLogon to access the desktop.

```
retCode = DisLogon(userName.Text, password.Text,  ,
sessionHandle)
```

9. Free the space used by disBuffer.

```
retCode = DisFreeBuffer(disBuffer)
Screen.MousePointer = 11
```

Disconnecting from a Desktop

To end a desktop connection, use the DisLogoff, DisDisconnect, and DisTermClient functions. Your program should:

Log off the desktop

End the connection to the LAN

Release any resources

The following code samples demonstrate how to terminate access to a desktop:

1. Call DisLogoff, to leave the desktop.

```
Screen.MousePointer = 11
retCode = DisLogoff( , sessionHandle)
```

2. Check the return code and display the result.

```
If retCode =   Then
   MsgBox 'Logging off DIS.',
   MB_OK + MB_ICONINFORMATION, 'ODAS'
Else
   MsgBox 'Failed to log off DIS.',
   MB_OK + MB_ICONEXCLAMATION, 'ODAS'
End If
```

3. Call DisDisconnect to end the connection to the network.

```
retCode = DisDisconnect(DIS_WAIT, sessionHandle)
```

4. Check the return code and display the result.

```
If retCode =   Then
   MsgBox 'Disconnected from DIS.',
   MB_OK + MB_ICONINFORMATION, 'ODAS'
Else
   MsgBox 'Failed to disconnect from DIS.',
   MB_OK + MB_ICONEXCLAMATION, 'ODAS'
End If
Screen.MousePointer =
```

5. Call DisTermClient to clean up loose ends.

```
retCode = DisTermClient( )
```

Viewing the Mail Tray

One use of ODAS is to let users exchange mail. Your program should:

Display the list of mail items to let the user choose which items to download

For each selected mail item, download that item and any enclosures it might have

Listing the Mail

To check the contents of the Mail Tray, use the DisQueryObjectInfo function. For each mail Envelope, your program should call DisQueryObjectInfo to retrieve information such as the Envelope's name, timestamp, and size, and show that information to the user.

The following code samples demonstrate how to list the contents of the Mail Tray. In this example, vbBuffer is a pointer to a structure of the type DISOBJECTINFO, as described in “DisQueryObjectInfo” on page 151.

1. Declare the DISOBJECTINFO parameters structure and other necessary variables.

```
Dim disBuffer As Long
Dim vbBuffer() As DISOBJECTINFO
Dim bufferSize As Integer
Dim index As Integer
Dim numMessages As Integer
Dim retCode As Integer
```

2. Call DisQueryObjectInfo with the DIS_QOIL_TEST code and the DIS_QOI_COUNT_CONTENTS flag to see if there is any mail in the desktop's Mail Trail container.

```
bufferSize =
retCode = DisQueryObjectInfo('Mail Tray\', DIS_QOIL_TEST,
    , numMessages,  , bufferSize,
DIS_QOIL_COUNT_CONTENTS, sessionHandle)
```

3. If there is mail (retCode > 0), call DisCreateBuffer to allocate and lock a buffer to be used by DisQueryObjectInfo.

```
disBuffer = DisCreateBuffer(DISOBJECTINFO_FORMAT,
  numMessages,  bufferSize)
```

4. Call DisQueryObjectInfo with the DIS_QOIL_STANDARD code to retrieve the list of mail items. This list shows the same information that a workstation user can see in the Mail Tray container.

```
Screen.MousePointer = 11
retCode = DisQueryObjectInfo
   ( 'Mail Tray\ ',              ' Match all items.
   DIS_QOIL_STANDARD,            ' Return standard information.
   index,                        ' starting index
   numMessages,                  ' item count
   disBuffer,                    ' buffer pointer
   bufferSize,                   ' buffer size
   DIS_WAIT,                     ' Wait for completion.
  sessionHandle)
Screen.MousePointer =
```

5. Call DisBufferToVBArray to copy the data from the disBuffer structure into the vbBuffer array of structures.

```
ReDim vbBuffer(1 To numMessages)
retCode = DisBufferToVBArray(vbBuffer(1),
   Len(vbBuffer(1)), disBuffer, bufferSize,
  DISOBJECTINFO_FORMAT,  numMessages)
```

6. Loop through the contents of the Mail Tray. In each iteration, display the header of a mail message in a list box named listMail.

```
For i = 1 To numMessages
    listMail.AddItem  vbBuffer(i).szName
Next i
```

7. Free the space used by disBuffer, because vbBuffer now holds the message data.

```
retCode = DisFreeBuffer(disBuffer)
```

Retrieving an Envelope from the Mail Tray

To copy a selected mail item to the client, the most important function is DisGetObject. The following code samples demonstrate how to retrieve the mail item from the desktop to the client, where the user can read it. This example does not retrieve any enclosures of that mail item, however.

1. Assuming that the available mail is listed as in the previous example, set the source-file name to be the mail item that the user has selected from the list box.

   ```
   Screen.MousePointer = 11
   sourceFile = '\Mail Tray\' & List(listMail.ListIndex)
   ```

2. Assuming that your form includes a file list box and a directory list box, insert code into each list box's Click procedure to assemble the full path of the destination file. By displaying this path in a text box, you give the user the option to enter a new name, overruling the directory- and file-box selections.

   ```
   txtClientFile.Text = dirList.Path & fileList.FileName
   ```

3. Call DisGetObject to retrieve the Envelope, converting it into TEXT format (unformatted ASCII text with spaces simulating the column spacing, paragraph formatting, and tabs of the Envelope).

   ```
   retCode = DisGetObject(
      sourceFile,                    ' selected mail
      txtClientFile.Text          ' destination for incoming mail
      DIS_FILETYPE_TEXT,             ' file type on client
       ,                             ' retrieve the entire message
       ,                             ' no maximum line width
      DIS_WAIT,                      ' Wait for completion.
     sessionHandle)
   Screen.MousePointer =
   ```

Retrieving all Enclosures of an Envelope

To retrieve enclosures, use the DisSetConversionTable and DisGetMultObjects functions.

The following code samples demonstrate how to retrieve the mail items from the desktop to the client, where the user can read them. In this example, convTableIn is an array of structures of the type DISCONVERTENTRY, as described in "DisSetConversionTable" on page 162.

1. Declare pointers to the DISCONVERTENTRY parameters structure, with other necessary variables.

   ```
   Dim retCode As Integer
   Dim vbConvertEntry() As DISCONVERTENTRY
   Dim bufferSize As Integer
   Dim tableEntry As Integer
   Dim enclList As String
   ```

2. Call DisSetConversionTable to set the conversion table so it will convert all spreadsheets to Data Interchange Format (DIF) and all text objects to TEXT format.

```
ReDim vbConvertEntry(1 To 2)
Screen.MousePointer = 11
vbConvertEntry(1).bFileType = DIS_FILETYPE_DIF
vbConvertEntry(1).bObjType = DIS_OBJTYPE_SPREADSHEET2
vbConvertEntry(2).bFileType = DIS_FILETYPE_TEXT
vbConvertEntry(2).bObjType = DIS_OBJTYPE_DOCUMENT
vbConvertEntry(2).cchLineWidth = LINEWIDTH
entries = 2
tableEntry = DisCreateBuffer(DISCONVERTENTRY_FORMAT,
  entries,  bufferSize)
retCode = DisBufferToVBType(vbConvertEntry(1),
   Len(vbConvertEntry(1)), tableEntry, bufferSize,
  DISCONVERTENTRY_FORMAT)
retCode = DisSetConversionTable(tableEntry, entries,
   DIS_FILETYPE_TEXT, DIS_NOWAIT, sessionHandle)
```

3. Free the space used by tableentry.

```
retCode = DisFreeBuffer(tableEntry)
```

4. Assuming that the available mail is listed as in the previous examples, use the wildcard (*) to construct a string to specify all of the user-selected Envelope's enclosures.

```
enclList = '\Mail Tray\' & List(listMail.ListIndex) & '\ '
```

5. Insert code into each list box's Click procedure to assemble the full path of the destination files, letting the user enter a new name as in the previous example.

```
txtClientFile.Text = dirList.Path & fileList.FileName
```

6. Call DisCreateBuffer to allocate and lock a buffer to be used by DisGetMultObjects.

```
numObjects = 1          ' Retrieve as many as 1  enclosures.
disBuffer = DisCreateBuffer(DISGETOBJECTINFO_FORMAT,
  numObjects,  bufferSize)
```

7. Call DisGetMultObjects to retrieve the enclosures, converting each enclosure as specified in the conversion table.

```
retCode = DisGetMultObjects(enclList,
   txtClientFile.Text, numObjects, disBuffer,
   bufferSize, DIS_WAIT Or DIS_DELETE_SOURCE,
  sessionHandle)
Screen.MousePointer =
```

8. Free the space used by disBuffer.

```
retCode = DisFreeBuffer(disBuffer)
```

Sending Mail

To send mail to the Mail Tray, use the DisPutMultObjects or DisPutObject function. The code to send mail without enclosures is somewhat simpler than the code for enclosures. In each case, your program should:

Ensure that each mail item exists as a properly formatted text file on the client

Copy each message file to the Mail Tray

One Item, without Enclosures

To send one mail message that has no enclosures, use the DisPutObject function. The following code samples demonstrate how to send mail files from the client to the desktop Mail Tray.

1. Declare the necessary variables.

   ```
   Dim retCode As Integer
   ```

2. Assuming that your form includes a file list box and a directory list box, insert code into each list box's Click procedure to assemble the full specification of the source file.

   ```
   txtClientFile.Text = dirList.Path & fileList.FileName
   ```

3. Call DisPutObject to send the message file to the Mail Tray.

   ```
   retCode = DisPutObject(txtClientFile.Text,
      DIS_FILETYPE_TEXT,  , '\Mail Tray',
     DIS_OBJTYPE_ENVELOPE,
      DIS_DELETE_SOURCE Or DIS_WAIT, sessionHandle)
   ```

Two or More Items, without Enclosures

To send two or more mail messages that have no enclosures, use the DisPutMultObjects function. The following code samples demonstrate how to send mail files from the client to the desktop Mail Tray. In this example, convTableOut is an array of structures of the type DISCONVERTENTRY, as described in "DisSetConversionTable" on page 162.

1. Declare pointers to the DISCONVERTENTRY parameters structure, with other necessary variables.

   ```
   Dim retCode As Integer
   Dim vbConvertEntry As DISCONVERTENTRY
   Dim bufferSize As Integer
   Dim tableEntry As Integer
   ```

2. Call DisSetConversionTable to set the conversion table to convert text files to Envelopes.

   ```
   Screen.MousePointer = 11
   vbConvertEntry.bFileType = DIS_FILETYPE_TEXT
   vbConvertEntry.bObjType = DIS_OBJTYPE_DOCUMENT
   vbConvertEntry.cchLineWidth = LINEWIDTH
   entries = 1
   tableEntry = DisCreateBuffer(DISCONVERTENTRY_FORMAT,
     entries,  bufferSize)
   retCode = DisBufferToVBType(vbConvertEntry(1),
      Len(vbConvertEntry(1)), tableEntry, bufferSize,
     DISCONVERTENTRY_FORMAT)
   retCode = DisSetConversionTable(tableEntry, entries,
      DIS_FILETYPE_TEXT, DIS_NOWAIT, sessionHandle)
   retCode = DisFreeBuffer(tableEntry)
   ```

3. Assuming that your form includes a file list box and a directory list box, insert code into each list box's Click procedure to assemble the full specification of the source

files. By displaying this path in a text box, you give the user the option to enter a wildcard specification, overruling the directory- and file-box selections.

```
txtSourceFile.Text = dirList.Path & fileList.FileName
```

4. Call DisPutMultObjects to send the message files to the Mail Tray.

```
retCode = DisPutMultObjects(txtSourceFile.Text,
   '\Mail Tray', DIS_DELETE_SOURCE Or DIS_WAIT,
  sessionHandle)
Screen.MousePointer =
```

One Item, with Enclosures

To send a mail message and enclose other messages, spreadsheets, or other files in the message, use the DisPutMultObjects function. The following code samples demonstrate how to send a mail file with enclosures from the client to the desktop Mail Tray. In this example, convTableOut is an array of structures of the type DISCONVERTENTRY, as described in “DisSetConversionTable” on page 162.

1. Send the enclosing message file to the IDS desktop, as described in “One Item, without Enclosures” on page 97,except that you send it to a container other than the Mail Tray (or to the top level of the desktop). This indirect destination is necessary to let you put the enclosures into the new Envelope before mailing it.

```
retCode = DisPutObject(txtClientFile.Text,
   DIS_FILETYPE_TEXT,  ,   '\',              ' Send it to the top level of the desktop.
  DIS_OBJTYPE_ENVELOPE,
   DIS_DELETE_SOURCE Or DIS_WAIT, sessionHandle)
```

2. Send the enclosures as described in “Two or More Items, without Enclosures” on page 97,except that you send them to the Envelope you just created (newEnvelope), instead of to the Mail Tray.

```
newEnvelope = '\' & txtClientFile.Text
retCode = DisPutMultObjects(txtSourceFile.Text,
   newEnvelope, DIS_DELETE_SOURCE Or DIS_WAIT,
  sessionHandle)
```

3. Call DisMoveObject to mail the Envelope (and its new enclosures).

```
retCode = DisMoveObject(newEnvelope, '\Mail Tray\',
  DIS_NOWAIT,  sessionHandle)
```

s

There is no need to specify a meaningful destination name in the DisMoveObject call, because IDS mail always gets a new name based on its From and Subject header lines.

Moving Files to the Desktop

To move one or more files to the LAN, use the DisPutMultObjects or DisPutObject function. Your program should:

Ensure that each file exists, properly formatted, on the client

Specify the appropriate conversion

Call DisPutObject (or DisPutMultObjects) to copy the file (or files) into the container

The following code samples demonstrate how to copy one WKS-format spreadsheet file to a IDS desktop, automatically converting it to a Spreadsheet II object.

1. Declare the necessary variables.

```
Dim retCode As Integer
Dim vbConvertEntry(1 To 2) As DISCONVERTENTRY
Dim bufferSize As Integer
Dim tableEntry As Integer
```

2. Call DisSetConversionTable to set the conversion table so it will convert all spreadsheets to Data Interchange Format (DIF) and all text objects to TEXT format.

```
Screen.MousePointer = 11
vbConvertEntry(1).bFileType = DIS_FILETYPE_DIF
vbConvertEntry(1).bObjType = DIS_OBJTYPE_SPREADSHEET2
vbConvertEntry(2).bFileType = DIS_FILETYPE_TEXT
vbConvertEntry(2).bObjType = DIS_OBJTYPE_DOCUMENT
vbConvertEntry(2).cchLineWidth = LINEWIDTH
entries = 2
tableEntry = DisCreateBuffer(DISCONVERTENTRY_FORMAT,
  entries,  bufferSize)
retCode = DisBufferToVBType(vbConvertEntry(1),
   Len(vbConvertEntry(1)), tableEntry, bufferSize,
  DISCONVERTENTRY_FORMAT)
retCode = DisSetConversionTable(tableEntry, entries,
   DIS_FILETYPE_TEXT, DIS_NOWAIT, sessionHandle)
retCode = DisFreeBuffer(tableEntry)
```

s

3. Assuming that your form includes a file list box and a directory list box, insert code into each list box's Click procedure to assemble the full path of the source file or files.

```
txtSourceFile.Text = dirList.Path & fileList.FileName
```

4. Construct the target path to specify the desired desktop container and object name.

```
txtClientFile.Text = txtContainer.Text & '\' &
  List(listContainers.ListIndex)
```

5. Call DisPutObject to send the local file to the IDS desktop.

```
retCode = DisPutObject(txtSourceFile.Text,
   DIS_FILETYPE_TEXT,  , txtClientFile.Text,
  DIS_OBJTYPE_DEFAULT,
  DIS_DIS_WAIT,  sessionHandle)
Screen.MousePointer =
```

Retrieving Files from the Desktop

To retrieve one or more files from the LAN, use the DisGetMultObjects or DisGetObject function. Your program should:

Specify the appropriate conversion

Call the DisGetObject (or DisGetMultObjects) function to copy the file (or files) into the container

The following code samples demonstrate how to copy one Spreadsheet object from a IDS desktop, automatically converting it to a WKS-format spreadsheet file.

1. Declare the necessary variable.

   ```
   Dim retCode As Integer
   ```

2. Call DisGetObject to retrieve the spreadsheet, converting it into DIF format. Remember that the following sample would be entered on a single line.

   ```
   Screen.MousePointer = 11
   retCode = DisGetObject(
     txtSourceObject.Text,
     txtClientFile.Text
     DIS_FILETYPE_WKS,
      ,
      ,
     DIS_WAIT,
     sessionHandle)
   Screen.MousePointer =
   ```

Using Capsule Applications

For effective remote use of capsule applications, the end user should be able to list the available applications, set @-variables in the selected application, run the application, check its progress, and alter its queue status.

For example, you can create a window that lets the user select a capsule application, view and alter @-variables in it, and submit it for execution. You might also want to let the user schedule a time or time alias for running the job and select various options such as the maximum execution time or maximum delay time.

Setting Capsule Variables

To assign values to a capsule application's @-variables, use the DisSetCapsuleVars function. The following code samples demonstrate how to do this. In this example, atVars is an array of structures of the type DISCAPSULEVARINFO, as described in “DisSetCapsuleVars” on page 159.

This example assumes that you set up a grid control (grdAtVar) to display the selected capsule application's @-variables and to let the user select @-variables to be changed.

You cannot use @ in the ODAS programming. Specify “A” for @A, and so forth.

1. Declare the DISCAPSULEVARINFO structure and other necessary variables. This example assumes that no capsule application will have more than 26 @-variables.

```
Dim retCode As Integer
Dim startVar As String
Dim numVars As Integer
Dim bufferSize As Integer
Dim disBuffer As Long
Dim atVars() As DISCAPSULEVARINFO

startVar = 'A' & Chr$( )
numVars = 26
```

2. List the available capsule applications, as described in “Listing the Mail” on page 93, except that you list the contents of a container other than the Mail Tray (or list the top level of the desktop).

   This example specifies a container to restrict the user's access to a certain set of capsule applications. If you prefer, you can let the user specify a container.

3. Call DisCreateBuffer to allocate and lock a buffer to be used by DisGetCapsuleVars.

```
Screen.MousePointer = 11
disBuffer = DisCreateBuffer(DISCAPSULEVARINFO_FORMAT,
  numVars,  bufferSize)
```

4. Call DisGetCapsuleVars to retrieve the current @-variables of the capsule application the user has selected in the ListCaps box.

```
retCode = DisGetCapsuleVars(List(listCaps.ListIndex),
   startVar, numVars, disBuffer, bufferSize, DIS_NOWAIT,
  sessionHandle)
```

5. Call DisBufferToVBArray to copy the data from the disBuffer structure into the atVars array of structures.

```
ReDim atVars(1 To numVars)
retCode = DisBufferToVBArray(atVars(1),
   Len(atVars(1)), disBuffer, bufferSize,
  DISCAPSULEVARINFO_FORMAT,  numVars)
```

6. Set up a grid control (grdAtVar) to display the selected capsule application's @-variables and to let the user select @-variables to be changed.

```
grdAtVar.Rows = numVars + 1
For i = 1 To numVars
    grdAtVar.Row = i
    grdAtVar.Col =
    grdAtVar.Text = '@ ' & atVars(i).szName
    grdAtVar.Col = 1
    grdAtVar.Text = atVars(i).pszDescription
    grdAtVar.Col = 2
    grdAtVar.Text = atVars(i).pszValue
Next i
Screen.MousePointer =
```

7. Free the space used by disBuffer, because atVars now holds the message data. (If n–1 @-variables are already set and the user sets the nth one, then the buffer you created for DisGetCapsuleVars would now be too small for DisSetCapsuleVars. If you are certain that the user will never set previously unspecified @-variable data, you can leave disBuffer for use by DisSetCapsuleVars.)

   ```
   retCode = DisFreeBuffer(disBuffer)
   ```

8. When the user finishes changing @-variables, copy them from the grid to the atVars array of structures. In copying the name of the @-variable, remove the @. (For example, copy A instead of @A.) Also, concatenate a null byte to the end of the @-variable name.

   ```
   Screen.MousePointer = 11
   grdAtVar.Rows = numVars + 1
   For i = 1 To numVars
       grdAtVar.Row = i
       grdAtVar.Col =
       atVars(i).szName = Right$(grdAtVar.Text,
          Len(grdAtVar.Text) - 2) & Chr$( )
       grdAtVar.Col = 1
       atVars(i).pszDescription = grdAtVar.Text & Chr$( )
       grdAtVar.Col = 2
     atVars(i).pszValue = grdAtVar.Text & Chr$( )
   Next i
   ```

9. Call DisCreateBuffer to allocate and lock a buffer to be used by DisSetCapsuleVars.

   ```
   ReDim vbBuffer(1 To numVars)
   disBuffer = DisCreateBuffer(DISCAPSULEVARINFO_FORMAT,
     numVars,  bufferSize)
   ```

10. Call DisVBArrayToBuffer to copy the data from the atVars structure into the disBuffer structure.

    ```
    retCode = DisVBArrayToBuffer(disBuffer, bufferSize,
       vbBuffer(1), vbBuffer(1), DISCAPSULEVARINFO_FORMAT,
      numVars)
    ```

11. Call DisSetCapsuleVars to send the new values to the specified capsule application on the IDS desktop.

    ```
    retCode = DisSetCapsuleVars(txtCapsule.Text, disBuffer,
       numVars, DIS_NOWAIT, sessionHandle)
    ```

12. Free the space used by disBuffer.

    ```
    retCode = DisFreeBuffer(disBuffer)
    Screen.MousePointer =
    ```

Running Capsule Applications

To run an existing capsule application, use the DisSubmitCapsule function.

The following code samples demonstrate how to run a capsule application. In this example, jobParms is an array of structures of the type DISCAPSULEJOBPARMS, as described in “DisSetCapsuleVars” on page 159.

1. Declare the DISCAPSULEJOBPARMS and DISDATEANDTIME structures and other necessary variables.

```
Dim timeSpec As Long
Dim retCode As Integer
Dim jobNumber As Long
Dim disBuffer As Long
Dim bufferSize As Integer
Dim timeStructure As DISDATEANDTIME
Dim jobParm As DISCAPSULEJOBPARMS
```

2. Set the capsule job parameters in the DISCAPSULEJOBPARMS structure and set the corresponding flags in the flag mask member, flMask. For the cb member, call DisCFormatSize to find the size of the jobParms structure.

```
Screen.MousePointer = 11
jobParm.fbFlags =
jobParm.cb = DisCFormatSize(DISCAPSULEJOBPARMS_FORMAT,
  DISUB_TOTAL_SIZE)
jobParm.bPriority = 31
jobParm.flMask = DIS_CJP_PRIORITY    ' Flag use of bPriority

jobParm.pszDescription = txtJobDescription.Text & Chr$( )
jobParm.flMask = jobParm.flMask Or DIS_CJP_DESCRIPTION

If optAlias.Value = True Then
    jobParm.pszAliazName = txtAlias.Text & Chr$( )
    jobParm.flMask = jobParm.flMask Or DIS_CJP_ALIASNAME
End If
```

3. If you provide a text box for the user to specify starting time or maximum delay and execution times, convert the text string into a DISDATEANDTIME structure of integers, then call DisMkTime to convert that structure into a long integer, as specified in “DisMkTime” on page 143.

```
If optTime.Value = True Then
    jobParm.timeScheduled = DisMkTime(timeStructure)
    jobParm.flMask = jobParm.flMask Or DIS_CJP_SCHEDULED
End If
```

4. If necessary, set the capsule application's @-variables as described in “Setting Capsule Variables” on page 100.
5. If you use a list box to help the user select a folder into which the capsule application is to be copied after execution, call DisQueryObjectInfo with the DIS_QOIL_STANDARD code to retrieve the list of objects in the desktop, then display only those for which the bObjType member of the DISOBJECTINFO structure indicates a container, such as DIS_OBJTYPE_FOLDER, DIS_OBJTYPE_DRAWER, or DIS_OBJTYPE_ENVELOPE.
6. Call DisCreateBuffer to allocate and lock a buffer to be used by DisSubmitCapsule.

```
disBuffer = DisCreateBuffer(DISCAPSULEJOBPARMS_FORMAT, 1,
  bufferSize)
```

7. Call DisVBTypeToBuffer to copy the data from the jobParm structure into the disBuffer structure.

   ```
   retCode = DisVBTypeToBuffer(disBuffer, bufferSize,
   jobParm, Len(jobParm), DISCAPSULEJOBPARMS_FORMAT)
   ```

8. Call DisSubmitCapsule to submit the capsule application to the capsule server.

   ```
   retCode = DisSubmitCapsule(List(listCaps.ListIndex),
      List(listContainers.ListIndex), disBuffer, jobNumber,
     DIS_NOWAIT,  sessionHandle)
   ```

   If you want to use the DisQueryCapsuleInfo, DisSetCapsuleStatus, or DisWaitCapsuleStatus function later, be sure to store the returned jobNumber for that later use.

9. Free the space used by disBuffer.

   ```
   retCode = DisFreeBuffer(disBuffer)
   Screen.MousePointer =
   ```

Checking a Capsule Application's Progress

To check the status of a capsule application in the queue, use the DisQueryCapsuleInfo function.

The following code samples demonstrate how to check a capsule application's status. In this example, capsuleJobInfo is a structure of the type DISCAPSULEJOBINFO_L1, as described in “DisQueryCapsuleInfo” on page 148.

1. Declare pointers to the DISCAPSULEJOBINFO_L1 parameters structure, with other necessary variables, and get the session handle.

   ```
   Dim retCode As Integer
   Dim bufferSize As Integer
   Dim numJobs As Integer
   Dim jobNumber As Long
   Dim capsuleStatus As DISCAPSULEJOBINFO_L1
   ```

2. Wait for the completion code for the current capsule application, using the job number that the DisSubmitCapsule function returned.

   ```
   Screen.MousePointer = 11
   retCode = DisWaitCapsuleStatus(jobNumber, DIS_WCS_OUTPUT,
     -1, completion, DIS_WAIT, sessionHandle)
   ```

3. If completion is 0, the capsule application is not in the queue. Call DisQueryCapsuleInfo to make sure that it is run and that it exists in the execution folder.

```
If completion =   Then
   numJobs = 1
   capsuleStatus = DisCreateBuffer
      (DISCAPSULEJOBINFO_L1_FORMAT, numJobs, bufferSize)
   retCode = DisQueryCapsuleInfo(DIS_QCI_JOB, jobNumber,
      DIS_QCIL_STANDARD, numJobs, capsuleStatus,
      bufferSize, DIS_WAIT, sessionHandle)
  End  If
Screen.MousePointer =
```

Advanced ODAS Programming

This section describes how to create object identifiers, process functions asynchronously, and maximize your application's performance.

Object Identifiers

Unlike the DOS file system, the IDS file system permits more than one object of the same name to exist in a container. When you refer to an object by name, ODAS always selects the object that appears first in the container directory. The IDS directory is not sorted, so you cannot depend on which of the like-named objects will be selected. (The IDS workstation sorts the directory before displaying it.)

ODAS lets applications distinguish between several like-named objects by using a specially formatted, unique object identifier. The DisQueryCapsuleInfo function returns this identifier in the ID field of the DISOBJECTINFO structure. (See “DisQueryObjectInfo” on page 151.)

To use the object identifier, substitute a name of the form “MET@number”, where number is a string created from the hexadecimal representation of the unique object identifier. (See “Objects with Non-Unique Names” on page 11.

You must specify any container navigation required to locate the object. The object identifier name must only appear wherever the actual object name can appear.

Asynchronous Processing

Sometimes it is desirable to initiate an operation and not have to wait for the operation to complete. ODAS allows this through use of the DIS_NOWAIT flag, as described in “DIS_NOWAIT” on page 15.When you specify this flag in a function call, the function returns as soon as the request is sent to the server.

Later, your application can use DisQueryResult to query the result of the request. For example:

```
Dim retCode As Integer
Dim batchRetCode(1 To 3) As Integer

retCode = DisInitBatch( , sessionHandle)
If retCode =   Then
   ' Copy three spreadsheets and total their return codes.
   retCode = DisCopyObject('\Budgets\FY93',
      '\PC Budgets\FY93.WKS', DIS_OBJTYPE_WKS, DIS_WAIT
     sessionHandle)
   retCode = retCode + DisCopyObject('\Budgets\FY92',
      '\PC Budgets\FY92.WKS', DIS_OBJTYPE_WKS, DIS_WAIT,
     sessionHandle)
   retCode = retCode + DisCopyObject('\Budgets\FY91',
      '\PC Budgets\FY91.WKS', DIS_OBJTYPE_WKS, DIS_WAIT,
     sessionHandle)
   If retCode =   Then                  ' if all three were error-free
      retCode = DisSubmitBatch(batchRetCode(1), 3,
        DIS_NOWAIT,  sessionHandle)
      If retCode =   Then
         ' Do something until DisQueryResult gives a result
         '       or there is nothing more to do
        Do
           If NothingMoreToDo() = 1 Then
              retCode = DisQueryResult( , DIS_WAIT,
                sessionHandle)
             Exit  Do
           End  If
           retCode = GoDoSomethingElse()
           retCode = DisQueryResult( , DIS_NOWAIT,
             sessionHandle)
         Loop While retCode = ERROR_MET_REQUEST_PENDING
         If retCode =   Then
            fSuccess = True
        End  If
      End  If
  End  If
End If
```

You can callDisQueryResult either with or without the DIS_NOWAIT flag. Without DIS_NOWAIT, DisQueryResult waits until the request completes. If you use DisQueryResult on a batch request, the iSubRequest parameter determines whether you want the overall result of the batch request or the result of an individual subrequest. Because the previous example specifies a zero iSubRequest, DisQueryResult examines the overall state of the batch request.

You cannot use the DIS_NOWAIT flag on DisInitClient, DisTermClient, DisConnect, or DisDisconnect.

Maximum Application Performance

The most effective way to ensure the highest performance and best response time in ODAS applications is to reduce the number of requests and responses that must pass between the client and the server.

Two methods that can reduce the request-and-response requirements of an application are:

1. Structure the application logic to require fewer ODAS Access function calls and resources for the desired operation, including eliminating unnecessary function calls, supplying large buffers when calling query-type functions, and using path names to refer to objects in various containers on the desktop.
2. Take advantage of the batch-request capability to bundle requests.

Unnecessary Functions

The easiest function call to eliminate is DisLogoff. The DisDisconnect and DisLogon functions automatically log off the current desktop if DisLogoff has not been called. Because the typical ODAS application disconnects the client from the server when it has completed its desktop tasks, you can simply use the DisDisconnect function to both log off the desktop and end the ODAS session.

Avoid introducing too much robustness into the application at the expense of good application performance. While it is attractive to call DisQueryObjectInfo to ensure existence of an object before retrieving it, you can gain better performance by omitting that call and checking the return code from DisGetObject to determine whether the object was present as expected. On the other hand, if your user will have to type the object name, instead of just selecting it from your menu, you might find the overhead of DisQueryObjectInfo worthwhile by providing an up-to-date menu for the user's selection.

Large Buffers

When you call the query-type functions (such as DisQueryObjectInfo or DisQueryCapsuleInfo), you can greatly improve application response time by using larger buffers and retrieving many entries at once, rather than retrieving only one entry per function call.

For example, this method for listing the contents of the Mail Tray in a list box (listMail) performs poorly because it retrieves only one object's information in each DisQueryObjectInfo call:

```
index =
numMessages = 1
disBuffer = DisCreateBuffer(DISOBJECTINFO_FORMAT,
  numMessages,  bufferSize)

Do
   retCode = DisQueryObjectInfo('\Mail Tray\',
      DIS_QOIL_STANDARD, index, numMessages, disBuffer,
      bufferSize, DIS_QOI_COUNT_CONTENTS, sessionHandle)
   If retCode <>
     Exit  Do
  End  If
   retCode = DisBufferToVBType(vbBuffer, Len(vbBuffer),
      disBuffer, bufferSize, DISOBJECTINFO_FORMAT)
   listMail.AddItem(vbBuffer.cbObject & ' - ' &
     vbBuffer.szName)
Loop While index <>
```

A more efficient application would store many objects' information in a single, large buffer, as shown in the following example.

```
Dim vbBuffer() As DISOBJECTINFO
numMessages = 1
bufferSize =  x4
index =
Do
   retCode = DisQueryObjectInfo('\Mail Tray\',
      DIS_QOIL_STANDARD, index, numMessages, disBuffer,
      bufferSize, DIS_QOI_COUNT_CONTENTS, sessionHandle)
   If retCode <>   & retCode <> ERROR_MET_BUFFER_TOO_SMALL
     Exit  Do
  End  If
   ReDim vbBuffer(1 To numMessages)
   retCode = DisBufferToVBArray(vbBuffer(1),
      Len(vbBuffer(1)), disBuffer, bufferSize,
     DISOBJECTINFO_FORMAT,  numMessages)
   For i =   To numMessages - 1
      listMail.AddItem(vbBuffer(i).cbObject & ' - ' &
        vbBuffer(i).szName
  Next  i

   ' DisQueryObjectInfo increments index, then resets it to
   '       zero when there is nothing left to query
Loop While index <>
```

Path Names

The DisSetCurrentContainer function sets the name of the default container to which unqualified paths are directed. An unqualified path is one that does not begin with the path separator character (which, by default, is the backslash character). For example, if you use DisSetCurrentContainer to set the folder Budgets as the current container, the path FY93 Expenses refers to the object named FY93 Expenses in the Budgets folder;

compare this with the path \FY93 Expenses, which refers to the object FY93 Expenses at the desktop level.

You can also use the path separator character to construct paths that navigate through multiple levels of containers. For example, \New Icons*Blank Icons\Folder refers to the blank Folder icon. If the current container is Budgets, then FY92 Budgets\Q4 Expenses refers to the Q4 Expenses object in the FY92 Budgets folder which, in turn, is in the Budgets folder.

As a general rule, avoid using the DisSetCurrentContainer function, and use qualified and unqualified paths to refer to objects in different containers on the desktop. This increases the complexity of the application somewhat, but avoiding this usage improves the application response time, even over an application that bundles DisSetCurrentContainer into a batch request.

Deletion of Desktop Objects

ODAS deletes objects in a manner similar to the way the developer's workstation deletes objects. Deleted objects are first moved into the Discard folder, from which they can be retrieved until subsequent deletions push the earlier objects out of the Discard folder, purging them from the desktop. This feature lets users recover inadvertently deleted icons. However, it is typically useful only in interactive applications, in which the user can manipulate objects directly.

If your application is non-interactive, or does not let users directly manipulate icons, you can streamline it by using the DIS_DELETE_EXPUNGE flag on function calls that delete desktop objects. This flag truly deletes the objects instead of moving them into the Discard folder; hence, the ODAS server and file server have fewer operations to perform.

Multiple-Transfer Functions

Whenever possible, your application should attempt to use the multiple-transfer functions (DisGetMultObjects and DisPutMultObjects) to send and receive a set of objects rather than calling DisGetObject or DisPutObject for each individual object.

ODAS can convert and transfer several objects more quickly with the multiple-transfer functions because:

- Fewer requests and responses have to pass between the client and server
- The overhead of searching for objects and setting up for conversions is shared among all the object transferals
- If enough Conversion Services engines are available, they can convert several objects simultaneously, thereby reducing the overall conversion time

Batch Requests

The ODAS batch facility lets you bundle several client requests and send them to the server as a unit. This reduces both transmission time between the client and server and the transaction time on the server.

DisInitBatch and

To use the batch facility, enclose the functions between the ODAS functions DisSubmitBatch. The DisInitBatch function initiates batch processing. All subsequent ODAS function calls executed by the application are not sent to the server but are queued locally as “subrequests.”

When the application is ready to send the batch request to the server, it executes the DisSubmitBatch function. If the application supplies a suitable array when it calls DisSubmitBatch, ODAS will return the individual results of the batch subrequests in that array.

The following example transfers a mail message to the desktop, adds several enclosures, and moves the resulting envelope to the Mail Tray to deliver the message to the Post Office. The example assumes that the text of the message is in the file \NEWMAIL\ENV, the enclosures are in the directory \NEWMAIL\ENCL, and the conversion table is appropriately configured to perform the desired conversions on the enclosures.

```
Dim retCode As Integer
Dim aerrorResults(1 To 3) As Integer

retCode = DisInitBatch( , sessionHandle)
If retCode =
    retCode = DisPutObject('\newmail\env', DIS_FILETYPE_TEXT,
      '\NewEnv',  DIS_OBJTYPE_ENVELOPE,
       DIS_OVERWRITE Or DIS_DELETE_EXPUNGE, sessionHandle)
    If retCode =
        retCode = DisPutMultObjects('\newmail\encl\ ',
           '\NewEnv', DIS_WAIT, sessionHandle)
   End  If
    If retCode =
        retCode = DisMoveObject('\NewEnv', '\Mail Tray\',
          DIS_WAIT,  sessionHandle)
   End  If
    If retCode =
        retCode = DisSubmitBatch(aerrorResults(1), 3, DIS_WAIT,    sessionHandle)
   Else
        retCode = DisTermBatch( , sessionHandle)
   End  If
End If
```

Notice the call to DisTermBatch if one of the subrequest calls returns an error code. This function terminates batch processing and discards the remaining queued subrequests without sending them to the server.

The return code from DisSubmitBatch provides an overall indication of the success or failure of the subrequests. You can examine the elements of the aerrorResults array to obtain the return codes from the individual subrequests.

Whenever possible, an application should combine requests using the batch facility. Another common batch request is to combine DisGetObject with DisQueryObjectInfo to retrieve the text of an envelope and at the same time determine the names, types, and sizes of the envelope's enclosures. Still another is to combine DisSetConversionTable with DisGetMultObjects to configure the appropriate conversions and to convert and retrieve the desired objects.

The DisInitClient, DisTermClient, DisConnect, and DisDisconnect functions cannot be executed as batch subrequests.

Chapter 6. API Functions for Visual Basic Programming

Each of the application programming interface (API) functions described in this chapter performs a specific action. In most cases, this action is analogous to an action frequently performed by Intelligent Decision Server (IDS) workstation users. For example, the DisLogoff function is equivalent to the workstation user selecting the desktop's Log Off button.

The functions require libraries specifically designed for the Windows operating system. “Compiling C Applications” on page 8 describes how to obtain the appropriate library.

In Visual Basic, each statement must be all on one line. In this manual, however, the available page space requires some statements to be spread over two or more lines. For short examples, you can assume that the line continues until the statement semicolon (;). For longer examples, , a hollow arrow (

) is used to indicate the continuation of a statement on the next line.

Functional Lists of API Functions

Table 25 through Table 31 on page 114 list the API functions, grouped by their uses. Later in this chapter, the detailed API descriptions are sorted in alphabetical order for easy reference.

Table 25. API functions for session and desktop controlFm Variable:Table Sheet

Function	Action
DisInitClient ()	Initializes the ODAS application.
DisTermClient ()	Releases resources allocated for the ODAS application.
DisConnect ()	Establishes a connection to the ODAS server.
DisDisconnect ()	Terminates the application's connection with the ODAS server.
DisLogoff ()	Logs off a IDS desktop.
DisLogon ()	Logs on a IDS desktop.
DisQueryObjectInfo ()	Checks the type, size, creation time, and other information of one or more desktop objects.
DisSetPassword ()	Changes the password of the connected desktop.
DisSetSessionInfo ()	Changes the separator character to be used in path names on the desktop.

© Copyright IBM Corp. 1996

Table 26 (Page 1 of 2). API functions for transfer of electronic mail andfilesFm Variable:Table Sheet

Function	Action
DisGetSysMessage ()	Retrieves a message from the system administrator.
DisGetObject ()	Fetches an object from the IDS desktop to a personal computer.
DisGetMultObjects ()	Fetches objects from the IDS desktop to a personal computer.
DisPutObject ()	Sends one personal computer file to the IDS desktop.
DisPutMultObjects ()	Sends one or more personal computer files to the IDS desktop.
DisQuerySessionInfo ()	Checks either the current container path or the existence of pending system messages or mail.
DisSetConversionTable ()	Establishes the type-conversion table to be used when copying a personal computer file or desktop object.

Table 27. API functions for manipulating desktop objectsFm Variable:Table Sheet

Function	Action
DisCopyObject ()	Duplicates a desktop object, placing the new copy in a specified container.
DisDeleteObject ()	Deletes the specified object from the desktop.
DisMoveObject ()	Relocates a desktop object, renames the object, or performs both actions.
DisSetCurrentContainer ()	Specifies the current (default) desktop container on which a subsequent function is to be performed.

Table 28 (Page 1 of 2). API functions for using Capsule applicationsFm Variable:Table Sheet

Function	Action
DisGetCapsuleVars ()	Obtains a copy of the current values of the Capsule application's @-variables.
DisSetCapsuleVars ()	Assigns values to the Capsule application's @-variables.
DisSubmitCapsule ()	Submits the specified Capsule application to the Capsule Services for execution.
DisSubmitCapsuleAndSetVars ()	Submits a Capsule job and sets Capsule @-variables.
DisQueryCapsuleInfo ()	Checks the current status of jobs in the Capsule Services queue.
DisSetCapsuleStatus ()	Specifies whether to cancel, purge, hold, release, or modify the specified Capsule application in the Capsule Services queue, or to set various job parameters.

Table 28 (Page 2 of 2). API functions for using Capsule applicationsFm Variable:Table Sheet

Function	Action
DisWaitCapsuleStatus ()	Pauses the personal-computer application until the specified Capsule application achieves the specified status.

Table 29. API functions for automated file distribution (AFD)

Function	Action
DisAFD ()	Performs a complete AFD session for AFD lists that are set for automatic execution.
DisAFDList ()	Performs a complete AFD session for all matching AFD lists, whether for manual or automatic execution.

Table 30. Other API functionsFm Variable:Table Sheet

Function	Action
DisCancelRequest ()	Cancels the operation that the server is currently processing.
DisGetErrorInfo ()	Returns detailed results information for a specified function call or batch submission.
DisGetPerfInfo ()	Retrieves response-time monitoring (RTM) data for the current session.
DisInitBatch ()	Begins a queue of functions for batch execution.
DisQueryResult ()	Checks the result of an asynchronous operation on the IDS network; determines whether a prior request has been executed and, if so, returns the completion code.
DisSubmitBatch ()	Submits a queue of functions for batch execution.
DisTermBatch ()	Terminates a batch submission, discarding any calls that have not yet been submitted to the ODAS server.

Table 31 (Page 1 of 2). API functions specific to Visual Basic useFm Variable:Table Sheet

Function	Action
DisBufferToVBArray ()	Copies data from an array of variables created by DisCreateBuffer and filled by another ODAS API function, into an array of variables of some IDS-defined type.
DisBufferToVBType ()	Copies data from a variable created by DisCreateBuffer and filled by another ODAS API function, into a variable of some IDS-defined type.
DisVBArrayToBuffer ()	Copies data from an array of variables of some IDS-defined type into an array created by DisCreateBuffer and suitable for use by another ODAS API function.
DisVBTypeToBuffer ()	Copies data from a variable of some IDS-defined type into an variable created by DisCreateBuffer and suitable for use by another ODAS API function.

Table 31 (Page 2 of 2). API functions specific to Visual Basic useFm Variable:Table Sheet

Function	Action
DisCreateBuffer ()	Allocates and locks a buffer to be used by other ODAS API functions.
DisFreeBuffer ()	Unlocks and frees a buffer created by the DisCreateBuffer function.
DisCFormatSize ()	Obtains the size of a C structure, described by a format string.
DisVBFormatSize ()	Obtains the size of a IDS-defined type, described by a format string.
DisMkTime ()	Converts a date and time from a structure of integers to a long integer suitable for passing in a structure to the DisSubmitCapsule function.

Special Functions Specific to Visual Basic

Most of the functions described in this chapter are very similar to the C-language functions described in Chapter 4, “API Functions for C-Language Programming” on page 37.” These functions work identically to the corresponding C functions, except that they are defined for use with the Visual Basic language.

The Visual Basic API also includes the functions listed in Table 31 on page 114. These functions let you convert data between the C and Visual Basic forms.

DisCreateBuffer and DisFreeBuffer allocate and release space for variables to be used by the functions that correspond to C functions.

DisCFormatSize and DisVBFormatSize return the size of a C structure or Visual Basic defined type, described by a format string. Appendix C, “Visual Basic API Type Definitions” on page 192,” lists the structured types defined in the Visual Basic API to correspond to the C structures.

The other four special functions, DisBufferToVBArray, DisBufferToVBType, DisVBArrayToBuffer, and DisVBTypeToBuffer, copy data between a variable or array of a C type and a variable or array of the Visual Basic type.

Data Transfer

Several of the regular API functions retrieve data from the IDS system. To use such data in Visual Basic:

1. Create a IDS buffer for the regular API function to fill.
2. Call the regular API function.
3. Call one of the special functions to copy the retrieved data from the IDS buffer to another buffer of the appropriate data type.
4. Free the IDS buffer.

For example, Figure 5 on page 116 illustrates how to get data about a desktop object into a form that Visual Basic can manipulate. The code sequence would resemble this example:

```
Dim disBuffer As Long
Dim vbBuffer As DISOBJECTINFO
Dim retCode As Integer
. . .
disBuffer = DisCreateBuffer(DISOBJECTINFO_FORMAT, ...)
retCode = DisQueryObjectInfo(..., disBuffer, ...)
retCode = DisBufferToVBType(vbBuffer, ..., disBuffer, ...,
  ↪ DISOBJECTINFO_FORMAT)
retCode = DisFreeBuffer(disBuffer)
<Visual Basic code to operate on  vbBuffer data >
```

A similar procedure works in the opposite direction. To send data to the ODAS service from Visual Basic:

1. Set up the data in a normal Visual Basic variable or array.
2. Create a IDS buffer, with its structure defined by the format string corresponding to the regular API function that you want to call.
3. Call one of the special functions to copy the data from the Visual Basic variable or array into the newly created IDS buffer.
4. Call the regular API function.
5. Free the IDS buffer.

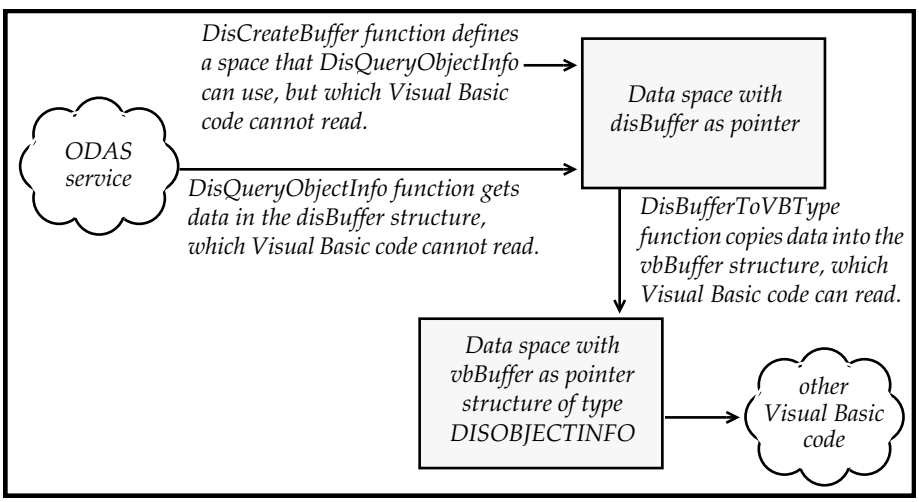


Figure 5. Getting data from the ODAS service for Visual Basic

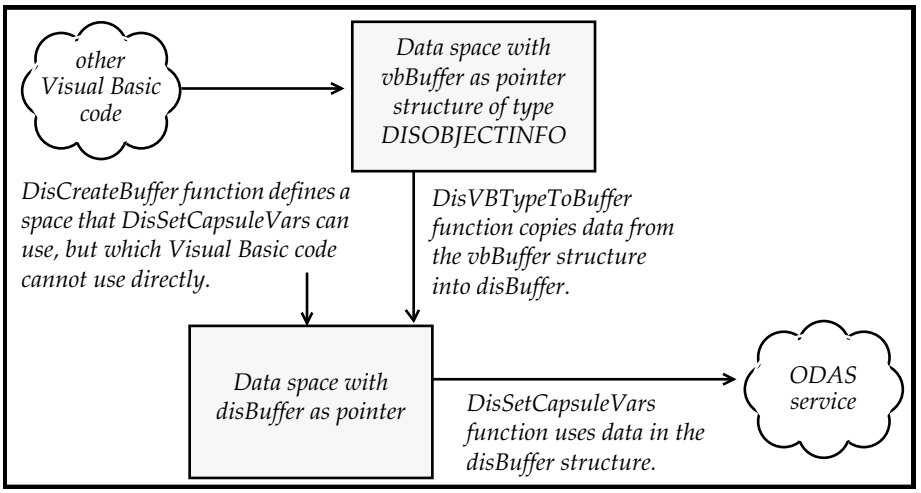


Figure 6. Sending data to the ODAS service from Visual Basic

Figure 6 illustrates how to send Capsule variables from Visual Basic to the ODAS service. The code sequence would resemble this example:

```
Dim disBuffer As Long
Dim vbBuffer As DISOBJECTINFO
Dim retCode As Integer
. . .
<Visual Basic code to operate on  vbBuffer data >
disBuffer = DisCreateBuffer(DISCAPSULEVARINFO_FORMAT, ...)
retCode = DisVBTypeToBuffer(disBuffer, ..., vbBuffer, ...,
  ↪ DISCAPSULEVARINFO_FORMAT)
retCode = DisSetCapsuleVars(..., disBuffer, ...)
retCode = DisFreeBuffer(disBuffer)
```

Some of the special functions require a format string (in the FormatString$ variable), to specify the format of the source and destination variables and arrays. They also return a special set of error codes, in addition to the codes returned by the regular ODAS API functions. The sections below list these format strings and error codes.

Format Strings for Defined Types

Table 32 lists the format strings used in the FormatString$ variable of certain functions listed in Table 31 on page 114.

Table 32 (Page 1 of 2). Format strings for FormatString$Fm Variable:Table Sheet

Format string	Is used for structure	Is used for function
DISCAPSULEJOBINFO_L1_FORMAT	DISCAPSULEJOBINFO_L1	DisQueryCapsuleInfo
DISCAPSULEJOBPARMS_FORMAT	DISCAPSULEJOBPARMS	DisSetCapsuleStatus and DisSubmitCapsule

Table 32 (Page 2 of 2). Format strings for FormatString$Fm Variable:Table Sheet

Format string	Is used for structure	Is used for function
DISCAPSULEVARINFO_FORMAT	DISCAPSULEVARINFO	DisGetCapsuleVars, and DisSetCapsuleVars
DISCONVERTENTRY_FORMAT	DISCONVERTENTRY	DisSetConversionTable
DISGETOBJECTINFO_FORMAT	DISGETOBJECTINFO	DisGetMultObjects
DISOBJECTINFO_FORMAT	DISOBJECTINFO	DisQueryObjectInfo
DISSYSMESSAGE_FORMAT	DISSYSMESSAGE	DisGetSysMessage

Error Codes for Visual Basic Functions

Table 33 through Table 35 list the error codes returned by the special functions.

Table 33. Allocation error codes for Visual Basic functions

Error code	Value	Description
DISVB_OUTOFMEMORY	–1	The system is out of memory.
DISVB_TOOMANYBUFFERS	–2	The program has allocated too many buffers.
DISVB_BADELEMENTSIZE	–3	The buffer size is greater than 64 KB.

Table 34. Format error codes for Visual Basic functions

Error code	Value	Description
DISVB_BADFMTTOKENTYPE	–10	The format string contains an unsupported token.
DISVB_BADFORMAT	–11	A token in the format string is missing its size indicator.

Table 35. Transfer error codes for Visual Basic functions

Error code	Value	Description
DISVB_INVALID_BUFFER	–20	The Visual Basic buffer or IDS buffer is null.
DISVB_INVALID_CBUFFERSIZE	–21	The size of the C buffer is not what the format string specifies.
DISVB_INVALID_VBBUFFERSIZE	–22	The size of the Visual Basic buffer is not what the format string specifies.

Functions Reference Section

The remainder of this chapter presents the Visual Basic API functions in alphabetical order, with full details including the calling syntax and examples of usage.

This chapter uses several notational conventions, described in the beginning of this manual, to indicate the correct syntax of each function.

DisAFD

This function performs a complete AFD session, but only for AFD lists that are set for automatic execution.

If your application must execute AFD operations conditionally, or if you want to specify how much data is to be transferred before actually confirming the action, you should use the individual functions DisInitAFD, then DisExecuteAFD, then DisTermAFD. DisAFDcalls those three functions.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisAFD Lib 'metrapiw.dll' (ByVal fFlags As Integer, ByVal hdis As Long) As Integer

where:

fFlags	Must be 0 (zero).
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example performs a complete AFD session.

```
Dim er As Integer
er = DisAFD ( , sessionHandle )
```

DisAFDList

This function performs a complete AFD session for all matching AFD lists, whether for manual or automatic execution.

If your application must execute AFD operations conditionally, or if you want to specify how much data is to be transferred before actually confirming the action, you should use the individual functions DisInitAFDList, then DisExecuteAFD, then DisTermAFD. DisAFDListcalls those three functions.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Prototype

Declare Function DisAFDList Lib “metrapiw.dll” (ByVal pszNameAs
String, ByVal fFlagsAs Integer, ByVal hdisAs Long) As Integer

where:

pszName	Is a pointer to the name pattern to be compared with names of AFD lists assigned to the user or to the user's groups. The name pattern can contain a wildcard asterisk (*).
fFlags	Must be 0 (zero).
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example performs a complete AFD session for all AFD lists, both manual and automatic, that are assigned to the user.

```
Dim er As Integer
er = DisAFDList ( '   ',  , sessionHandle )
```

DisBufferToVBArray

This function copies data from an array of variables created by DisCreateBuffer and filled by another ODAS API function, into an array of variables of some IDS-defined type that Visual Basic can read.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisBufferToVBArray% Lib 'disvb.dll' (Dest As
Any, ByVal DestLen%, ByVal Src&, ByVal SrcLen%, ByVal FormatString$, ByVal
NumEntries%)

where:

Dest	is a variable that Visual Basic can read. Dest must be declared to be one of the defined types (such as DISOBJECTINFO) listed in Appendix C, “Visual Basic API Type Definitions” on page 192.”The first array element is passed as Dest(1).
DestLen%	is the length of each element of the Dest variable, as obtained through the function Len(Dest(1)). DestLen% verifies that FormatString$ and Dest match.
Src&	is a pointer to an array created by DisCreateBuffer and filled by a regular ODAS API function.

SrcLen%	is the length, in bytes, of the Src array, as returned by DisCreateBuffer or obtained through the function DisCFormatSize. SrcLen% verifies that FormatString$ and Src& match.
FormatString$	is a string, as listed in Table 32 on page 117, to specify the formats of Src& and Dest.
NumEntries%	specifies the number of elements in the arrays Src& and Dest.

Remarks

This function returns the error codes listed in Table 33 on page 118 through Table 35 on page 118. Returning DISVB_OK indicates that it successfully copied the data.

This function is identical to DisBufferToVBType, except that it copies data between arrays instead of individual variables.

Example

This example gets data about a desktop object into a form that Visual Basic can manipulate.

The example assumes that the associated form contains a text box named txtContainer, where the user can type the name of the container whose contents DisQueryObjectInfo is to list.

```
Dim maxObjects As Integer
Dim disBuffer As Long
Dim vbBuffer() As DISOBJECTINFO
...
maxObjects = 5
disBuffer = DisCreateBuffer(DISOBJECTINFO_FORMAT,
  ↪ maxObjects, bufferSize)
retCode = DisQueryObjectInfo(txtContainer.Text,
  ↪ DIS_QOIL_STANDARD, index, maxObjects, disBuffer,
  ↪ bufferSize, DIS_WAIT, sessionHandle)
ReDim vbBuffer(1 To maxObjects)
retCode = DisBufferToVBArray(vbBuffer(1), Len(vbBuffer(1)),
  ↪ disBuffer, bufferSize, DISOBJECTINFO_FORMAT, maxObjects)
retCode = DisFreeBuffer(disBuffer)
<Visual Basic code to operate on  vbBuffer data >
```

DisBufferToVBType

This function copies data from a variable created by DisCreateBuffer and filled by another ODAS API function, into a variable of some IDS-defined type that Visual Basic can read.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisBufferToVBType% Lib 'disvb.dll' (Dest As Any,
ByVal DestLen%, ByVal Src&, ByVal SrcLen%, ByVal FormatString$)

where:

Dest	is a variable that Visual Basic can read. Dest must be declared to be one of the defined types (such as DISOBJECTINFO) listed in Appendix C, "Visual Basic API Type Definitions" on page 192."
DestLen%	is the length, in bytes, of the Dest variable, as obtained through the function Len(Dest). DestLen% verifies that FormatString$ and Dest match.
Src&	is a variable created by DisCreateBuffer and filled by a regular ODAS API function.
SrcLen%	is the length, in bytes, of the variable in Src, as returned by DisCreateBuffer or obtained through the function DisCFormatSize. SrcLen% verifies that FormatString$ and Src& match.
FormatString$	is a string, as listed in Table 32 on page 117, to specify the formats of Dest and Src&.

Remarks

This function returns the error codes listed in Table 33 on page 118 through Table 35 on page 118. Returning DISVB_OK indicates that it successfully copied the data.

This function is identical to DisBufferToVBArray, except that it copies data between individual variables instead of arrays.

Example

This example gets data about a desktop object into a form that Visual Basic can manipulate.

```
Dim disBuffer As Long
Dim vbBuffer As DISOBJECTINFO
...
disBuffer = DisCreateBuffer(DISOBJECTINFO_FORMAT, 1,
  ↪bufferSize)
retCode = DisQueryObjectInfo("\Mail Tray\",
  ↪DIS_QOIL_STANDARD, index, maxObjects, disBuffer,
  ↪bufferSize, DIS_WAIT, sessionHandle)
retCode = DisBufferToVBType(vbBuffer, Len(vbBuffer),
  ↪disBuffer, bufferSize, DISOBJECTINFO_FORMAT)
retCode = DisFreeBuffer(disBuffer)
<Visual Basic code to operate on  vbBuffer data >
```

DisCancelRequest

This function cancels the operation that the ODAS server is currently processing.

If the server is already at a certain point in processing the request you want to cancel, the server may not be immediately available for a new request.

If a batch request is being processed, some subrequests may already have been processed. Check the result codes of the individual subrequests to find which, if any, are complete.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

```
Declare Function DisCancelRequest Lib 'metrapiw.dll' (ByVal hdis
As Long) As Integer
```

where:

hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example cancels the request currently being processed by the ODAS server.

```
Dim er As Integer
er = DisCancelRequest ( sessionHandle )
```

DisCFormatSize

This function obtains the size of a C structure, described by a format string.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

```
Declare Function DisCFormatSize% Lib 'disvb.dll' (ByVal
FormatString$, ByVal Flag%)
```

where:

FormatString$	is a string, as listed in Table 32 on page 117,to specify the fields layout of a C structure.
Flag%	specifies which size information to return. Table 36 on page 124 lists the valid values.

Table 36. Information selection codes for DisCFormatSize ()

Specifier	Value	Description
DISVB_TOTAL_SIZE	0	Gets the total size of the structure, including variable-length strings.
DISVB_FIXED_LEN_SIZE	1	Gets the fixed-length size, which does not include variable-length strings.

Remarks
If this function returns a negative value, it is one of the error codes listed in Table 33 on page 118 through Table 35 on page 118.If it returns a positive value, that value is the requested size, in bytes.

Example
This example sets the size of a structure named lanComm.

```
Dim lanComm As DISLANCOMM
...
lanComm.cb = DisCFormatSize(DISLANCOMM_FORMAT,
  DISUB_FIXED_LEN_SIZE)
```

DisConnect

This function establishes a connection to the ODAS server. Call this function before calling any other ODAS API function (other than DisInitClient, which must be called first).

Desktop Equivalent
The equivalent action, if the user were at a IDS workstation, would be to start the workstation from the OS/2 desktop.

Function Declaration

Declare Function DisConnect% Lib 'metrapiw.dll' (ByVal uCommType%, pCommParms As Any, ByVal fFlags%, hdis&)

where:

uCommType%	must be 2or the constant DIS_COMM_LAN.
pCommParms	is a pointer to the communication-parameters structure defined for the communications type specified in uCommType. The client application must supply the name of the server, as defined by the network software (such as the TCP/IP domain-name service, or DNS). The pCommParms structure is:

```
Type DISLANCOMM
    cb As Integer          ' Size of the structure, in bytes
    usTimeout As Integer   ' Connection timeout; how many seconds to wait
                                                          for a completed connection
    pszServerName As String          ' Server's name
End Type
Global Const DISLANCOMM_FORMAT$ = 'IIP8 '
```

fFlags%	Must be 0 (zero).
hdis&	Is the desktop-session

handle, to be used when calling other functions.

Remarks

The ODAS administrator can configure the server to automatically update the ODAS software whenever a client computer with outdated software is connected. This updating can cause DisConnect to appear to take an abnormally long time to complete.

Example

This example specifies a connection with the communications parameters specified in the structure lanComm, then waits for completion.

```
Dim lanComm As DISLANCOMM
Dim retCode As Integer
Dim disBuffer As Long
...
disBuffer = DisCreateBuffer(DISLANCOMM_FORMAT, 1,
  ↪ bufferSize)
lanComm.cb = DisCFormatSize(DISLANCOMM_FORMAT,
  ↪ DISUB_FIXED_LEN_SIZE)
retCode = DisVBTypeToBuffer(disBuffer, bufferSize, lanComm,
  ↪ Len(lanComm), DISLANCOMM_FORMAT)
retCode = DisConnect(DIS_COMM_DEFAULT, ByVal disBuffer,
  ↪ DIS_WAIT, sessionHandle)
```

DisCopyObject

This function duplicates a desktop object. The new copy can be in the same location as the original or it can be in a specified container. (See “Containers” on page 10.) It can have the same name or a different name. If convertible, it can be in the same format or a different format.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to select an object and use the [Copy] function key to duplicate it.

Function Declaration

Declare Function DisCopyObject% Lib 'metrapiw.dll' (ByVal pszObjectName$, ByVal pszNewObjectName$, ByVal uNewObjectType%, ByVal fFlags%, ByVal hdis&)

where:

pszObjectName$	specifies the original name of the desktop object to be copied.
pszNewObjectName$	specifies the name of the new copy.
uNewObjectType%	specifies a numerical code representing the type of desktop object that the new copy is to be. Choose the code from Table 2 on page 13.To copy without conversion, use DIS_OBJTYPE_DEFAULT.
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. specify the DIS_DELETE_SOURCE flag to move the original object into the desktop's Discard folder after the operation. specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove the original object from the desktop.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example copies the Next Report spreadsheet, then returns without waiting for completion:

The example assumes that the associated form contains three text boxes named txtObjectName, txtNewObjectName, and txtNewObjectType, where the user can type the old and new names of the object to be copied, and the type of the new object.

If also assumes the existence of a function named GetDISFileType, which converts the string from txtNewObjectType into one of the object-type constants (such as DIS_OBJTYPE_DOCUMENT) listed in Table 2 on page 13.

```
Dim retCode As Integer
...
retCode = DisCopyObject(txtObjectName.Text,
  txtNewObjectName.Text,
  GetDISFileType(txtNewObjectType.Text),
  DIS_NOWAIT, sessionHandle)
```

DisCreateBuffer

This function allocates and locks a buffer to be used by other ODAS API functions.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisCreateBuffer& Lib 'disvb.dll' (ByVal FormatString$, ByVal NumEntries%, BufferSize%)

where:

FormatString$	is a string, as listed in Table 32 on page 117,to specify the formats of the buffer.
NumEntries%	specifies the number of elements in the buffer.
BufferSize%	returns the size of the buffer, in bytes.

Remarks

This function returns a pointer to a locked buffer. If it fails, it returns one of the error codes in Table 33 on page 118 through Table 35 on page 118.

Example

This example creates and frees a buffer named disBuffer.

```
Dim disBuffer As Long
Dim bufferSize As Integer
...
disBuffer = DisCreateBuffer(DISOBJECTINFO_FORMAT, 1,
  ↪bufferSize)
retCode = DisFreeBuffer(disBuffer)
```

DisDeleteObject

This function deletes the specified object from the desktop.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to select an object and use the [Delete] function key to remove it from the desktop.

Function Declaration

Declare Function DisDeleteObject% Lib 'metrapiw.dll' (ByVal pszObjectName$, ByVal fFlags%, ByVal hdis&)

where:

pszObjectName$	specifies the name of the object to be deleted.
fFlags%	specifies optional flags, as described in "Common Bit Flags" on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove the object from the desktop.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example completely deletes a specified object from the desktop, then returns without waiting for completion.

The example assumes that the associated form contains a text box named txtObjectName, where the user can type the name of the object to be deleted.

```
Dim retCode As Integer
...
retCode = DisDeleteObject(txtObjectName.Text,
  ➪ DIS_DELETE_EXPUNGE Or DIS_NOWAIT, sessionHandle)
```

For instance, a user wanting to delete the Next Report spreadsheet from the desktop might type Next Report in the txtObjectName space.

DisDisconnect

This function terminates the desktop session instance and the application's connection with the ODAS server.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to select Exit Intelligent Decision Server.

Function Declaration

```
Declare Function DisDisconnect% Lib 'metrapiw.dll' (ByVal
fFlags%, ByVal hdis&)
```

where:

fFlags%	Must be 0 (zero).
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example
This example ends the current desktop session:

```
Dim retCode As Integer
. retCode = DisDisconnect( , sessionHandle)
```

DisFreeBuffer
This function unlocks and frees a buffer created by the DisCreateBuffer function.

Desktop Equivalent
There is no equivalent action for a user at a IDS workstation.

Function Declaration
```
Declare Function DisFreeBuffer% Lib 'disvb.dll' (ByVal BufferPointer&)
```

where:

BufferPointer&	is a pointer to the buffer, as returned by the DisCreateBuffer function.

Remarks
This function returns the error codes listed in Table 33 on page 118 through Table 35 on page 118. Returning DISVB_OK indicates that it successfully freed the buffer.

Example
This example creates and frees a buffer named disBuffer.

```
Dim disBuffer As Long
Dim bufferSize As Integer
Dim retCode As Integer
...
disBuffer = DisCreateBuffer(DISOBJECTINFO_FORMAT, 1,
  ↪ bufferSize)
retCode = DisFreeBuffer(disBuffer)
```

DisGetCapsuleVars
This function obtains a copy of the current values of the Capsule application's @-variables. The values are returned in a DISCAPSULEVARINFO structure.

Desktop Equivalent
The equivalent action, if the user were at a IDS workstation, would be to select the Show Controls button in a Capsule application to open its User Input Controlwindow and look at the currently set @-variables.

Function Declaration

Declare Function DisGetCapsuleVars% Lib 'metrapiw.dll' (ByVal pszCapsuleName$, ByVal pszStartVar$, pcVars%, ByVal pBuf&, pcbBuf%, ByVal fFlags%, ByVal hdis&)

where:

pszCapsuleName$	specifies the name of the Capsule application whose variables are to be obtained.
pszStartVar$	specifies the first of the Capsule variables that are to be obtained. Do not include the @ character; for example, specify B for the @B variable.
pcVars%	specifies the number of variables that are to be obtained. Upon completion, the function returns the number of variables that are actually obtained.
pBuf&	is a pointer to an array of one or more structures that receive the Capsule application's @-variables. You get this pointer from DisCreateBuffer. The DISCAPSULEVARINFO structure is:

```
Global Const DIS_CAPSULEVAR_NAME_LEN = 4
Global Const DIS_CAPSULEVAR_DESC_LEN = 5  'Maximum description length
Global Const DIS_CAPSULEVAR_VALUE_LEN = 255 'Maximum value length
Global Const DIS_CAPSULEVAR_TYPE_DEFAULT =   ' Preserve existing    type
                                    and set new type
Global Const DIS_CAPSULEVAR_TYPE_NUM = 1 ' Variable type is a number
Global Const DIS_CAPSULEVAR_TYPE_DATE = 2 ' Variable type is a date
Global Const DIS_CAPSULEVAR_TYPE_TEXT = 3 ' Variable type is a string
Type DISCAPSULEVARINFO
    szName As String   DIS_CAPSULEVAR_NAME_LEN                    ' Variable name
    ulType As Long                                  Variable type
    pszDescription As String                  Variable's description
    pszValue As String                             Variable's value
End Type
Global Const DISCAPSULEVARINFO_FORMAT$ = 'P4LP5 P255'
```

pcbBuf%	is the size of the pBuf buffer, in bytes, as obtained from DisCreateBuffer. Upon completion, the function returns the actual length of the data it obtained.
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Remarks

While the at-sign (@) is an essential part of the variable name in the normal workstation environment, you cannot use it in the ODAS programming; specify “A” for @A, and so forth.

Example

This example retrieves the current @-variables of a specified Capsule application, then returns without waiting for completion.

The example assumes that the associated form contains three text boxes named txtCapsule, txtStartVar, and txtNumVars, where the user can type the name of the Capsule application, the first @-variable to be retrieved, and the number of @-variables to be retrieved.

```
Dim retCode As Integer
Dim startVar As String
Dim numVars As Integer
Dim bufferSize As Integer
Dim disBuffer As Long
Dim vbBuffer() As DISCAPSULEVARINFO
...
' Remove the at-sign (@):
startVar = Right(txtStartVar.Text, Len(txtStartVar.Text) -
  ↪ InStr(1, txtStartVar.Text, " ")) & Chr$(0)
numVars = Val(txtNumVars.Text)       ' Convert text to number
disBuffer = DisCreateBuffer(DISCAPSULEVARINFO_FORMAT,
  ↪ numVars, bufferSize)
retCode = DisGetCapsuleVars(txtCapsule.Text, startVar,
  ↪ numVars, disBuffer, bufferSize, DIS_NOWAIT,
  ↪ sessionHandle)               ' retrieve the @-variables
ReDim vbBuffer(1 To numVars)
retCode = DisBufferToVBArray(vbBuffer(1), Len(vbBuffer(1)),
  ↪ disBuffer, bufferSize, DISCAPSULEVARINFO_FORMAT,
  ↪ numVars)
<Visual Basic code to operate on  vbBuffer data >
```

For instance, a user, wanting to retrieve @-variables @B, @C, and @D from the Budget Capsule application, might type Budget Capsule in the txtCapsule space, @B in the txtStartVar space, and Capsule in the txtNumVars space.

DisGetErrorInfo

This function returns detailed results information for a specified function call or batch submission.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

```
Declare Function DisGetErrorInfo% Lib 'metrapiw.dll' (ByVal
iRequest%, perri As DISERRORINFO, ByVal hdis&)
```

where:

iRequest%	Specifies the index of the function call for which results are desired. Use 0(zero) to get information about a function call that was not in a batch, or to get summary information about the most recent batch submission. Use a positive integer to get information about a specific function call within the most recent batch submission
perri	is a pointer to a structure that receives the requested information. The DISERRORINFO structure is:

```
' Size of structure (must be set on entry)
    err As Integer                  ' Primary error code
    errSystem As Integer            ' System error code
    uErrLoc As Integer              ' Component in which error occurred
    uErrType As Integer             ' Type of error
    errBaseSys As Integer           ' Untranslated, low-level error code
End Type
```

hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

If the most recent function call was a batch submission, this example returns summary information about the batch. Otherwise, it returns information about the non-batch call.

```
Dim retCode As Integer
Dim disErrorInfo As DISERRORINFO
.retCode = DisGetErrorInfo( %, disErrorInfo, sessionHandle)
```

DisGetMultObjects

This function fetches one or more objects from the IDS desktop to a personal computer file, automatically converting their formats as needed. The names and types of the objects and the resultant files are returned in a DISGETOBJECTINFO structure.

The system tries to determine the type of each new file by matching the type of the original desktop object against the current entries in the conversion table. If no match is found, the system uses the default type specified in the table. To modify the conversion table for the current session, use the DisSetConversionTable function.

If you use the DIS_NOWAIT flag with DisGetMultObjects, no files are transferred until you also call the DisQueryResult function.

Desktop Equivalent

The most equivalent action, if the user were at a IDS workstation, would be to use the PC Directory tool to move files from the desktop to the workstation's OS/2 file system.

Transferring files between the IDS desktop and a different personal computer is not possible without using ODAS.

Function Declaration

Declare Function DisGetMultObjects% Lib 'metrapiw.dll' (ByVal pszObjectSpec$, ByVal pszFileSpec$, pcObjects%, ByVal pBuf&, pcbBuf%, ByVal fFlags%, ByVal hdis&)

where:

pszObjectSpec$	is a string specifying the desired desktop objects. (See "Identifying Desktop Objects" on page 10.)
pszFileSpec$	specifies the base file name and path for all copies on the destination personal computer. The system appends a three-digit number to pszFileSpec$ for each copy made. For example, specifying "C:\HQ\FOO." causes the copies to be named C:\HQ\FOO.001, C:\HQ\FOO.002, and so forth. However, the period (.) is not provided automatically. In the example, specifying "C:\HQ\FOO" (without the period) causes the copies to be named C:\HQ\FOO001, C:\HQ\FOO002, and so forth. Some flags affect this appendage. See fFlags, below.
pcObjects%	specifies the maximum number of desktop objects to be retrieved. Upon completion, the function returns the actual number of desktop objects that were retrieved.
pBuf&	is a pointer to an optional buffer that receives information about the source object and destination file. If you do not specify the DIS_GMO_OBJECT_INFO flag in the fFlags parameter, the function returns information in the DISGETOBJECTINFO structure:

```
Global Const DIS_OBJECT_NAME_LEN = 8
                              /  Maximum length of a desktop object's name  /
Type DISGETOBJECTINFO {
     cb As Integer                  ' Size of this entry, in bytes
     bObjType As Integer            ' Numerical code indicating the original type of
                                          desktop object, as shown in
                                          Table 2 on page 13
     bFileType As Integer           ' Numerical code indicating the destination file
                                          type, as shown in Table 1 on page 12
     cbFile As Long                 ' Size of destination file, in bytes
     szName As String               ' Original desktop-object name (maximum length
                                          is  DIS_OBJECT_NAME_LEN)
End Type
Global Const DISGETOBJECTINFO_FORMAT$ = 'ICCLZ8 '
```

If you do specify the DIS_GMO_OBJECT_INFO flag in the fFlags parameter, the function returns information in the DISOBJECTINFO structure:

```
Global Const DIS_OBJECT_NAME_LEN    8
                                        /  Maximum length of a desktop object's name  /
Type DISOBJECTINFO {
      cb As Integer                      ' Size of this entry, in bytes
      bObjType As Integer                ' Numerical code indicating the type of desktop
                                             object, as shown in Table 2 on page 13
      BYTE fbFlags                       /  Reserved for future use; specify   (zero)  /
      id As Long                         /  Unique identifier for each desktop object, to
                                              distinguish between objects with identical
                                               names   /
      timeCreation As Long               /  Time and date when the object was created /
      timeLastRead As Long               /  Time and date when it was last read  /
      timeLastWrite As Long              /  Time and date when it was last written  /
      cbObject As Long                   /  Size of the object, in bytes  /
      szName As String                    ' Original desktop-object name (maximum length
                                                                    is  DIS_OBJECT_NAME_LEN)
End Type
Global Const DISOBJECTINFO_FORMAT$ = 'ICCLLLLLZ8 '
```

If you do not want DisGetMultObjects to return this information, specify a null pointer as ByVal 0&.

pcbBuf% is the size of the pBuf& buffer, in bytes, as obtained from DisCreateBuffer. Upon completion, the function returns the actual length of the buffer that it retrieved.

fFlags% specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

Specify the DIS_DELETE_SOURCE flag to move the objects into the desktop's Discard folder after the operation.

Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove the objects from the desktop after the operation.

Specify the DIS_OVERWRITE flag to overwrite any existing files that have the same name as retrieved files. If the client computer has an existing file with the same name, and you do not specify DIS_OVERWRITE, the function returns an error.

Specify the DIS_GMO_ID_NAMEflag to alter the text appended to the name in the pszFileSpec parameter. With DIS_GMO_ID_NAME, the unique file-identifier number of the source object is appended instead of the three-digit number that occurs with no flag.

Specify the DIS_GMO_SOURCE_NAMEflag to alter the text appended to the name in the pszFileSpec parameter. With DIS_GMO_SOURCE_NAME, the original IDS name of the source object is appended instead of the three-digit number that occurs with no flag.

Specify the DIS_GMO_OBJECT_INFOflag to return pBuf information in the DISOBJECTINFO structure instead of the default DISGETOBJECTINFO structure.

hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Examples

This example copies all objects from the desktop Mail Tray into a DOS directory DISMAIL as files named ITEM001, ITEM002, and so forth, and then completely deletes the original objects from the desktop. The function returns data in the DISGETOBJECTINFO structure, and waits for completion.

The example assumes that the associated form contains three text boxes named txtSource, txtMaxObjects, and txtDestination, where the user can specify which and how many objects to copy, and what naming scheme to use for the new copies.

```
Dim numObjects As Integer
Dim retCode As Integer
Dim disBuffer As Long
...
numObjects = Val(txtMaxObjects.Text) ' Convert text to numbe
disBuffer = DisCreateBuffer(DISGETOBJECTINFO_FORMAT,
  ↪ numObjects, bufferSize)
retCode = DisGetMultObjects(txtSource.Text,
  ↪ txtDestination.Text, numObjects, disBuffer, bufferSize,
  ↪ DIS_WAIT Or DIS_DELETE_SOURCE, sessionHandle)
```

For instance, a user wanting to retrieve 10 mail messages might type Mail Tray* in the txtSource space, DISMAIL\ITEM in the txtDestination space, and 10 in the txtMaxObjects space.

The next example copies the same objects but does not return the DISGETOBJECTINFO structure:

```
Dim bufferSize As Integer
...
bufferSize = 0
retCode = DisGetMultObjects(txtSource.Text,
  ↪ txtDestination.Text, numObjects, 0&, bufferSize,
  ↪ DIS_WAIT Or DIS_DELETE_SOURCE, sessionHandle)
```

DisGetObject

This function fetches an object from the IDS desktop to a personal computer file, automatically converting its format as needed. You can specify the destination file format to override the conversion table that is currently defined for the session. To modify the conversion table for the current session, use the DisSetConversionTable function.

If you use the DIS_NOWAIT flag with DisGetObject, the file is not transferred until you also call the DisQueryResult function.

Desktop Equivalent

The most equivalent action, if the user were at a IDS workstation, would be to use the PC Directory tool to move a file from the desktop to the OS/2 file system. Transferring a file between the IDS desktop and a different personal computer is not possible without using ODAS.

Function Declaration

Declare Function DisGetObject% Lib 'metrapiw.dll' (ByVal pszObjectName$, ByVal pszFileName$, ByVal uFileType%, pcbFile&, ByVal cchLineWidth%, ByVal fFlags%, ByVal hdis&)

where:

pszObjectName$	specifies the name of the desktop object to be retrieved. A desktop can have more than one object with the same name. To indicate a specific object, you can specify “MET@number”, where number is a string created from the hexadecimal representation of the unique object identifier returned by the DisQueryObjectInfo function. (See “Objects with Non-Unique Names” on page 11.)
pszFileName$	specifies the file name on the destination personal computer.
uFileType%	specifies a numerical code representing the type of file conversion. Choose the code, from Table 1 on page 12, corresponding to the destination file type. You can specify DIS_FILETYPE_DEFAULT to let the conversion table decide the object type, based on the value of uFileType or on the suffix of the name in pszFileName.
pcbFile&	specifies the maximum number of bytes to fetch. If you specify 0 (zero), the function fetches the entire object, regardless of size. Upon completion, the function returns the actual number of bytes that were retrieved. If you want DisGetObject to ignore this parameter, specify a null pointer as 0& for pcbFile.
cchLineWidth%	specifies the maximum line width for a destination text file. If the destination file is not a text file, or you want to specify no maximum, specify 0 (zero).
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_DELETE_SOURCE flag to move the object into the desktop's Discard folder after the operation. Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove the object from the desktop after the operation.

	Specify the DIS_OVERWRITE flag to overwrite any existing file that has the same name as the retrieved file. If the client computer has an existing file with the same name, and you do not specify DIS_OVERWRITE, the function returns an error.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example copies a spreadsheet from the desktop into a Lotus 1-2-3 spreadsheet (type WKS), then completely deletes the document from the desktop. The function call waits for completion.

The example assumes that the associated form contains two text boxes named txtSourceFile and txtDestinationFile, where the user can type the names of the existing desktop spreadsheet and the destination Lotus 1-2-3 spreadsheet.

```
Dim cbFile As Long
Dim retCode As Integer
...
bytes = 0
retCode = DisGetObject(txtSourceFile.Text,
  ↪ txtDestinationFile.Text, DIS_FILETYPE_WKS, bytes, 0,
  ↪ DIS_WAIT Or DIS_DELETE_SOURCE, sessionHandle)
```

DisGetPerfInfo

This function retrieves response-time monitoring (RTM) data — statistics monitoring the system's response time — for the current session.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisGetPerfInfo Lib 'metrapiw.dll' (ByVal pBuf As Long, pcbBuf As Integer, ByVal fFlags As Integer, ByVal hdis As Long) As Integer

where:

pBuf	Points to a buffer that receives the DISAPIPERF structure. All response-time statistics are given in milliseconds. The DISAPIPERFSERVICE structure is:

```
Global Const DIS_PERF_SERVICE_PROT =
Global Const DIS_PERF_SERVICE_DISK = 1
Global Const DIS_PERF_SERVICE_DIR = 2
Global Const DIS_PERF_SERVICE_CAPS = 3
Global Const DIS_PERF_SERVICE_CONV = 4
Global Const DIS_PERF_SERVICE_MAX = 5

Type DISAPIPERFSERVICE
    msec As Long                    / Service's response time /
    msecAccum As Long               / Service's accumulated response time since
                                      the last time the accumulators were reset /
    msecTotal As Long               / Total accumulated response time /
End Type
Global Const DISAPIPERFSERVICE_FORMAT$ = 'LLL'

Type DISAPIPERF
    cRequests As Integer          / Number of requests processed in this session /
    cServices As Integer          / Number of services for which response-time
                                      statistics are available in the Services field of
                                      the DISAPIPERF structure /
    bReserved1 As Integer         / (Reserved for future use) /
    msecIdle As Long              / Time spent outside any API functions, that is,
                                      in the user application or interactive delay
                                      (waiting for user input) /
    msecActive As Long            / Time the API is active; "response time" for the
                                      function or for a complete batch request /
    msecCallActive As Long        / Individual response time for the most recent
                                      batch subrequest completed /
    msecTotalIdle As Long         / Accumulated idle time for the application /
    msecTotalActive As Long       / Accumulated active time for the application /
    msecAccumInterval As Long     / Time elapsed since the accumulators were last
                                      reset (length of the current accumulation) /
    msecAccumActive As Long       / Accumulated active time since the accumulators
                                      were last reset /
    Services (DIS_PERF_SERVICE_MAX) As DISAPIPERFSERVICE
End Type
Global Const DISAPIPERF_FORMAT$ = 'ICCLLLLLLL'

Global Const DIS_GPI_RESET_ACCUM = &H1
```

pcbBuf	Is the size of the pBuf buffer. Before returning, the function sets the value to the size of the structure actually placed in the pBuf buffer.
fFlags	Specifies optional flags. DIS_GPI_RESET_ACCUM 0 Resets the accumulators after reading them.
hdis	Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

DisGetSysMessage

This function retrieves one or more pending messages from the system administrator. Each message is returned in a DISSYSMESSAGE structure.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to click on the desktop's IMPORTANT MESSAGE button and read the displayed message.

Function Declaration

Declare Function DisGetSysMessage% Lib 'metrapiw.dll'
(pcMessages%, ByVal pBuf&, pcbBuf%, ByVal fFlags%, ByVal hdis&)

where:

pcMessages% is the maximum number of messages to be retrieved. The function sets this to the number of messages actually retrieved.

pBuf& is a pointer to an array of one or more structures that receive the messages. You get this pointer from DisCreateBuffer. The DISSYSMESSAGE structure is:

```
Global Const DIS_SYSMESSAGE_LEN     256     ' Maximum message length
Type DISSYSMESSAGE
     cb As Integer                          ' Size of structure, in bytes
     time As Long                           ' Time message was received
     szMessage As String              ' Message text (maximum length
                                                    is  DIS_SYSMESSAGE_LEN)
End Type
Global Const DISSYSMESSAGE_FORMAT$ = 'ILZ256'
```

pcbBuf% is the size of the pBuf& buffer, in bytes, as obtained from DisCreateBuffer. Upon completion, the function returns the actual length of the data it obtained.

fFlags% specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

hdis& is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example retrieves zero to five system messages, but waits for completion:

```
Dim retCode As Integer
Dim numMessages As Integer
Dim bufferSize As Integer
Dim disBuffer As Long
Dim vbBuffer() As DISSYSMESSAGE
...
numMessages = 5;
disBuffer = DisCreateBuffer(DISSYSMESSAGE_FORMAT,
  ↪ numMessages, buffersize)
retCode = DisGetSysMessage(numMessages, disBuffer,
  ↪ bufferSize, DIS_WAIT, sessionHandle)
...
ReDim vbBuffer(1 To numMessages)
retCode = DisBufferToVBArray(vbBuffer(1), Len(vbBuffer(1)),
  ↪ disBuffer, bufferSize, DISSYSMESSAGE_FORMAT,
  ↪ numMessages)
<Visual Basic code to operate on  vbBuffer data >
```

DisInitBatch

This function starts a queue of submitted functions. All subsequent functions are queued until DisSubmitBatch is reached; at that point, the queued functions are submitted to the ODAS server for execution.

Each function in the batch can return errors due to local conditions (such as a missing local file in a DisPutObject call). In this situation, you can use the DisTermBatch command to terminate the batch instead of submitting it to the ODAS server. (See "DisSubmitCapsule" on page 168.)

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

```
Declare Function DisInitBatch% Lib 'metrapiw.dll' (ByVal fFlags%, ByVal hdis&)
```

where:

fFlags%	must be 0 (zero).
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example enables, creates, and submits a batch request comprising four desktop operations:

```
Dim batchResults() As Integer
Dim retCode As Integer
...
retCode = DisInitBatch(0, sessionHandle)
...
   retCode = DisGetObject("Next Report", "NEXTRPT.WKS",
     ↪ DIS_FILETYPE_WKS, 0, 0,
     ↪ DIS_NOWAIT Or DIS_DELETE_SOURCE, sessionHandle)
...
   retCode = DisGetSysMessage(numMessages, disBuffer,
     ↪ bufferSize, DIS_NOWAIT, sessionHandle)
...
   retCode = DisCopyObject("Next Report",
     ↪ "Copy of Next Report", DIS_OBJTYPE_DEFAULT, 0,
     ↪ sessionHandle)
...
   retCode = DisLogoff(DIS_NOWAIT, sessionHandle)
ReDim batchResults(1 To 4)
retCode = DisSubmitBatch(batchResults, 4, DIS_NOWAIT,
  ↪ sessionHandle)
```

DisInitClient

This function initializes the ODAS application. You must call DisInitClient before calling any other ODAS API functions.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisInitClient% Lib 'metrapiw.dll' (ByVal fFlags%)

where:

fFlags%	Specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_INIT_RTMflag if you plan to call DisGetPerfInfofor performance monitoring.

Example

This example initializes the client computer before any other ODAS API functions are called:

```
Dim retCode As Integer
retCode = DisInitClient( )
```

DisLogoff

This function logs off a IDS desktop. The client application can access several different desktops during a single session, but not simultaneously.

Special flags, described below, start or suppress automated file distribution (AFD). AFD lets the ODAS server update your application software as needed.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to select the desktop's Log Off button.

Function Declaration

```
Declare Function DisLogoff% Lib 'metrapiw.dll' (ByVal fFlags%, ByVal hdis&)
```

where:

fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_LOFF_AFD_NOWflag to start an AFD session before logging off, regardless of the user's AFD mode setting in the user-profile database. Specify the DIS_LOFF_AFD_SUPPRESSflag to avoid starting an AFD session when the user's AFD mode in the user-profile database is set for Automatic at logoff. This flag also suppresses a deferred AFD session that would be started if DisLogon is called with the DIS_LON_AFD_DEFERflag.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example logs off the desktop that is currently logged on, then returns without waiting for completion:

```
Dim retCode As Integer
retCode = DisLogoff(DIS_WAIT, sessionHandle)
```

DisLogon

This function accesses a IDS desktop. The client application can access several different desktops during a single session, but not simultaneously. If the application is already logged on, logging on again terminates the previous logon.

Special flags, described below, start or suppress automated file distribution (AFD). AFD lets the ODAS server update your application software as needed.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to enter a desktop name and password.

Function Declaration

Declare Function DisLogon% Lib 'metrapiw.dll' (ByVal pszDesktopName$, ByVal pszDesktopPassword$, ByVal fFlags%, ByVal hdis&)

where:

pszDesktopName$	specifies the desktop to be accessed.
pszDesktopPassword$	specifies the password for that desktop.
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_LON_AFD_NOWflag to start an AFD session after logging on, regardless of the user's AFD mode setting in the user-profile database. Specify the DIS_LON_AFD_SUPPRESSflag to avoid starting an AFD session when the user's AFD mode in the user-profile database is set for Automatic at logon. Specify the DIS_LON_AFD_DEFERflag to defer starting an AFD session until the user logs off, when the user's AFD mode in the user-profile database is set for Automatic at logon.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example assumes that the associated form contains two text boxes named txtName and txtPassword, where the user can type the name and password of the desktop to be accessed. The example accesses the specified desktop and does not return until the request is completed.

```
Dim retCode As Integer
...
retCode = DisLogon(txtName.Text, txtPassword.Text, DIS_WAIT,
  ↪ sessionHandle)
```

DisMkTime

This function converts a date and time from a DISDATEANDTIME structure of integers (for the year, month, day, hour, minute, and second) to a long integer suitable for passing in a DISCAPSULEJOBPARMS structure to the DisSubmitCapsule function.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisMkTime% Lib 'disvb.dll' (time As DISDATEANDTIME)

where:

time	is a string in the format mm/dd/yy hh:mm:ss, where: mm Is a one- or two-digit month representation: 1 = January; 2 = February; ; 12 = December dd Is the day of the month (1 through 31). yy Is the last two digits of the year. hh Is the hour. mm Is the minute. ss Is the second. The DISDATEANDTIME structure is:

```
Type DISDATEANDTIME      sec As Integer                    ' must be   through 59
    min As Integer                  ' must be   through 59
    hour As Integer                 ' must be   through 23
    day As Integer                  ' must be 1 through 7
    month As Integer                ' must be 1 through 12
    year As Integer                 ' must be   through 99
End Type
```

Example

This example converts a date-and-time structure into a long integer.

```
Dim timeStructure As DISDATEANDTIME
Dim timeSpec As Long
timeSpec = DisMkTime(timeStructure)
```

DisMoveObject

This function relocates a desktop object into a different container, renames the object, or performs both actions.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to select an object and use the [Move] function key to relocate it on the desktop. The equivalent to the optional renaming would be to open the object's Icon Options window, delete the old name, and type a new one.

Function Declaration

Declare Function DisMoveObject% Lib 'metrapiw.dll' (ByVal pszObjectName$, ByVal pszNewObjectName$, ByVal fFlags%, ByVal hdis&)

where:

pszObjectName$	specifies one or both of the original name and location of the object. (See “Containers” on page 10.)
pszNewObjectName$	specifies one or both of the new name and location of the object. If you do not specify a location, the object remains in its current location. If you do not specify a new name, the object keeps its current name. If you specify a new name and no new location, the object is renamed.

fFlags%	specifies optional flags, as described in "Common Bit Flags" on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_OVERWRITE flag to replace a existing desktop object that has the same name as a moved object and to move the existing object into the Discard folder. Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove from the desktop an object that has the same name as a moved object.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example moves a document from one folder to another folder and renames it, but returns without waiting for completion.

The example assumes that the associated form contains two text boxes named txtObjectName and txtNewObjectName, where the user can type the old and new names of the object to be moved.

```
Dim retCode As Integer
...
retCode = DisMoveObject(txtObjectName.Text,
  txtNewObjectName.Text, DIS_NOWAIT, sessionHandle)
```

For instance, suppose a user wants to move the Next Report document from the In Basket folder to the Out Basket folder and rename it Finished Report. With this example, the user could type Next Report in the txtObjectName space and Out Basket\Finished Report in the txtNewObjectName space.

DisPutMultObjects

This function sends one or more personal computer files to the IDS desktop, automatically converting their formats as needed.

The system tries to determine the type of each new desktop object by matching the file extension of the original file against the current entries in the conversion table. If the file extension does not match any table entry, the system uses the default type specified in the table. To modify the conversion table for the current session, use the DisSetConversionTable function.

When using DisPutMultObjects with the DIS_DELETE_SOURCE flag, you should either call it synchronously by avoiding the DIS_NOWAIT flag, or call the DisQueryResult function to wait for completion of the DisPutMultObjects call. This caveat is necessary because DisPutMultObjects with DIS_DELETE_SOURCE requires some cleanup processing on the client computer.

Desktop Equivalent

The most equivalent action, if the user were at a IDS workstation, would be to use the PC Directory tool to move files to the desktop from the client computer's file system.

Transferring files between the IDS desktop and a different personal computer is not possible without using ODAS.

Function Declaration

Declare Function DisPutMultObjects% Lib 'metrapiw.dll' (ByVal pszFileSpec$, ByVal pszContainerName$, ByVal fFlags%, ByVal hdis&)

where:

pszFileSpec$	is a string specifying the personal computer files to be sent. (See “Identifying Desktop Objects” on page 10.)
pszContainerName$	is a string specifying the name of the desktop container where the copies are to be placed. (See “Containers” on page 10.)
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_DELETE_SOURCE flag to delete the files from the personal computer's directory after the operation. Specify the DIS_OVERWRITE flag to replace a existing desktop object that has the same name as a copied object and to move the existing object into the Discard folder. Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove from the desktop an object that has the same name as a copied object.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example sends one or more files from the client computer to the IDS desktop, then deletes them from the client computer. The function call waits for completion.

The example assumes that the associated form contains two text boxes named txtPCFile and txtContainer, where the user can type the names of the original file and of the desktop container where the copy is to go.

```
Dim retCode As Integer
...
retCode = DisPutMultObjects(txtPCFile.Text,
  txtContainer.Text, DIS_DELETE_SOURCE Or DIS_WAIT,
  sessionHandle)
```

For instance, suppose a user wants to send files MAIL.TXT and MAIL.WKS to the desktop's My Mail folder, then deletes them from the personal computer. With this example, the user might type MAIL.* in the txtPCFile space and My Mailin the txtContainer space.

DisPutObject

This function sends one personal computer file to the IDS desktop, automatically converting its format as needed.

When using DisPutObject with the DIS_DELETE_SOURCE flag, you should either call it synchronously by avoiding the DIS_NOWAIT flag, or call the DisQueryResult function to wait for completion of the DisPutObject call. This caveat is necessary because DisPutObject with DIS_DELETE_SOURCE requires some cleanup processing on the client computer.

Desktop Equivalent

The most equivalent action, if the user were at a IDS workstation, would be to use the PC Directory tool to move files to the desktop from the client computer's file system. Transferring files between the IDS desktop and a different personal computer is not possible without using ODAS.

Function Declaration

Declare Function DisPutObject% Lib 'metrapiw.dll' (ByVal pszFileName$, ByVal uFileType$, ByVal cbFile%, ByVal pszObjectName$, ByVal uObjectType%, ByVal fFlags%, ByVal hdis&)

where:

pszFileName$	is a pointer to the file identifier of the personal computer file to be sent.
uFileType$	specifies a numerical code representing the type of file to be sent. Choose the code, from Table 1 on page 12, corresponding to the type of source file. You can specify DIS_FILETYPE_DIS to bypass conversion or DIS_FILETYPE_DEFAULT to select the default file type, based on the suffix of the original file name.
cbFile%	specifies the maximum number of bytes to transfer. Specify 0 (zero) if you do not want to set any limit.
pszObjectName$	is a pointer to the name you want to give the new desktop object.
uObjectType%	specifies a numerical code representing the type of desktop object that the new copy is to be. Choose the code from Table 2 on page 13.You can specify DIS_FILETYPE_DIS to bypass conversion or DIS_OBJTYPE_DEFAULT to let the conversion table decide the object type, based on the value of *uFileType* or on the suffix of the original file name.
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_DELETE_SOURCE flag to delete the file from the personal computer's directory after the operation.

	Specify the DIS_OVERWRITE flag to replace a existing desktop object that has the same name as a copied object and to move the existing object into the Discard folder.
	Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove from the desktop an object that has the same name as a copied object.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example sends a file from the client computer to the IDS desktop, then deletes it from the client computer. The function waits for completion.

The example assumes that the associated form contains four text boxes named txtPCFileName, txtPCFileType, txtObjectName, and txtObjectType, where the user can type the names and types of the existing file and the destination desktop object. The example also assumes the existence of two functions:

GetPCFileType, which converts the string in myPCFileType (such as TEXT) into one of the file-type constants (such as DIS_FILETYPE_TEXT) listed in Table 1 on page 12

GetDISFileType, which converts the string from myDISFileType into one of the object-type constants (such as DIS_OBJTYPE_DOCUMENT) listed in Table 1 on page 12.

```
Dim retCode As Integer
...
pcFileType = GetPCFileType(txtPCFileType.Text)
objFileType = GetDisFileType(txtObjectType.Text)
retCode = DisPutObject(txtPCFileName.Text, pcFileType, 0,
  txtObjectName.Text, objFileType,
  DIS_DELETE_SOURCE Or DIS_WAIT, sessionHandle)
```

For instance, suppose a user wants to send file REPORT.TXT to the desktop's June Reportsfolder, then delete them from the personal computer. With this example, the user might type REPORT.TXT in the txtPCFileName space, TEXT in the txtPCFileType space, June Reports\Claybrookin the txtObjectName space, and DOCUMENT in the txtObjectType space.

DisQueryCapsuleInfo

This function checks the current status of jobs in the Capsule Services queue. Each job's information is returned in a DISCAPSULEJOBINFO_L1 structure.

This function returns information about only those jobs that are waiting to run or are currently running. It does not return information about jobs that have finished running. You can use DisWaitCapsuleStatus to check on the jobs that might have already completed.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to open the Capsule Services icon, select Show Queue, and visually inspect the window.

Function Declaration

Declare Function DisQueryCapsuleInfo% Lib 'metrapiw.dll' (ByVal uMode%, ByVal idJob%, ByVal uInfoLevel%, ByVal pcJobs%, ByVal pBuf&, pcbBuf%, ByVal fFlags%, ByVal hdis&)

where:

uMode% is the information selection code, specifying which information you want to retrieve. Table 37 lists the valid values.

Table 37. Information selection codes for DisQueryCapsuleInfo ()

Selection code	Value	Description
DIS_QCI_JOB	1	Return information about the job that the idJob number identifies.
DIS_QCI_FIRST	2	Return information about the first n jobs in the execution queue, where n is the smaller of pcJobs or the number of jobs that fit in the pBuf& buffer. The first job is defined as the one with the earliest scheduled starting time.
DIS_QCI_NEXT	3	Return information about the next n jobs in the execution queue, where n is as described for DIS_QCI_FIRST.

idJob% is the identifying number of the first job whose information is to be retrieved. The DisSubmitCapsule function provides this number. If uMode% is not DIS_QCI_JOB (1), then idJob% is ignored.

uInfoLevel% specifies the amount of information to be retrieved. Table 38 lists the valid values.

Table 38. Detail selection codes for DisQueryCapsuleInfo ()

Selection code	Value	Description
DIS_QCIL_TEST	0	Check for the job's existence and return no other information.
DIS_QCIL_STANDARD	1	Return information in the DISCAPSULEJOBINFO_L1 structure.

pcJobs% specifies the number of jobs whose information is to be retrieved. The function sets this to the actual number of jobs for which it retrieves information.

pBuf&

is a pointer to an array of one or more structures that receive the requested information. The DISCAPSULEJOBINFO_L1 structure is:

```
Type DISCAPSULEJOBINFO_L1
    cb As Integer                       ' Size of this entry, in bytes
    bPriority As Integer                ' Job priority; specify 1 for the lowest priority to
                                                31 for the highest priority
    bState As Integer                   ' Job state
    id As Long                          ' Job identifier
    pszOwner As String                  ' Desktop that submitted the job
    pszDescription As String            ' Job's description
    pszAliasName As String              ' Scheduler alias name
    timeSubmitted As Long               ' Time when job was submitted
    timeScheduled As Long               ' Job's scheduled starting time
    secsMaxStartDelay As Long           ' Maximum delay past scheduled time,
                                                in  seconds
    secsMaxExecution As Long            ' Maximum execution time, in seconds
End Type
Global Const DISCAPSULEJOBINFO_L1_FORMAT$ = 'ICCLP8 P8 P8 LLLL'
```

The bState member returns one of the numerical codes in Table 39 to indicate current state of execution.

Table 39. Capsule job-state codesFm Variable:Table Sheet

Job-state code	Value	Description
DIS_CJI_WAITING	1	The job is waiting for its scheduled time.
DIS_CJI_READY	2	The job is waiting for an execution resource.
DIS_CJI_HELD	3	The job is being held.
DIS_CJI_RUNNING	4	The job is currently executing.
DIS_CJI_ABENDED	5	The job has been cancelled or has been abnormally terminated due to a Capsule execution error.

pcbBuf%

is the size of the pBuf& buffer, in bytes, as obtained from DisCreateBuffer. Upon completion, the function returns the actual length of the data it obtained.

fFlags%

specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

Specify the DIS_QCI_OWNEDflag to return information about only those jobs that were submitted by the user issuing the request. Otherwise, the returned information includes jobs submitted by all users.

hdis&

is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example retrieves information about the first 10 jobs in the execution queue, then returns that information in the capsuleStatus buffer without waiting for completion.

```
Dim retCode As Integer
Dim bufferSize As Integer
Dim numJobs As Integer
Dim jobNumber As Integer
Dim capsuleStatus As Long
...
retCode = DisSubmitCapsule("Budget Capsule",
  ↪"Capsule Execution Drawer", jobParameters, jobNumber,
  ↪DIS_NOWAIT, sessionHandle)
   ' DisQueryCapsuleInfo needs the job number that
   '   DisSubmitCapsule sets.
...
numJobs = 1
capsuleStatus = DisCreateBuffer(DISCAPSULEJOBINFO_L1_FORMAT,
  ↪numJobs, bufferSize)
retCode = DisQueryCapsuleInfo(DIS_QCI_JOB, jobNumber,
  ↪DIS_QCIL_STANDARD, numJobs, capsuleStatus, bufferSize,
  ↪DIS_WAIT, sessionHandle)
```

DisQueryObjectInfo

This function checks the type, size, creation time, and other information of one or more desktop objects. The function returns the information in one or more DISOBJECTINFO structures.

Desktop Equivalent

The most equivalent action, if the user were at a IDS workstation, would be to open a container holding the objects and show the container's contents with their sizes and dates.

Function Declaration

```
Declare Function DisQueryObjectInfo% Lib 'metrapiw.dll'
 (ByVal pszObjectSpec$, ByVal uInfoLevel%, piStart%, pcEntries%,
 ByVal pBuf&, ByVal pcbBuf%, ByVal fFlags%, ByVal hdis&)
```

where:

pszObjectSpec$	is a string specifying the desired desktop objects. (See "Identifying Desktop Objects" on page 10.)
uInfoLevel%	specifies the amount of information to be retrieved. Table 40 on page 152 lists the valid values.

Table 40. Detail selection codes for DisQueryObjectInfo ()

Selection code	Value	Description
DIS_QOIL_TEST	0	Check for the object's existence and return no other information.
DIS_QOIL_STANDARD	1	Return information in the DISOBJECTINFO structure.
DIS_QOIL_FULLPATH	2	Return the full path name of the object. With this value, you cannot use wildcard asterisks (*) in the function.

piStart% is an integer specifying the object to be queried. Use 0 to begin the search with the first object in a desktop container, 1 to begin with the second object, and so on. If you store the index value between calls, a series of calls can search the container sequentially. The function sets this parameter to 0 when it has investigated the last object in the container.

pcEntries% is an integer specifying the quantity of desktop objects whose information is to be retrieved. The function sets this to the actual number of objects for which it retrieves information.

pBuf& is a pointer to an array of one or more structures that receive the requested information. You get this pointer from DisCreateBuffer. The DISOBJECTINFO structure is:

```
Global Const DIS_OBJECT_NAME_LEN 8      ' Maximum length of a desktop
                                                          object's  name
Type DISOBJECTINFO
     cb As Integer                          ' Size of this entry, in bytes
     bObjType As Integer                ' Numerical code indicating the type of desktop
                                              object, as shown in Table 2 on page 13
     fbFlags As Integer                  ' Reserved for future use; specify   (zero)
     id As Long                                ' Unique identifier for each desktop object, to
                                                   distinguish between objects with identical
                                                  names
     timeCreation As Long             ' Time and date when the object was created
     timeLastRead As Long            ' Time and date when it was last read
     timeLastWrite As Long           ' Time and date when it was last written
     cbObject As Long                     ' Size of the object, in bytes
     szName As String                     ' Object's name (maximum length
                                                          is  DIS_OBJECT_NAME_LEN)
End Type
Global Const DISOBJECTINFO_FORMAT$ = 'ICCLLLLLZ8 '
```

The timeCreation, timeLastRead, and timeLastWrite members are integers measuring the number of seconds since the start of January 1, 1900.

pcbBuf% is the size of the pBuf& buffer, in bytes, as obtained from DisCreateBuffer. Upon completion, the function returns the actual length of the data it obtained.

fFlags% specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

Specify the DIS_DELETE_SOURCE or DIS_DELETE_IFEMPTY flag to move the objects into the desktop's Discard folder after the operation.

Specify the DIS_DELETE_EXPUNGE flag to bypass the normal Discard folder and actually remove the objects from the desktop.

You can also specify the flags listed in Table 41 on page 153.

Table 41. Flags for DisQueryObjectInfo ()

Flag	Value	Description
DIS_QOI_COUNT_CONTENTS	1	Count the contents of the container and return them in the cbObject field. Without DIS_QOI_COUNT_CONTENTS, the function returns the container's size, in bytes, in the cbObject field.
DIS_QOI_MININFO	2	Speed the processing by not returning the time Last Read, timeLastWrite, cbFile, and attrFilefields.

hdis Is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Examples

The first example queries the Mail Tray to see if it has any contents.

```
Dim bufferSize As Integer
Dim numMessages As Integer
Dim retCode As Integer
...
retCode = DisQueryObjectInfo("\Mail Tray\", DIS_QOIL_TEST,
0,  ↪numMessages, 0, bufferSize, DIS_QOIL_COUNT_CONTENTS,
  ↪sessionHandle)
```

The following example then retrieves the contents of the Mail Tray, waiting for completion before it returns.

```
Dim disBuffer As Long
Dim vbBuffer() As DISOBJECTINFO
Dim index As Integer
...
disBuffer = DisCreateBuffer(DISOBJECTINFO_FORMAT,
  ↪ numMessages, bufferSize)
retCode = DisQueryObjectInfo("\Mail Tray\",
  ↪ DIS_QOIL_STANDARD, index, numMessages, disBuffer,
  ↪ bufferSize, DIS_WAIT, sessionHandle)
ReDim vbBuffer(1 To numMessages)
retCode = DisBufferToVBArray(vbBuffer(1), Len(vbBuffer(1)),
  ↪ disBuffer, bufferSize, DISOBJECTINFO_FORMAT,
  ↪ numMessages)
<Visual Basic code to operate on  vbBuffer data >
```

DisQueryResult

This function checks the result of an asynchronous operation on the IDS network. You can call DisQueryResult to determine whether a prior request has been executed and, if so, to return one of the completion codes shown in Table 42.

Table 42. Completion codes for DisQueryResult ()

Completion code	Description
ERROR_MET_REQUEST_PENDING	The specified operation is still awaiting completion.
ERROR_MET_SUBREQUEST_ERROR	If you specified iSubRequest = 0, at least one operation within the batch returned an error. For more specific information about the error, specify the index of the specific operation or check the results buffer used in the DisSubmitBatch function.
ERROR_MET_REQUEST_NOT_FOUND	The index you specified was not within the number of operations in the most recent batch request.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisQueryResult% Lib 'metrapiw.dll' (ByVal iSubRequest%, ByVal fFlags%, ByVal hdis&)

where:

iSubRequest%	specifies an operation within a batch request. If the operation was not a batch request, or you want to check a global batch result, specify 0 (zero).
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force this function to return as soon as the request has been sent to the ODAS server.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example checks the result of the first function in a previous batch request, then returns without waiting for completion:

```
Dim retCode As Integer
retCode = DisQueryResult(1, DIS_NOWAIT, sessionHandle)
```

DisQuerySessionInfo

This function checks one of: the current container path; the number of pending system messages, number of items in the mail tray, and number of new items in the mail tray since the previous query; or information about the ODAS service. The function returns the information as a string or structure, as described in Table 43.

Desktop Equivalent

If the user were at a IDS workstation, the action equivalent to selection code 1 would be to look to see if the desktop is displaying the IMPORTANT MESSAGE button, and to open and look at the Mail Tray. There is no workstation action equivalent to checking the number of new items in the Mail Tray since the previous query.

There is no workstation action equivalent to selection code 2.

Function Declaration

```
Declare Function DisQuerySessionInfo% Lib 'metrapiw.dll' (ByVal
iInfo%, pBufAs Any, pcbBuf%, ByVal fFlags%, ByVal hdis&)
```

where:

iInfo%	is the information selection code, specifying which information you want to retrieve. Table 43 lists the valid values.

Table 43 (Page 1 of 2). Information selection codes for DisQuerySessionInfo ()

Selection code	Value	Description
DIS_QSI_MSGCOUNT	1	Return a DISMSGCOUNT structure, containing various information about pending mail and system messages.
DIS_QSI_CURRENTPATH	2	Return the current container path as a string.

Table 43 (Page 2 of 2). Information selection codes for DisQuerySessionInfo ()

Selection code	Value	Description
DIS_QSI_SERVICEINFO	3	Return a DISSERVICEINFO structure, containing various information about the ODAS service.

pBuf — is a pointer either to a string that receives the current container path or to a buffer for a DISMSGCOUNT or DISSERVICEINFO structure containing the other requested information, depending on iInfo. You get this pointer from DisCreateBuffer for the appropriate structure.

The DISMSGCOUNT structure is:

```
Type DISMSGCOUNT
    cSysMessages As Integer     ' Number of pending system messages
    cMail As Integer                    ' Number of items in the Mail Tray
    cNewMail As Integer             ' New items in Mail Tray since the prior query
End Type
```

The DISSERVICEINFO structure is:

```
Global Const DIS_SERVICENAME_LEN     64    ' Maximum service-name length
Type DISSERVICEINFO     idSession As Long                    ' Current-session identifier
    szServiceName As String  DIS_SERVICENAME_LEN    ' Service name
    szNetworkName As String  DIS_SERVICENAME_LEN    ' Network name End Type
```

pcbBuf% — is the size of the pBuf buffer, in bytes, as obtained from DisCreateBuffer. Upon completion, the function returns the actual length of the data it obtained.

fFlags% — specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

hdis& — is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Examples

The first example identifies the IDS desktop's current container path, then waits for completion.

```
Dim retCode As Integer
Dim bufferSize(1 To 3) As Integer
...
retCode = DisQuerySessionInfo(DIS_QSI_CURRENTPATH,
  ↪ ByVal pathBuffer, bufferSize(1), DIS_WAIT,
  ↪ sessionHandle)
```

The second example tells how many system messages, mail messages, and new mail messages are waiting on the desktop; it then waits for completion.

```
Dim msgCount As DISMSGCOUNT
...
retCode = DisQuerySessionInfo(DIS_QSI_MSGCOUNT,
  ↪ msgCount, bufferSize(2), DIS_WAIT, sessionHandle)
```

The third example retrieves the current-session identifier, ODAS service name, and IDS network name, then waits for completion.

```
Dim serviceInfo As DISSERVICEINFO
...
retCode = DisQuerySessionInfo(DIS_QSI_SERVICEINFO,
  ↪ serviceInfo, bufferSize(3), DIS_WAIT, sessionHandle)
```

DisSetCapsuleStatus

This function specifies whether to cancel, purge, hold, release, or modify the specified Capsule application in the Capsule Services queue, or to set various job parameters in a DISCAPSULEJOBPARMS structure.

Desktop Equivalent

If the user were at a IDS workstation, the action equivalent to action code 1 would be to open the Capsule Services icon, select the job to be cancelled, and click on the Cancelbutton.

There is no workstation action equivalent to the other action codes.

Function Declaration

Declare Function DisSetCapsuleStatus% Lib 'metrapiw.dll' (ByVal uAction%, ByVal idJob&, ByVal pjobp&, ByVal fFlags%, ByVal hdis&)

where:

uAction% — specifies a numerical code representing the action to be applied to the specified Capsule application. Table 44 lists the valid codes.

Table 44 (Page 1 of 2). Action codes for DisSetCapsuleStatus ()

Action code	Value	Description
DIS_SCS_CANCEL	1	Cancel the Capsule job from the queue. If the job is periodic, and there are more instances of it in the queue, DIS_SCS_CANCEL cancels only the one instance with the earliest scheduled starting time.
DIS_SCS_PURGE	2	Purge a periodic Capsule job.
DIS_SCS_HOLD	3	Hold the job until it is released by the DIS_SCS_RELEASE code.

Table 44 (Page 2 of 2). Action codes for DisSetCapsuleStatus ()

Action code	Value	Description
DIS_SCS_RELEASE	4	Release the held job.
DIS_SCS_MODIFY	5	Modify the Capsule job parameters.

idJob& is the identifying number of the Capsule application that is to be affected. The DisSubmitCapsule function provides this number.

pjobp& is a pointer to a buffer for the structure that specifies the job parameters. If you specify an action code other than DIS_SCS_MODIFY, specify a null pointer as 0& for pjobp. The DISCAPSULEJOBPARMS structure is:

```
Type DISCAPSULEJOBPARMS
    cb As Integer
    bPriority As Integer              ' Job priority; specify 1 for the lowest priority
                                            to 31 for the highest priority
    fbFlags As Integer                ' Bit mask relating to when the job is to be
                                            executed, as listed in Table 5 on page 17
    flMask As Long                    ' Bit mask telling which parameters are set in
                                            various structure members, as listed in
                                            Table 4 on page 16
    pszDescription As String          ' Job description
    pszAliasName As String            ' Scheduler's alias name
    timeScheduled As Long             ' Scheduled starting time
    secsMaxStartDelay As Long         'Maximum delay past scheduled time,
                                                                in  seconds
    secsMaxExecution As Long          ' Maximum execution time, in seconds
End Type
Global Const DISCAPSULEJOBPARMS_FORMAT$ = 'ICCLP5 P15LLL'
```

The pszDescription and pszAliasName members are pointers to strings.

The timeScheduled member is an integer measuring the number of seconds since the start of January 1, 1900.

fFlags% specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

hdis& is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example cancels the previously submitted Capsule application, then returns without waiting for completion.

```
Dim retCode As Integer
Dim jobNumber As Integer
...
retCode = DisSubmitCapsule(..., jobNumber, ...)
' DisSetCapsuleStatus needs the job number that
'   DisSubmitCapsule sets.
...
retCode = DisSetCapsuleStatus(DIS_SCS_CANCEL, jobNumber, 0&,
  ↪ DIS_NOWAIT, sessionHandle)
```

DisSetCapsuleVars

This function assigns values to the Capsule application's @-variables. You specify the job parameters in a DISCAPSULEVARINFO structure.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to select the Show Controls button in a Capsule application to open its User Input Controlwindow and type values for one or more @-variables.

Function Declaration

Declare Function DisSetCapsuleVars% Lib 'metrapiw.dll' (ByVal pszCapsuleName$, ByVal pcapv&, ByVal cVars%, ByVal fFlags%, ByVal hdis&)

where:

pszCapsuleName$	specifies the name of the Capsule application whose variables are to be set.
pcapv&	is a pointer to an array of one or more structures that specify the Capsule application's @-variables. The DISCAPSULEVARINFO structure is:

```
Global Const DIS_CAPSULEVAR_NAME_LEN = 4
Global Const DIS_CAPSULEVAR_DESC_LEN = 5  ' Maximum description length
Global Const DIS_CAPSULEVAR_VALUE_LEN = 255 ' Maximum value length
Global Const DIS_CAPSULEVAR_TYPE_DEFAULT =   ' Preserve existing    type
                                                          and set new type
Global Const DIS_CAPSULEVAR_TYPE_NUM = 1 ' Variable type is a number
Global Const DIS_CAPSULEVAR_TYPE_DATE = 2 ' Variable type is a date
Global Const DIS_CAPSULEVAR_TYPE_TEXT = 3 ' Variable type is a string
```

```
Type DISCAPSULEVARINFO
    szName As String   DIS_CAPSULEVAR_NAME_LEN          ' Variable name
    ulType As Long                                      ' Variable type
    pszDescription As String          ' Variable's description
    pszValue As String                  ' Variable's value
End Type
Global Const DISCAPSULEVARINFO_FORMAT$ = 'S4LP5 P255'
```

	The pszDescription and pszValue members are pointers to strings. If either pointer is 0 (zero), the function does not change the corresponding value or description for the @-variable. To delete either a value or a description, specify a null string (pszDescription= Chr$(0) or pszValue= Chr$(0)).
cVars%	specifies the number of DISCAPSULEVARINFO structures that pcapv& points to.
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.
	Specify the DIS_OVERWRITE or DIS_OVERWRITE_VARS flag to replace the Capsule application's current variables instead of merging the new and old lists of variables. For example, assume that the existing list has @A and @B and pcapv& specifies only @B. With DIS_OVERWRITE_VARS, the resulting variable list has only @B. Without DIS_OVERWRITE_VARS, the resulting variable list has the old @A and the new @B.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example sets the current @-variables of a specified Capsule application, then returns without waiting for completion.

The example assumes that the associated form contains three text boxes named txtCapsule, txtDescription, and txtValue, where the user can type the name of the Capsule application and the description and value of a selected @-variable. It also assumes that a grid control is established with three columns for the @-variables' names, descriptions, and values, as shown in Figure 7 on page 161.

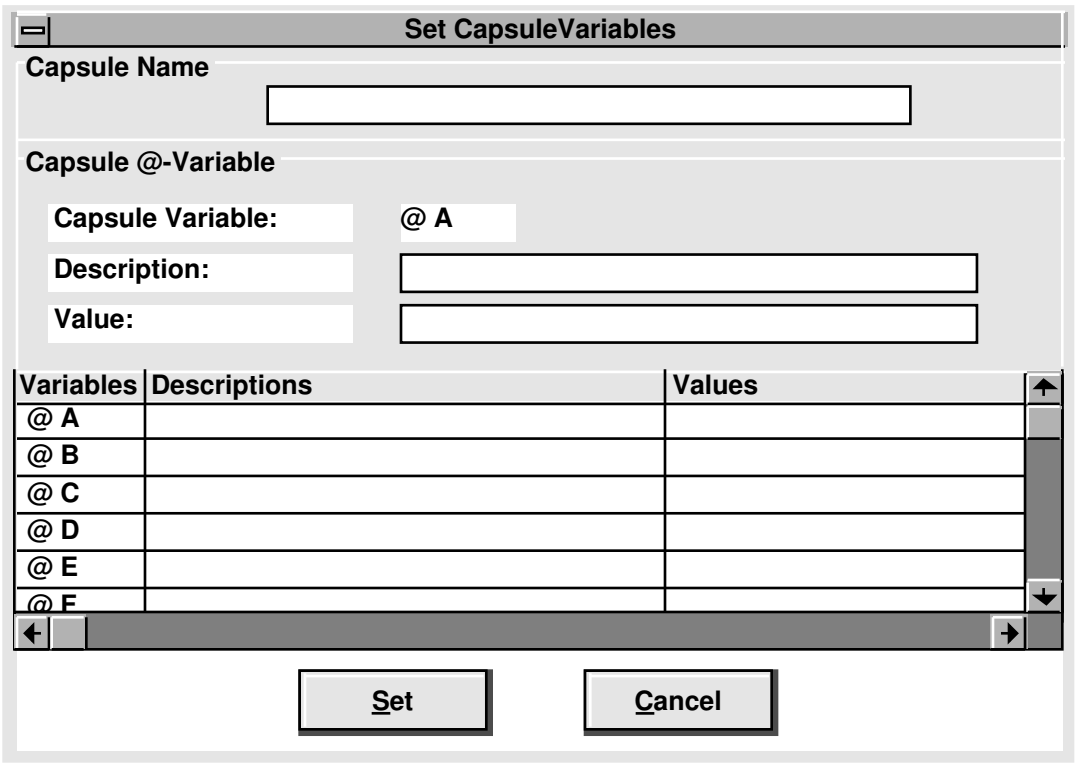


Figure 7. Sample DisSetCapsuleVars window with grid control

```
Dim retCode As Integer
Dim bufferSize As Integer
Dim numVars As Integer
Dim disBuffer As Long
Dim vbBuffer() As DISCAPSULEVARINFO
...
<Visual Basic code to set up the grdAtVar grid, looping on
numVars to copy user-entered description and value for each
@-variable into vbBuffer array >
...
disBuffer = DisCreateBuffer(DISCAPSULEVARINFO_FORMAT,
  ↪ numVars, bufferSize)
ReDim vbBuffer(1 To numVars)
retCode = DisVBArrayToBuffer(disBuffer, bufferSize,
  ↪ vbBuffer(1), vbBuffer(1), DISCAPSULEVARINFO_FORMAT,
  ↪ numVars)
retCode = DisSetCapsuleVars(txtCapsule.Text, disBuffer,
  ↪ numVars, DIS_NOWAIT, sessionHandle)
retCode = DisFreeBuffer(disBuffer)
```

You might prefer to call DisGetCapsuleVars first to display the existing @-variables.

For instance, a user, wanting to set @-variable @A before executing the Budget Capsule application, might type Budget Capsule in the txtCapsule space, then select @A in the grid and type Western Region in the txtDescription space and 1234 in the txtValue space.

DisSetConversionTable

This function modifies the type-conversion table. The system uses this table when copying a personal computer file to a desktop or a desktop object to a personal computer.

You specify each conversion table entry in a DISCONVERTENTRY structure.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisSetConversionTable% Lib 'metrapiw.dll' (ByVal pcnveTable, ByVal ccnveTable%, ByVal uDefaultFileType%, ByVal fFlags%, ByVal hdis&)

where:

pcnveTable — is a pointer to an array of one or more structures that specify entries in the conversion table. The DISCONVERTENTRY structure is:

```
Type DISCONVERTENTRY
    pszFileExt As String          ' File type on the personal computer (characters
                                      after the last period in a file identifier)
    bFileType As Integer          ' Personal-computer file type; choose a
                                      numerical code from Table 1 on page 12
    bObjType As Integer           ' Desktop-object type; choose a numerical code
                                      from Table 2 on page 13
    cchLineWidth As Integer       ' Maximum line width when bObjType
                                      is  DIS_FILETYPE_TEXT
End Type
Global Const DISCONVERTENTRY_FORMAT$ = 'P1 CCI'
```

ccnveTable% — specifies the number of entries in the new conversion table. If you do not specify DIS_OVERWRITE, ccnveTable is the number of entries added to the table.

uDefaultFileType% — specifies the default personal-computer file type to use if a DisPutObject or DisPutMultObjects call specifies a source file whose file-name extension does not correspond to any entry in the conversion table.

fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_OVERWRITE flag to replace the current conversion table instead of merely appending the new entries.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example modifies the conversion table so personal-computer files with a specified suffix will, when copied to a desktop, be converted to Text documents. The function then returns without waiting for completion.

The example assumes that the associated form contains three text boxes named txtFileExtension, txtPCFileType, and txtDISFileType, where the user can specify a file-extension suffix and the file and object types to be associated with that suffix. The example also assumes the existence of two functions:

GetPCFileType, which converts the string in txtPCFileType (such as .WRD) into one of the file-type constants (such as DIS_FILETYPE_TEXT) listed in Table 1 on page 12

GetDISFileType, which converts the string from txtDISFileType into one of the object-type constants (such as DIS_OBJTYPE_DOCUMENT) listed in Table 1 on page 12.

```
Dim retCode As Integer Dim bufferSize As Integer Dim vbConvertEntry As
DISCONVERTENTRY Dim bufferSize As Integer Dim tableEntry As Integer
vbConvertEntry.pszFileExt = txtFileExtension.Text & Chr$(0)
vbConvertEntry.bFileType = GetPCFileType(txtPCFileType.Text)
vbConvertEntry.bObjType =GetDISFileType(txtDISFileType.Text)
vbConvertEntry.cchLineWidth = 0 tableEntry =
DisCreateBuffer(DISCONVERTENTRY_FORMAT, 1, bufferSize retCode =
DisBufferToVBType(vbConvertEntry, Len(vbConvertEntry), tableEntry, bufferSize,
DISCONVERTENTRY_FORMAT) retCode = DisSetConversionTable(tableEntry, entries,
DIS_FILETYPE_TEXT, DIS_NOWAIT, sessionHandle) retCode =
DisFreeBuffer(tableEntry)
```

For instance, suppose a user wants the table to consider files with the suffix .WRD to be equivalent to formatted ASCII text (DIS_FILETYPE_SHOWTEXT) on the client computer and DIS_OBJTYPE_DOCUMENT on the IDS desktop. With this example, the user might type .WRD in the txtFileExtension space, DIS_FILETYPE_SHOWTEXT in the txtPCFileType space, and DIS_OBJTYPE_DOCUMENT in the txtDISFileType space.

DisSetCurrentContainer

This function specifies the desktop container on which a future function is to be performed. This specification determines how relative object names are mapped. (Relative object names are names that do not begin with the current path separator, as set by the DisSetSessionInfo function or defaulted to the backslash (\) character.)

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

```
Declare Function DisSetCurrentContainer% Lib 'metrapiw.dll'
(ByVal pszContainerName$, ByVal fFlags%, ByVal hdis&)
```

where:

pszContainerName$	specifies the name of the new current container. (See “Containers” on page 10.)
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example sets the current default container, then returns without waiting for completion.

The example assumes that the associated form contains a text box named txtCurrentContainer, where the user can specify a container name (or a backslash for the top level of the desktop).

```
Dim retCode As Integer retCode = DisSetCurrentContainer
(RTrim(txtCurrentContainer.Text) & Chr$(0), DIS_NOWAIT, sessionHandle)
```

For instance, a user wanting a future function to operate on the In Basket folder might type In Basket in the txtCurrentContainer space.

DisSetEventHandler

This function cannot be called from Visual Basic; it must be called from C. On “DisSetEventHandler” on page 77,you will find information on the C-language API function DisSetEventHandler, which establishes an event-handling procedure for the specified session.

DisSetPassword

This function changes the password of the connected desktop. The session remains logged on.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to select the desktop's Change Password button and enter a new password.

Function Declaration

Declare Function DisSetPassword% Lib 'metrapiw.dll' (ByVal pszOldPassword$, ByVal pszNewPassword$, ByVal fFlags%, ByVal hdis&)

where:

pszOldPassword$	specifies the current password.
pszNewPassword$	specifies the new password.
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14.Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example changes the password, then returns without waiting for completion. The example assumes that the associated form contains two text boxes named txtOldPassword and txtNewPassword, where the user can specify the current and desired passwords.

```
Dim retCode As Integer retCode = DisSetPassword
(RTrim(txtOldPassword.Text) & Chr$(0), (RTrim(txtNewPassword.Text) &
Chr$(0), DIS_NOWAIT, sessionHandle)
```

DisSetSessionInfo

This function either changes the separator character to be used in path names on the desktop, or sets the IBM code-page number.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisSetSessionInfo% Lib 'metrapiw.dll' (ByVal iInfo%, pBuf As Any, ByVal cbBuf%, ByVal fFlags%, ByVal hdis&)

where:

iInfo% is the selection code, specifying which information you want to change. Table 45 lists the valid values.

Table 45. Information selection codes for DisSetSessionInfo ()

Selection code	Value	Description
DIS_SSI_CODEPAGE	1	The function specifies the code-page number in use by the personal computer. With this selection code, the pBuf value is of type Integer and the cbBuf% length is 2.
DIS_SSI_PATHSEPCHAR	2	The function specifies the separator character, to be used in path names, to any other printable ASCII character. The default character is a backslash (\ ). With this selection code, the pBuf value is of type string*1 and the cbBuf% length is 1.

pBuf is either the code-page number (of type Integer) or the separator character (of type string*1), depending on iInfo%.

cbBuf% is the size of the pBuf buffer, in bytes. This size is 2 if iInfo% is DIS_SSI_CODEPAGE, or 1 if iInfo% is DIS_SSI_PATHSEPCHAR.

fFlags% specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

hdis& is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Examples

This example sets the path separator to the vertical bar (|), then waits for completion.

```
Dim separator As String * 1 Dim retCode As Integer separator =
'|' retCode = DisSetSessionInfo(DIS_SSI_PATHSEPCHAR, ByVal
separator, Len(separator), DIS_WAIT, sessionHandle)
```

This example sets the IBM code-page number, then waits for completion. The example assumes that the associated form contains a text box named txtCodePage, where the user can type the desired code number.

```
Dim retCode As Integer retCode =
DisSetSessionInfo(DIS_SSI_CODEPAGE, Val(txtCodePage.Text), 2, DIS_WAIT,
sessionHandle)
```

DisSubmitBatch

This function submits queued functions for execution. The DisInitBatch function starts a queue of submitted functions. All subsequent functions are queued until DisSubmitBatch is reached; at that point, the queued functions are submitted to the ODAS server for execution.

Each function in the batch can return errors due to local conditions (such as a missing local file in a DisPutObject call). In this situation, you can use the DisTermBatch command to terminate the batch instead of submitting it to the ODAS server. (See “DisSubmitCapsule” on page 168.)

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisSubmitBatch% Lib 'metrapiw.dll' (pjResults%, ByVal cResults%, ByVal fFlags%, ByVal hdis&)

where:

pjResults%	is a pointer to the buffer that is to receive the results of the functions that this operation submits.
cResults%	specifies the number of entries in the array that is to be passed to pjResults%.
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server. Specify the DIS_SB_SUBRESULTSflag to retrieve the results of each subrequest as soon as the server has completed the request. Otherwise, the function retrieves results when all subrequests are completed.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example enables, creates, and submits a batch request comprising four desktop operations. The result codes of DisGetObject, DisGetSysMessage, DisCopyObject, and DisLogoff are returned in batchResults.

```
Dim batchResults() As Integer Dim retCode As Integer .retCode =
DisInitBatch(0, sessionHandle) . retCode = DisGetObject('Next
Report', 'NEXTRPT.WKS', DIS_FILETYPE_WKS, 0, 0, DIS_NOWAIT Or
DIS_DELETE_SOURCE, sessionHandle) . retCode =
DisGetSysMessage(numMessages, disBuffer, bufferSize, DIS_NOWAIT,
sessionHandle) . retCode = DisCopyObject('Next Report',
'Copy of Next Report', DIS_OBJTYPE_DEFAULT, DIS_NOWAIT,
sessionHandle) retCode = DisLogoff(DIS_NOWAIT, sessionHandle) ReDim
batchResults(1 To 4) retCode = DisSubmitBatch(batchResults, 4, DIS_NOWAIT,
sessionHandle)
```

DisSubmitCapsule

This function submits a copy of the specified Capsule application to the Capsule Services, which then runs the copy once or periodically, according to the schedule you give it. If you want the results to replace the original Capsule application, use the DIS_OVERWRITE flag.

You specify the DisSubmitCapsule job parameters in a DISCAPSULEJOBPARMS structure.

This function returns as soon as the Capsule job has been queued for execution. It does not wait for the job to be executed. You can use DisWaitCapsuleStatusto wait for a Capsule job to complete execution.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to copy the Capsule icon onto the closed Capsule Services icon.

Function Declaration

```
Declare Function DisSubmitCapsule% Lib 'metrapiw.dll' (ByVal
pszCapsuleName$, ByVal pszDestContainerName$, ByVal pjobp&, pidJob&,
ByVal fFlags%, ByVal hdis&)
```

where:

pszCapsuleName$	is the name of the Capsule application to be submitted.
pszDestContainerName$	is the name of the container into which the Capsule application is to be copied after execution. (See “Containers” on page 10.) To discard the executed copy of the Capsule application after it completes execution, specify a null pointer as 0&.
pjobp&	is a pointer to a buffer for the structure that specifies the job parameters. If you want to use the default parameters, specify a null pointer as 0& for pjobp. The DISCAPSULEJOBPARMS structure is:

```
Type DISCAPSULEJOBPARMS
      cb As Integer
      bPriority As Integer          ' Job priority; specify 1 for the lowest priority
                                          to 31 for the highest priority
      fbFlags As Integer            ' Bit mask relating to when the job is to be
                                          executed, as listed in Table 5 on page 17
      flMask As Long                ' Bit mask telling which parameters are set in
                                          various structure members, as listed in
                                           Table 4 on page 16
      pszDescription As String      ' Job description
      pszAliasName As String        ' Scheduler's alias name
      timeScheduled As Long         ' Scheduled starting time
           secsMaxStartDelay As Long ' Maximum delay past scheduled time,
                                          in  seconds
           secsMaxExecution As Long ' Maximum execution time, in seconds
End Type
Global Const DISCAPSULEJOBPARMS_FORMAT$ = 'ICCLP5 P15LLL'
```

The timeScheduled, secsMaxStartDelay, and secsMaxExecution members are long integers measuring the number of seconds since the start of January 1, 1900. Use the DisMkTime function to convert each time specification from a DISDATEANDTIME structure into a single long integer.

pidJob& receives the job's identification number. Functions DisQueryCapsuleInfo, DisSetCapsuleStatus, and DisWaitCapsuleStatus require this number.

fFlags% specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.

Specify DIS_OVERWRITE to cause the results to replace the original Capsule application.

hdis& is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example submits a Capsule application to be copied to the Capsule Execution Drawer container and then executed; it then waits for completion.

The example assumes that the associated form contains three text boxes named txtCapsule, txtContainer, and txtJobDescription, where the user can type the name of the Capsule application to be run, the container into which the Capsule application is to be copied after execution, and the job description.

```
Dim timeStructure As DISDATEANDTIME
Dim timeSpec As Long
timeSpec = DisMkTime(timeStructure)
```

Dim retCode As Integer Dim jobNumber As Integer Dim jobParm As DISCAPSULEJOBPARMS Dim disBuffer As Long Dim bufferSize As Integer jobParm.fbFlags = 0 jobParm.cb = DisCFormatSize(DISCAPSULEJOBPARMS_FORMAT, DISUB_TOTAL_SIZE) jobParm.bPriority = 31 jobParm.flMask = DIS_CJP_PRIORITY ' Flag use of bPriority <Visual Basic code to convert user-entered date and times into integers in thetimeStructurestructure> jobParm.timeScheduled = DisMkTime(timeStructure) jobParm.pszDescription = txtJobDescription.Text & Chr$(0) jobParm.flMask = jobParm.flMask Or DIS_CJP_DESCRIPTION <Similar code to handle other members ofjobParm> disBuffer = DisCreateBuffer(DISCAPSULEJOBPARMS_FORMAT, 1, bufferSize) retCode = DisVBTypeToBuffer(disBuffer, bufferSize, jobParm, Len(jobParm), DISCAPSULEJOBPARMS_FORMAT) retCode = DisSubmitCapsule(txtCapsule.Text, txtContainer.Text, disBuffer, jobNumber, DIS_WAIT, sessionHandle)

For instance, a user wanting to run the Budget Capsule application and leave the results in the Capsule Execution Drawer container might type Budget Capsule in the txtCapsule space, Capsule Execution Drawer in the txtContainer space, and Sales budget for Junein the txtJobDescription space.

DisSubmitCapsuleAndSetVars

This function submits a copy of the specified Capsule application to the Capsule Services, which then runs the copy once or periodically, according to the schedule you give it. If you want the results to replace the original Capsule application, use the DIS_OVERWRITE flag.

When submiting a Capsule job, you specify the parameters in a DISCAPSULEJOBPARMS structure.

This function also assigns values to the Capsule application's @-variables. You specify this job parameters in a DISCAPSULEVARINFO structure.

Desktop Equivalent

The equivalent action, if the user were at a IDS workstation, would be to copy the Capsule icon onto the closed Capsule Services icon, or to to select the Show Controls button in a Capsule application to open its User Input Controlwindow and type values for one or more @-variables.

Function Declaration

Declare Function DisSubmitCapsuleAndSetVars% Lib 'metrapiw.dll' (ByVal pszCapsuleName$, ByVAL pszDestContainerName$, ByVal pszCapsuleNameOut$, ByVal pjobp&, pidJob&, ByVal pcapv&, ByVal cVars%, ByVal fFlags%, ByVal hdis&)

where:

pszCapsuleName$	specifies the name of the Capsule application to be submitted or whose variables are to be set.

pszDestContainerName$	is the name of the container into which the Capsule application is to be copied after execution. (See “Containers” on page 10.) To discard the executed copy of the Capsule application after it completes execution, specify a null pointer as 0&.
pszCapsuleNameOut	is the name used to rename the completed Capsule job output. Pass 0& or CHR$(0) to not rename the Capsule job output.
pjobp&	is a pointer to a buffer for the structure that specifies the job parameters. If you want to use the default parameters, specify a null pointer as 0& for pjobp. The DISCAPSULEJOBPARMS structure is:

```
Global Const DIS_CAPSULEVAR_NAME_LEN = 4
Global Const DIS_CAPSULEVAR_DESC_LEN = 5  ' Maximum description length
Global Const DIS_CAPSULEVAR_VALUE_LEN = 255 ' Maximum value length
Global Const DIS_CAPSULEVAR_TYPE_DEFAULT =      ' Preserve existing            type
                                                    and set new type
Global Const DIS_CAPSULEVAR_TYPE_NUM = 1    ' Variable type is a number
Global Const DIS_CAPSULEVAR_TYPE_DATE = 2    ' Variable type is a date
Global Const DIS_CAPSULEVAR_TYPE_TEXT = 3    ' Variable type is a string

Type DISCAPSULEVARINFO
     szName As String   DIS_CAPSULEVAR_NAME_LEN                    ' Variable name
     ulType As Long                              ' Variable type
     pszDescription As String           ' Variable's description
     pszValue As String                    ' Variable's value
End Type
Global Const DISCAPSULEVARINFO_FORMAT$ = 'S4LP5 P255'

Type DISCAPSULEJOBPARMS
     cb As Integer
     bPriority As Integer                  ' Job priority; specify 1 for the lowest priority
                                                 to 31 for the highest priority
     fbFlags As Integer                    ' Bit mask relating to when the job is to be
                                                 executed, as listed in Table 5 on page 17
     flMask As Long                          ' Bit mask telling which parameters are set in
                                                 various structure members, as listed in
                                                 Table 4 on page 16
     pszDescription As String          ' Job description
     pszAliasName As String             ' Scheduler's alias name
     timeScheduled As Long             ' Scheduled starting time
     secsMaxStartDelay As Long       ' Maximum delay past scheduled time,
                                           in  seconds
     secsMaxExecution As Long       ' Maximum execution time, in seconds
End Type
Global Const DISCAPSULEJOBPARMS_FORMAT$ = 'ICCLP5 P15LLL'
```

	The pszDescription and pszValue members are pointers to strings. If either pointer is 0 (zero), the function does not change the corresponding value or description for the @-variable. To delete either a value or a description, specify a null string (pszDescription= Chr$(0) or pszValue= Chr$(0)).
cVars%	specifies the number of DISCAPSULEVARINFO structures that pcapv& points to.
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14. Specify the DIS_NOWAIT flag to force the function to return as soon as the request has been sent to the ODAS server.
pidJob&	receives the job's identification number. Functions DisQueryCapsuleInfo, DisSetCapsuleStatus, and DisWaitCapsuleStatus require this number.
pcapv&	specifies the 2-variable entries. Specify the DIS_OVERWRITE or DIS_OVERWRITE_VARS flag to replace the Capsule application's current variables instead of merging the new and old lists of variables. For example, assume that the existing list has @A and @B and pcapv& specifies only @B. With DIS_OVERWRITE_VARS, the resulting variable list has only @B. Without DIS_OVERWRITE_VARS, the resulting variable list has the old @A and the new @B.
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

For examples of how to submit and set the Capsule variables, see “DisSetCapsuleVars” on page 159 and “DisSubmitCapsule” on page 168.

DisTermBatch

This function terminates a batch submission, discarding any calls that have not yet been submitted to the ODAS server. (See “DisInitBatch” on page 140,and “DisSubmitBatch” on page 167.)

Each function in the batch can return errors due to local conditions (such as a missing local file in a DisPutObject call). In this situation, you can use the DisTermBatch command to terminate the batch instead of submitting it to the ODAS server.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

```
Declare Function DisTermBatch% Lib 'metrapiw.dll' (ByVal fFlags%,
ByVal hdis&)
```

where:

fFlags%	must be 0 (zero).
hdis&	is the desktop-session handle, as returned by the DisConnect function. For the default session, specify 0 (zero).

Example

This example enables, creates, and either submits or terminates a small batch request, depending on the results of DisInitBatch.

```
Dim batchResults() As Integer Dim retCode As Integer .retCode =
DisInitBatch(0, sessionHandle) . retCode = DisGetObject('Next
Report', 'NEXTRPT.WKS', DIS_FILETYPE_WKS, 0, 0, DIS_NOWAIT Or
DIS_DELETE_SOURCE, sessionHandle) . retCode =
DisGetSysMessage(numMessages, disBuffer, bufferSize, DIS_NOWAIT,
sessionHandle) .If retCode = 0 Then ReDim batchResults(1 To 2)
retCode = DisSubmitBatch(batchResults, 2, DIS_NOWAIT, sessionHandle) Else
retCode = DisTermBatch(0, sessionHandle)
```

DisTermClient

This function releases all resources allocated for this ODAS application. Use it at the end of the application.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

```
Declare Function DisTermClient% Lib 'metrapiw.dll' (ByVal fFlags%)
```

where:

fFlags%	is reserved for future use; specify 0 (zero).

Example

This example releases API resources:

```
Dim retCode As Integer
. retCode = DisTermClient( )
```

DisVBArrayToBuffer

This function copies data from an array of variables of some DIS-defined type into an array created by DisCreateBuffer and suitable for use by another ODAS API function.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisVBArrayToBuffer% Lib 'disvb.dll' (ByVal Dest&, ByVal DestLen%, Src As Any, ByVal SrcLen%, ByVal FormatString$)

where:

Dest&	is a pointer to an array created by DisCreateBuffer and to be read by a regular ODAS API function.
DestLen%	is the length, in bytes, of each element of the array in Dest&, as returned by DisCreateBuffer. DestLen% verifies that FormatString$ and Dest& match.
Src	is an array that Visual Basic can read. Src must be declared to be one of the defined types (such as DISOBJECTINFO) listed in Appendix C, "Visual Basic API Type Definitions" on page 192."
SrcLen%	is the length, in bytes, of each element of the array in Src, as obtained through the function Len(Src(1)). SrcLen% verifies that FormatString$ and Src match.
FormatString$	is a string, as listed in Table 32 on page 117, to specify the format of Src and Dest&.

Remarks

This function returns the error codes listed in Table 33 on page 118 through Table 35 on page 118. Returning DISVB_OK indicates that it successfully copied the data.

This function is identical to DisVBTypeToBuffer, except that it copies data between arrays instead of individual variables.

Example

This example sets the current @-variables of a specified Capsule application, then returns without waiting for completion.

The example assumes that the associated form contains three text boxes named txtCapsule, txtDescription, and txtValue, where the user can type the name of the Capsule application and the description and value of a selected @-variable. It also assumes that a grid control is established with three columns for the @-variables' names, descriptions, and values, as shown in Figure 7 on page 161.

Dim retCode As Integer Dim bufferSize As Integer Dim numVars As Integer Dim disBuffer As Long Dim vbBuffer() As DISCAPSULEVARINFO **.**<Visual Basic code to set up forDisSetCapsuleVars> ReDim vbBuffer(1 To numVars) **.**disBuffer = DisCreateBuffer(DISCAPSULEVARINFO_FORMAT, numVars, bufferSize) retCode = DisVBArrayToBuffer(disBuffer, bufferSize, vbBuffer(1), vbBuffer(1), DISCAPSULEVARINFO_FORMAT, numVars) retCode = DisSetCapsuleVars(**.**, disBuffer, **.**) retCode = DisFreeBuffer(disBuffer)

DisVBFormatSize

This function obtains the size of a DIS-defined type, described by a format string.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

```
Declare Function DisVBFormatSize% Lib 'disvb.dll' (ByVal
FormatString$, ByVal Flag%)
```

where:

FormatString$	is a string, as listed in Table 32 on page 117,to specify the fields layout of a Visual Basic type defined by the ODAS API.
Flag%	specifies which size information to return. Table 36 on page 124 lists the valid values.

Table 46. Information selection codes for DisVBFormatSize ()

Specifier	Value	Description
DISVB_TOTAL_SIZE	0	Gets the total size of the structure, including variable-length strings.
DISVB_FIXED_LEN_SIZE	1	Gets the fixed-length size, which does not include variable-length strings.

Remarks

If this function returns a negative value, it is one of the error codes listed in Table 33 on page 118 through Table 35 on page 118.If it returns a positive value, that value is the requested size, in bytes.

Example

This example returns the size of a structure named lanComm.

```
Dim disBuffer As Long .disBuffer =
DisVBFormatSize(DISLANCOMM_FORMAT, DISUB_FIXED_LEN_SIZE)
```

DisVBTypeToBuffer

This function copies data from a variable of some IDS-defined type into a variable created by DisCreateBuffer and suitable for use by another ODAS API function.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisVBTypeToBuffer% Lib 'disvb.dll' (ByVal
Dest&, ByVal DestLen%, Src As Any, ByVal SrcLen%, ByVal FormatString$)

where:

Dest&	is a variable created by DisCreateBuffer to be read by a regular ODAS API function.
DestLen%	is the length, in bytes, of the variable in Dest&, as returned by DisCreateBuffer or obtained through the function Len(Dest). DestLen% verifies that FormatString$ and Dest& match.
Src	is a variable that Visual Basic can read. Src must be declared to be one of the defined types (such as DISOBJECTINFO) listed in Appendix C, “Visual Basic API Type Definitions” on page 192.”
SrcLen%	is the length, in bytes, of the Src variable, as obtained through the function Len(Src). SrcLen% verifies that FormatString$ and Src match.
FormatString$	is a string, as listed in Table 32 on page 117, to specify the format of Src and Dest&.

Remarks

This function returns the error codes listed in Table 33 on page 118 through Table 35 on page 118. Returning DISVB_OK indicates that it successfully copied the data.

This function is identical to DisVBArrayToBuffer, except that it copies data between individual variables instead of arrays.

Example

This example sends Capsule variables from Visual Basic to the ODAS service.

Dim disBuffer As Long Dim vbBuffer As DISOBJECTINFO .<Visual
Basic code to operate on vbBuffer data > disBuffer =
DisCreateBuffer(DISCAPSULEVARINFO_FORMAT, 1, bufferSize) retCode =
DisVBTypeToBuffer(disBuffer, bufferSize, vbBuffer, Len(vbBuffer),
DISCAPSULEVARINFO_FORMAT) retCode = DisSetCapsuleVars('Reports
Capsule', disBuffer, numVars, DIS_NOWAIT, sessionHandle) retCode =
DisFreeBuffer(disBuffer)

DisWaitCapsuleStatus

This function pauses the personal-computer application until the specified Capsule application achieves the specified status.

Desktop Equivalent

There is no equivalent action for a user at a IDS workstation.

Function Declaration

Declare Function DisWaitCapsuleStatus% Lib 'metrapiw.dll' (ByVal idJob&, ByVal uStatus%, ByVal ulTimeout&, plCompCode&, ByVal fFlags%, ByVal hdis&)

where:

idJob&	is the identifying number of the Capsule application whose status is to be checked. The DisSubmitCapsule function provides this number.
uStatus%	specifies a numerical code representing the status to be achieved by the specified Capsule application. Currently, the only valid value is DIS_WCS_OUTPUT, or 1, which waits until the Capsule job is completed or aborted.
ulTimeout&	specifies the maximum number of seconds to wait for the Capsule application to achieve the specified status. Specify –1 to wait forever.
plCompCode&	is a variable that receives the Capsule application's completion code (after that application has reached the output queue).
fFlags%	specifies optional flags, as described in “Common Bit Flags” on page 14.
hdis&	is the desktop-session handle, as returned by the DisConnect function.

Example

This example pauses execution of the Visual Basic application until the submitted Capsule job is completed or aborted:

```
Dim retCode As Integer Dim jobNumber As Integer .retCode =
DisSubmitCapsule ('My Capsule', , jobNumber, )
' DisWaitCapsuleStatus needs the job number that ' DisSubmitCapsule
sets. retCode = DisWaitCapsuleStatus(jobNumber, DIS_WCS_OUTPUT,
–1, completion, DIS_WAIT, sessionHandle)
```

Chapter 7. Custom Visual Basic Controls for ODAS

The Intelligent Decision Server Custom Visual Basic Controls are application controls that you can use to rapidly develop custom applications on the Windows 3.1 platform. The Custom Visual Basic Controls are designed to work with the Open Data Access Service (ODAS) component of IDS.

The following controls are provided:

Table 47. Custom Visual Basic Controls

Control name	Purpose
Browser.vbx	Utility for examing data in a table
DisCom.vbx	Control for managing an application connection and logon state
ODisCom.vbx	Same as DisCom.vbx with support for Automated File Distribution
DisConv.vbx	Provides control over conversion of files transferred between the IDS desktop and the ODAS client
DisDir.vbx	List box that displays the contents of an IDS container
DisSetVs.vbx	Control that manages @-variables in capsule applications.
DisSubCp.vbx	Provides control over the capsule service used
DisTxFer.vbx	Provides capability to transfer multiple files between the IDS desktop and an ODAS client
ODisTxFr.vbx	Same as DisTxFer.vbx with support for cancellation and monitoring functions
DisXfr.vbx	Similar to DISTxFer.vbx with container copy function, name preservation, schedule function, and conversion control
ODisXfr.vbx	Same as DisXfr.vbx with cancellation and monitoring functions

Hardware and Software Requirements

The hardware requirements for using these controls are dependent upon the application that you are developing.

You must have the necessary software required to compile and run Visual Basic applications for Windows 3.1. You must also have access to an IDS LAN running the Open Data Access Service (ODAS). See *Installing IDS Lan Components* for information on installing ODAS. See *Developing Applications with Open Data Access Service* for general information about ODAS.

Installing the Controls

The Custom Visual Basic Controls are installed seperately from the IDS base installation. The controls are shipped in a compressed format and must be expanded before use. Each control is compressed with the control name as the file name and vb_ as the

© Copyright IBM Corp. 1996

extension. Depending upon where you want to install these controls, it might be necessary for you to create directories before installation.

To install the controls on your Windows 3.1 developement workstation:

1. Insert the IDS Installation CD-ROM.
2. Open an MS-DOS prompt window.
3. At the prompt, create a directory on your hard disk called odasvbx for storing the controls.
4. Change to the odasvbx directory.
5. Type copy*x*:\odasclnt\win16\visbasic\vbx\ .vb_

   where

   x is the drive letter of the CD-ROM drive.

   The compressed files will be copied to the odasvbx directory. Use the Microsoft File Expansion Utiltity** to expand the controls that you want to use.
6. Choose which control you want to expand from Table 47 on page 178.
7. Type EXPAND.
8. At the Compressed file prompt, type the name of the control that you want to expand using the extension vb_.
9. At the Expanded file prompt, type the control name using the extention vbx, specifying directories where appropriate.

   The control will expand in the specified directory.

The Custom Visual Basic Controls you expanded are ready for use in your applications.

Adding a Control to Your Application

To use a control with your application:

1. From the directory where you installed the control you want to use, copy the control, listed in Table 47 on page 178 to your application project directory.
2. Apply the control to your design form as described in the Visual Basic documentation.

About the Controls

The following sections provided a general overview of each control. For information about error codes, limitations, and property details, see Appendix E, “Custom Visual Basic Controls Reference” on page 206

The Browser

Browser is a utility for examining data in a table. It:

- Organizes columns of data in a table
- Displays the data
- Selects data by constraining columns
- Displays only the items that meet the criteria you specify

Constraining means you specify the type and range of data displayed in a Browser window. For example, suppose you have a list of your company's products, their brand descriptions, and their identification numbers; you want to know the number of a specific product that belongs to the Acme brand, but you cannot remember the product's exact package size. With Browser, you constrain the Product Description column of the database to Acme, so the window displays only the data associated with Acme. Now, you can review the list of package sizes until you find the one you want. If the displayed list is still too long (perhaps Acme is also available in many colors), you can apply further constraints until you can identify the information you want.

The Browser window supports scrolling, both horizontally and vertically.

Figure 8 on page 181 shows the list box that opens when the user clicks on this icon in your application.

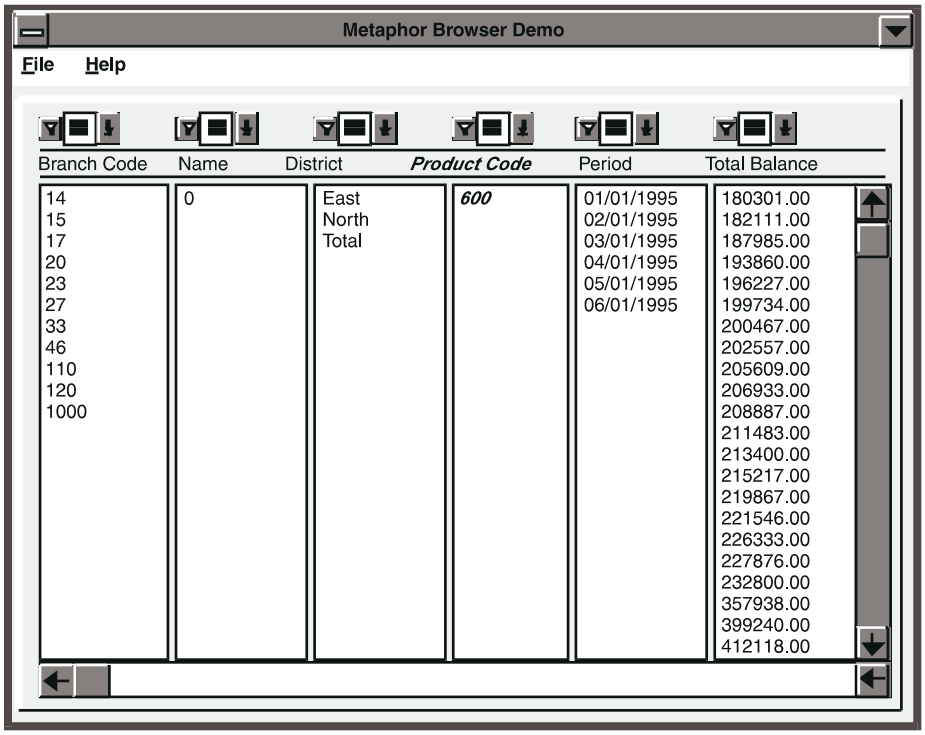


Figure 8. Example of a window generated by the Browser control

For more information, see “Browser” on page 206.

DisCapQ

DisCapQ is a list box that displays and controls the status of a submitted capsule application. It lets your application:

Cancel a capsule job that is running

Remove a capsule job, scheduled for periodic execution, from the queue

Put a capsule job on hold or release it

Change a capsule job's priority, execution time, maximum delay time, or maximum execution time, if the capsule job is not currently running

Display the completion status of a capsule job

Figure 9 on page 182 shows the list box that opens when the user clicks on the icon for this control in the application.

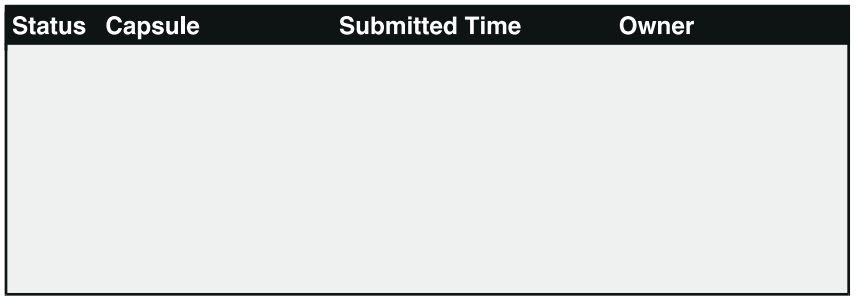


Figure 9. List box generated by the DisCapQ control

The list box that DisCapQ creates supports scrolling, both horizontally and vertically.

For more information, see “DisCapQ” on page 228.

DisCom and ODisCom

The DisCom control manages the application's connection and logon state, giving your application these basic communication capabilities:

- Connection to the ODAS server, whether by modem or a direct LAN connection
- Convenient logging on to and off of an IDS desktop
- Disconnection from the ODAS server

When the user wants to log on to an IDS desktop, DisCom automatically establishes communication between the server and the client before acting on the logon request.

The ODisCom control is identical to DisCom, except that it also supports automated file distribution (AFD). This feature lets your application download your software updates automatically or upon user demand.

For more information, see “DisCom and ODisCom” on page 237.

DisConv

DisConv gives you control of the conversion of files transferred between the IDS desktop and the client computer. It lets your application:

- Create a new conversion table with no entries
- Add or delete entries in an existing conversion table
- Reset a conversion table to the default values
- Choose one of several conversion tables for current use
- Read or write a conversion table as a local file

For more information, see “DisConv” on page 248.

DisDir

DisDir is a list box that displays the contents of an IDS container, such as a folder, file drawer, or Capsule application. This list box gives users access to IDS objects in a limited list of containers. When the user double-clicks on the name of a container, the container toggles between open and closed states.

To maximize performance, DisDir caches the information about the container and its contents.

The DisDir list box supports scrolling, both horizontally and vertically.

Figure 10 shows the list box that opens when the user clicks on this icon in your application.

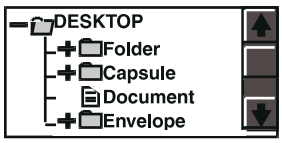


Figure 10. List box generated by the DisDir control

For more information, see “DisDir” on page 252.

DisSetVs

The DisSetVs control manages @-variables in Capsule applications. It lets your ODAS application:

Query or set the value and description of any @-variable. When using the DisSetVs function, the data type is always text.

Delete any @-variable by setting its value and description to null.

For more information, see “DisCapQ” on page 228.

DisSubCp

The DisSubCp is a control that lets your application submit capsule applications to a specified capsule service.

For more information, see “DisSubCp” on page 268.

DisTxfer and ODisTxfr

DisTxfer is a control that lets your application transfer one or more files between the computer and the IDS desktop. It accepts wildcards (asterisks) when files are transferred from the client computer to the IDS desktop, or from IDS to the client.

The ODisTxfr control is identical to DisTxfer, except that it also supports cancellation and monitoring. While ODisTxfr is transferring files, your application:

Receives events specifying the progress of the transfer

Can end the transfer at any time

For more information, see “DisTxfer and ODisTxfr” on page 273.

DisXfr and ODisXfr

DisXfr is an invisible control that lets your application transfer one or more files between the client computer and the IDS desktop. It is considerably more flexible and powerful than“DisTxfer and ODisTxfr” on page 273 in these ways:

DisXfr can copy an IDS container and all of its contents, including the contents of other containers that are nested within the specified container. DisXfr requires an existing target directory for the specified IDS container, but does not require existing directories for nested containers.

DisXfr preserves the IDS names of all copied objects. Where DisTxfer requires names compatible with the MS-DOS naming restrictions, DisXfr creates compatible names based on the IBM internal file-identification numbers. It stores these names with the corresponding IDS names and creation dates in an object information file. TheDisDircontrol then uses that information to display the original IDS names of transferred files and containers.

DisXfr can schedule a transfer to occur on or after a specified date.

DisXfr uses the conversion table that is defined through the DisConv control. Before copying any files or other objects, DisXfr uses the setting to set the default conversion table on the ODAS server.

Like DisTxfer, the DisXfr control accepts wildcards (asterisks) when transferring from the client computer to the IDS desktop, or from IDS to the client.

The ODisXfr control is identical to DisXfr, except that it also supports cancellation and monitoring. While ODisXfr is transferring files, your application:

Receives events specifying the progress of the transfer

Can end the transfer at any time

For more information, see “DisXfr and ODisXfr” on page 281.

Appendix A. C API Type Definitions

This appendix shows the typedef statements for the types used in the C API function parameters.

```
typedef       unsigned  long     ULONG;
typedef       unsigned  int      UINT;
typedef       unsigned  char     BYTE;
typedef       unsigned  short    USHORT;
typedef        unsigned long    PULONG; /  pointer types  /
typedef        unsigned int     PUINT;
typedef        unsigned char    CHAR FAR  ;
typedef       char                PCHAR;
typedef       void                PVOID;

typedef        ULONG                HDIS    /  INT DISENTRY session handle  /
typedef        HDIS                PHDIS /  pointer to INT DISENTRY session handle  /

typedef struct    _DISCAPSULEJOBINFO_L1 {
      USHORT cb;              /  Size of this entry, in bytes  /
      BYTE bPriority;      /  Job priority; specify 1 for the lowest priority to
                                        31 for the highest priority  /
      BYTE bState;            /  Job state  /
      ULONG id;              /  Job identifier  /
      CHAR FAR  pszOwner;      /  Desktop that submitted the job  /
      CHAR FAR  pszDescription    /  Job's description  /
      CHAR FAR  pszAliasName    /  Scheduler alias name  /
      ULONG timeSubmitted;      /  Time when job was submitted  /
      ULONG timeScheduled;    /  Job's scheduled starting time  /
      ULONG secsMaxStartDelay;    /  Maximum delay past scheduled time,
                                          in seconds  /
      ULONG secsMaxExecution; /  Maximum execution time, in seconds  /
} DISCAPSULEJOBINFO;

typedef struct    _DISCAPSULEJOBPARMS {
     USHORT  cb;
      BYTE bPriority;       /  Job priority; specify 1 for the lowest priority
                                   to 31 for the highest priority  /
      BYTE fbFlags;         /  Bit mask relating to when the job is to be run  /
      ULONG flMask;            /  Bit mask telling which parameters are set in
                                     various structure members  /
      CHAR FAR  pszDescription;    /  Job description  /
      CHAR FAR  pszAliasName;      /  Scheduler's alias name  /
      ULONG timeScheduled;         /  Scheduled start time  /
      ULONG secsMaxStartDelay;      /  Maximum delay past scheduled time,in seconds  /
      ULONG secsMaxExecution;    /  Maximum execution time, in seconds  /
} DISCAPSULEJOBPARMS;
```

© Copyright IBM Corp. 1996

```
#define DIS_CAPSULEVAR_NAME_LEN     4
#define  DIS_CAPSULEVAR_DESC_LEN    5    /  Maximum  description  length  /
#define  DIS_CAPSULEVAR_VALUE_LEN    255    /  Maximum  value  length  /
typedef struct    _DISCAPSULEVARINFO {
      CHAR szName[DIS_CAPSULEVAR_NAME_LEN] /  Variable name  /
      ULONG ulType;                                  / Variable's Type  /
      CHAR FAR  pszDescription; /  Variable's description  /
      CHAR FAR  pszValue;                /  Variable's value  /
} DISCAPSULEVARINFO;

typedef struct    _DISCONVERTENTRY {
      CHAR FAR  pszFileExt;          /  File type on the personal computer (characters
                                          after the last period in a file identifier)  /
      BYTE bFileType;                  /  Personal-computer file type  /
      BYTE bObjType;                    /  Desktop-object type  /
      USHORT cchLineWidth;         /  Maximum line width when bObjType is 1
                                                                    (DIS_FILETYPE_TEXT)   /
} DISCONVERTENTRY;

typedef struct    _DISERRORINFO {
      UINTcb;                          /  Size of structure(must be set on entry)  /
      INTerr;                         /  Primary error code  /
      INTerrSystem;                /  System error code  /
      UINTuErrLoc;                 /  Component in which error occurred  /
      UINTuErrType;                /  Type of error  /
      INTerrBaseSys;              /  Untranslated, low-level error code  /
} DISERRORINFO,   PDISERRORINFO;

typedef struct    _DISGETOBJECTINFO {
      USHORT cb;                     /  Size of this entry, in bytes  /
      BYTE bObjType;               /  Numerical code indicating the original type of
                                      desktop  object
      BYTE bFileType;             /  Numerical code indicating the destination file /
      ULONG cbFile;                          /  Size of destination file, in bytes  /
      CHAR szName [DIS_OBJECT_NAME_LEN];     /  Original desktop-object
                                                                    name   /
} DISGETOBJECTINFO;

typedef struct    _DISLANCOMM {
      USHORT cb;                                 /  Size of the structure, in bytes  /
      USHORT usTimeout;              /  Connection timeout; how many seconds to wait
                                                                    for a completed connection  /
      CHAR FAR  pszServerName;     /  Server's name  /
} DISLANCOMM;

typedef struct    _DISMSGCOUNT {
      USHORT cSysMessages;       /  Number of pending system messages  /
      USHORT cMail;                            /  Number of items in the mail tray  /
      USHORT cNewMail;                   /  New items in mail tray since the prior query  /
} DISMSGCOUNT;
```

```
#define DIS_OBJECT_NAME_LEN       8      / Maximum length of a desktop
                                                                    object's name  /
typedef struct    _DISOBJECTINFO {
      USHORT cb;                                  / Size of this entry, in bytes  /
      BYTE bObjType;                            / Numerical code indicating the type of desktop
                                                                    object   /
      BYTE fbFlags;                             / Reserved for future use; specify   (zero)  /
      ULONG id;                                   / Unique identifier for each desktop object, to
                                                      distinguish between objects with identical
                                                      names   /
      ULONG timeCreation;               / Time and date when the object was created /
      ULONG timeLastRead;               / Time and date when it was last read  /
      ULONG timeLastWrite;                / Time and date when it was last written  /
      ULONG cbObject;                         / Size of the object, in bytes  /
      CHAR szName[DIS_OBJECT_NAME_LEN];      / Object's name  /
} DISOBJECTINFO;

#define   DIS_SERVICENAME_LEN         64
typedef struct    _DISSERVICEINFO {
     ULONG idSession;                         / Current session identifier  /
   CHAR szServiceName [DIS_SERVICENAME_LEN];      / Servername  /
   CHAR szServiceName [DIS_SERVICENAME_LEN];      / Network name /
} DISSERVICEINFO,   PDISSERVICEINFO;

#define  DIS_SYSMESSAGE_LEN    256    /  Maximum  message  length   /
typedef struct    _DISSYSMESSAGE {
      USHORT cb;                                  / Size of structure, in bytes  /
      USHORT usReserved;                / Reserved for future use; specify   (zero)  /
      ULONG time;                                 / Time message was received  /
      CHAR szMessage[DIS_SYSMESSAGE_LEN];      / Message text  /
} DISSYSMESSAGE;
```

Appendix B. Program Code with C-Language API Functions

This appendix provides an annotated sample application program.

This application uses only C-language API functions.

```
#include <stdio.h>
#include <memory.h>
#include <stdlib.h>

#define INCL_DIS_ERRORS
#include <disapi.h>


/ Define a few datatypes. /

typedef BYTE     PBYTE;      / byte-pointer /
typedef VOID     PVOID;      / void-pointer /
typedef CHAR     PSZ;        / string-pointer /
typedef UINT     BOOL;       / boolean /


/ GAOK: Returns TRUE if no error occurs; otherwise,
     displays the error code and returns FALSE. /

static BOOL GAOK (INT error, PSZ pszPgmName)
{
     if (error) {
          printf ('%s() -> MET% 4u\n', pszPgmName,
                    (error - ERROR_MET_BASE));
     }
     return (error ==  );
}


#define BUFSIZE        x1             / buffer size /
#define BUFCOUNT       1                / item count /



/ ListMail: Lists the current contents of the Mail Tray /

static void ListMail (HDIS sessionHandle)
{
    INT  error;
    PVOID  pBuffer;
    UINT  iStart;
    UINT  cEntries;
    UINT  cbBuf;
    PDISOBJECTINFO  pobji;
    UINT  i;
```


© Copyright IBM Corp. 1996

```
BOOL fHeader = TRUE;

pBuffer = malloc (BUFSIZE);           /  Allocate a buffer.  /
iStart =  ;                  /  Initialize the starting index.  /
for (;;) {
    cEntries = BUFCOUNT;                /  Ask for many items.  /
    cbBuf = BUFSIZE;         /  Initialize the buffer size.  /

    /  Match all items.  /
    error = DisQueryObjectInfo ('Mail Tray\\ ',
            DIS_QOIL_STANDARD,      /  Return standard info.  /
            &iStart,                          /  starting index  /
            &cEntries,                        /  item count  /
            pBuffer,                          /  buffer pointer  /
            &cbBuf,                             /  buffer size  /
            DIS_QOI_COUNT_CONTENTS,    /  Count enclosures.  /
            sessionHandle);                 /  session handle  /



    /  Check for error.
        BUFFER_TOO_SMALL is 'normal' (meaning there are more
        entries to retrieve).
        FILE_NOT_FOUND means there are no more entries —
        treated as an error only if it occurs the first time
        through the loop.  /

    if (error != ERROR_MET_BUFFER_TOO_SMALL) {
        if ((error == ERROR_MET_FILE_NOT_FOUND) &&
                !fHeader)  {
            break;
        }else if (!GAOK (error, 'DisQueryObjectInfo')) {
            break;
        }
    }




    /  Display column headers first time through loop.  /

    if (fHeader) {
        puts  ('Encl    Name');
        fHeader = FALSE;
    }




    /  Loop through the returned items displaying the
```

```
            number of enclosures and names.  /

        pobji = (PDISOBJECTINFO)pBuffer;
        for (i =  ; i < cEntries; i++) {
            printf  ('%3lu       %s\n',  pobji->cbObject,
                    pobji->szName);
            pobji = PNEXTDISOBJECTINFO (pobji);
        }
        /  Resetting starting index means no more entries.  /
        if (iStart ==  ) {
            break;
        }
    }
    free (pBuffer);             /  Free the buffer  /
}


INT main (INT argc, PSZ argv[])
{
    INT  error;
    DISASYNCCOMM  async;
    HDIS  sessionHandle;
     PSZ pszDialString, pszDesktop, pszPassword;

     if (argc != 4) {
          puts ('Usage: gapcxmp <phone> <desktop> <password>');
         return  4;
     }
     pszDialString = argv[1];
     pszDesktop = argv[2];
     pszPassword = argv[3];

     error = DisInitClient ( );
     if (GAOK (error, 'DisInitClient')) {


         /  Initialize the communications parameters and set the
              dialing string.  /

         memset (&async,  , sizeof (async));
         async.cb = sizeof (async);
         async.pszDial = pszDialString;




         /  Try to connect to the server.  /

         error = DisConnect (DIS_COMM_ASYNC, &async,  , &sessionHandle);
```

```
        if (GAOK (error, 'DisConnect')) {


            /  Access the specified desktop.  /

            error = DisLogon (pszDesktop, pszPassword,  , sessionHandle);
            if (GAOK (error, 'DisLogon')) {


                /  List the Mail Tray.  /

               ListMail  (sessionHandle);
            }

            error = DisLogoff ( , sessionHandle);
            GAOK (error, 'DisLogoff');

            /  End the session.  /

            error = DisDisconnect ( , sessionHandle);
            GAOK (error, 'DisDisconnect');
        }
        DisTermClient  ( );
    }
    return error ? 16 :  ;
}
```

Appendix C. Visual Basic API Type Definitions

This appendix shows the *Global Const*statements for the types used in the Visual Basic API function parameters.

With the sole exception of *DISDATEANDTIME*, the defined data types listed here correspond to the structures defined for the C-language API.

```
Type DISCAPSULEJOBINFO_L1
     cb As Integer                    ' Size of this entry, in bytes
     bPriority As Integer             ' Job priority; specify 1 for the lowest priority to
                                           31 for the highest priority
     bState As Integer                ' Job state
     id As Long                       ' Job identifier
     pszOwner As String               ' Desktop that submitted the job
     pszDescription As String         ' Job's description
     pszAliasName As String           ' Scheduler alias name
     timeSubmitted As Long            ' Time when job was submitted
     timeScheduled As Long            ' Job's scheduled starting time
     secsMaxStartDelay As Long        ' Maximum delay past scheduled time,
                                           in  seconds
     secsMaxExecution As Long         ' Maximum execution time, in seconds
End Type
Global Const DISCAPSULEJOBINFO_L1_FORMAT$ = 'ICCLP8 P8 P8 LLLL'

Type DISCAPSULEJOBPARMS
     cb As Integer
     bPriority As Integer             ' Job priority; specify 1 for the lowest priority
                                           to 31 for the highest priority
     fbFlags As Integer               ' Bit mask relating to when the job is to be
                                           executed
     flMask As Long                   ' Bit mask telling which parameters are set in
                                           various structure members
     pszDescription As String         ' Job description
     pszAliasName As String           ' Scheduler's alias name
     timeScheduled As Long            ' Scheduled start time
     secsMaxStartDelay As Long        ' Maximum delay past scheduled time,
                                           in  seconds
     secsMaxExecution As Long         ' Maximum execution time, in seconds
End Type
Global Const DISCAPSULEJOBPARMS_FORMAT$ = 'ICCLP5 P15LLL'

Global Const DIS_CAPSULEVAR_NAME_LEN = 4
Global Const DIS_CAPSULEVAR_DESC_LEN = 5  ' Maximum description length
Global Const DIS_CAPSULEVAR_VALUE_LEN = 255 ' Maximum value length
Type DISCAPSULEVARINFO
     szName As String   DIS_CAPSULEVAR_NAME_LEN ' Variable name
     ulType As Long                   'Variable's Type
     pszDescription As String         ' Variable's description
     pszValue As String               ' Variable's value
End Type
Global Const DISCAPSULEVARINFO_FORMAT$ = 'S4LP5 P255'
```


© Copyright IBM Corp. 1996

```
Type DISCONVERTENTRY
      pszFileExt As String              ' File type on the personal computer (characters
                                              after the last period in a file identifier)
      bFileType As Integer              ' Personal-computer file type
      bObjType As Integer               ' Desktop-object type
      cchLineWidth As Integer           ' Maximum line width when bObjType
                                              is  DIS_FILETYPE_TEXT
End Type
Global Const DISCONVERTENTRY_FORMAT$ = 'P1 CCI'

Type DISDATEANDTIME sec As Integer ' must be   through 59
      min As Integer                    ' must be   through 59
      hour As Integer                   ' must be   through 23
      day As Integer                    ' must be 1 through 7
      month As Integer                  ' must be 1 through 12
      year As Integer                   ' must be   through 99
End Type

Type DISERRORINFO
      cb As Integer                     ' Size of structure(must be set on entry)
      err As Integer                    ' Primary error code
      errSystem As Integer              ' System error code
      uErrLoc As Integer                ' Component in which error occurred
      uErrType As Integer               ' Type of error
      errBaseSys As Integer             ' Untranslated, low-level error code
End Type

Global Const DIS_OBJECT_NAME_LEN = 8
Type DISGETOBJECTINFO {
      cb As Integer                     ' Size of this entry, in bytes
      bObjType As Integer               ' Numerical code indicating the original type of
                                              desktop  object
      bFileType As Integer              ' Numerical code indicating the destination file
                                              type
      cbFile As Long                    ' Size of destination file, in bytes
      szName As String                  ' Original desktop-object name (maximum length
                                              is  DIS_OBJECT_NAME_LEN)
End Type
Global Const DISGETOBJECTINFO_FORMAT$ = 'ICCLZ8 '

Type DISLANCOMM
      cb As Integer                     ' Size of the structure, in bytes
      usTimeout As Integer              ' Connection timeout; how many seconds to wait
                                              for a completed connection
      pszServerName As String           ' Server's name
End Type
Global Const DISLANCOMM_FORMAT$ = 'IIP8 '

Type DISMSGCOUNT
      cSysMessages As Integer           ' Number of pending system messages
      cMail As Integer                  ' Number of items in the mail tray
      cNewMail As Integer               ' New items in mail tray since the prior query
End Type
```

```
Global Const DIS_OBJECT_NAME_LEN    8       ' Maximum length of a desktop
                                              object's  name
Type DISOBJECTINFO
      cb As Integer                          ' Size of this entry, in bytes
      bObjType As Integer                    ' Numerical code indicating the type of desktop
                                                 object
      fbFlags As Integer                    ' Reserved for future use; specify   (zero)
      id As Long                             ' Unique identifier for each desktop object, to
                                                 distinguish between objects with identical
                                                 names
      timeCreation As Long                  ' Time and date when the object was created
      timeLastRead As Long                   ' Time and date when it was last read
      timeLastWrite As Long                 ' Time and date when it was last written
      cbObject As Long                       ' Size of the object, in bytes
      szName As String                       ' Object's name (maximum length
                                                 is  DIS_OBJECT_NAME_LEN)
End Type
Global Const DISOBJECTINFO_FORMAT$ = 'ICCLLLLLZ8 '

Global Const DIS_SERVICENAME_LEN 64          ' Maximum service-name length
Type DISSERVICEINFO
      idSession As Long              ' Current-session identifier
      szServiceName As String   DIS_SERVICENAME_LEN      'Service name
      szNetworkName As String   DIS_SERVICENAME_LEN      'Network name
End Type

Global Const DIS_SYSMESSAGE_LEN      256      ' Maximum message length
Type DISSYSMESSAGE
      cb As Integer              ' Size of structure, in bytes
      time As Long                ' Time message was received
      szMessage As String ' Message text (maximum length
                                    is  DIS_SYSMESSAGE_LEN)
End Type
Global Const DISSYSMESSAGE_FORMAT$ = 'ILZ256'
```

Appendix D. Sample Program Code with Visual Basic API Functions

This appendix provides an annotated sample application program, which is included in the client installation.

This application uses only Visual Basic API functions, with no C code. The project includes four forms:

GETOBJ.FRM frmGetObj

LOGON.FRM frmLogon

SETUP.FRM frmCommunication

SUBCAP.FRM frmSubmitCapsule

CONSTANT.BAS

DISAPI.BAS

DISVB.BAS

GLOBAL.BAS

METRERR.BAS

ANIBUTON.VBX

CMDIALOG.VBX

GRAPH.VBX

MSCOMM.VBX

MSMASKED.VBX

THREED.VBX

SETUP.FRM

The file SETUP.FRM these subroutines and functions:

Sub cmdCancel_Click

Sub cmdOk_Click

Sub Form_Load

```
Begin Form frmCommunication
...
Const maxBaud = 6
Dim baudRate() As Integer

Sub cmdCancel_Click ()
 Unload Me
End Sub
```

© Copyright IBM Corp. 1996

```
 Sub cmdOk_Click ()
Dim comPort As Integer
Dim currentSpeed As Integer
Dim speed As Integer

    If Len(cmbPort.Text) =   Then
    InvalidInputMsg 'You need to specify the
     asynchronous-communication-port',  Me.cmbPort
   Exit  Sub
   End  If
    comPort = Val(cmbPort.Text)
    If comPort <   Or cmbPort > 4 Then
 InvalidInputMsg 'You have specified an invalid
   communication-port value', Me.cmbPort
 Exit Sub
  End  If
   If Len(cmbBaudRate.Text) =   Then
 InvalidInputMsg 'You need to specify the baud rate
  value',  cmbBaudRate
 Exit Sub
  End  If
   currentSpeed = Val(cmbBaudRate.Text)
   speed =
   For i = 1 To maxBaud
     If currentSpeed = baudRate(i) Then
         speed = i
        Exit  For
    End  If
 Next i
 If speed =   Then
     InvalidInputMsg 'You have specified an invalid baud
        rate value', cmbBaudRate
    Exit  Sub
 End If

 If Len(txtPhone.Text) =   Then
     InvalidInputMsg 'You need to specify the phone number
        value',  txtPhone
    Exit  Sub
 End If

If Len(cmbTimeOut.Text) =   Then
     InvalidInputMsg 'You need to specify the timeout
        value',  cmbTimeOut
    Exit  Sub
 End If

 asyncComm.bPort = comPort
 asyncComm.bSpeed = speed
 asyncComm.pszDial = RTrim(txtPhone.Text) & Chr$( )
 asyncComm.usTimeOut = Val(cmbTimeOut.Text)
 connectMode = DIS_COMM_ASYNC
```

```
 Unload Me
End Sub

Sub Form_Load ()
 For i = 1 To 4
       cmbPort.AddItem  LTrim(Str$(i))
 Next
 ReDim baudRate(1 To maxBaud)
 baudRate(1) = 3
 baudRate(2) = 12
 baudRate(3) = 24
 baudRate(4) = 48
 baudRate(5) = 96
 baudRate(6) = 192
 For i = 1 To maxBaud
       cmbBaudRate.AddItem  LTrim(Str$(baudRate(i)))
 Next i
 For i = 1 To 15
        cmbTimeOut.AddItem LTrim(Str$(i   3 ))
 Next
 If connectMode = DIS_COMM_ASYNC Then
        cmbPort.Text = LTrim(Str$(asyncComm.bPort))
       cmbBaudRate.Text  =
          LTrim(Str$(baudRate(asyncComm.bSpeed)))
        txtPhone.Text = LTrim(asyncComm.pszDial)
        cmbTimeOut.Text = LTrim(Str$(asyncComm.usTimeOut))
 Else
        cmbPort.Text = '1'
        cmbBaudRate.Text = LTrim(Str$(baudRate(3)))
        cmbTimeOut.Text = '6 '
 End If
End Sub
```

LOGON.FRM

The file *LOGON.FRM* includes the form illustrated below and these subroutines and functions:

- Sub cmdCommunication_Click
- Sub cmdLogon_Click
- Sub cmdQuit_Click
- Sub Form_Load

```
Begin Form frmLogon
...
Sub cmdCommunication_Click ()
   frmCommunication.Show  1
End Sub

Sub cmdLogon_Click ()
Dim retLogon As Integer
```

```
Dim cbBuf As Integer
Dim cbufAsyncComm As Long
Dim sessionHandle As Long
    If Len(txtName.Text) =   Then
        InvalidInputMsg 'You need to specify your desktop
          name',  txtName
        Exit  Sub
    End  If
    If Len(txtPassword.Text) =   Then
        InvalidInputMsg 'You need to specify your password.',
          txtPassword
        Exit  Sub
    End  If
    Screen.MousePointer = 11
    If connectMode = DIS_COMM_DEFAULT Then
        retCode = DisConnect(connectMode, ByVal  &, DIS_WAIT,
          sessionHandle)
    Else
        cbufAsyncComm = DisCreateBuffer(DISASYNCCOMM_FORMAT,
          1,  cbBuf)
        If cbufAsyncComm <=  & Then
            ErrorMsg 'Failed to create buffer.'
        Else
            asyncComm.cb = DisCFormatSize(DISASYNCCOMM_FORMAT,
              DISVB_FIXED_LEN_SIZE)
            retCode = DisVBTypeToBuffer(cbufAsyncComm, cbBuf,
                asyncComm, Len(asyncComm), DISASYNCCOMM_FORMAT)
            If retCode = DISVB_OK Then
                retCode = DisConnect(connectMode, ByVal
                    cbufAsyncComm, DIS_WAIT, sessionHandle)
            End  If
            i = DisFreeBuffer(cbufAsyncComm)
        End  If
    End  If

    If retCode =   Then
        flConnected = True
        retCode = DisLogon(RTrim(txtName.Text) & Chr$( ),
            RTrim(txtPassword.Text) & Chr$( ), DIS_WAIT,
            sessionHandle)
        Screen.MousePointer =
        If retCode <>   Then
            ErrorMsg 'Failed to logon to the desktop.'
        Else
            Unload  Me
            frmSubmitCapsule.Show  1
        End  If
    Else
        Screen.MousePointer =
        ErrorMsg 'Failed to connect to ODAS Server.'
    End  If
```

```
End Sub

Sub cmdQuit_Click ()
    If flConnected Then
        Screen.MousePointer = 11
        retCode = DisDisconnect(DIS_WAIT,  &)
        Screen.MousePointer =
   End  If
    retCode = DisTermClient( )
   Unload  Me
End Sub

Sub Form_Load ()
Dim errCode As Integer

    On Error GoTo ErrorHandler
    i = Shell('winxnode.exe')              ' Shell winxnode program
    ' We need to retry here since winxnode needs to be
       initialized
    ' before DisInitClient can be executed.
    For j = 1 To 4
        errCode = DisInitClient( )
        If errCode =   Then
            connectMode = DIS_COMM_DEFAULT
           Exit  For
       Else
            n = Int(5      Rnd + 1)
            If j Mod 1    =   Then DoEvents
       End  If
   Next  j
    If errCode <>   Then
        ErrorMsg 'Failed to initialize client resources.'
   End  If
   Exit  Sub
ErrorHandler:
    Select Case Err
       Case  53:
            ErrorMsg 'Failed to start the "winxnode.exe"
                program.      Please make sure that the "nodesys"
                directory is included in your system path.'
           Resume  Next
       Case  Else:
            Error  Err
   End  Select
End Sub
```

GETOBJ.FRM

The file *GETOBJ.FRM* includes the form illustrated below and these subroutines and functions:

Sub cmdCapsule_Click

Sub cmdGet_Click

Sub cmdQuit_Click

Sub dirList_Change

Sub drvList_Change

Sub filList_Click

Sub Form_Load

Sub txtDestinationFile_KeyPress

```
Begin Form frmGetObj
...
Sub cmdCapsule_Click ()
    Unload  Me
    frmSubmitCapsule.Show  1
End Sub

Sub cmdGet_Click ()
Dim retCode As Integer
Dim pcbFile As Long
Dim fileType As Integer
Dim fFlags As Integer

     If Len(txtSourceFile.Text) =   Then
          ErrorMsg ('You need to specify the DIS source file
              name')
         Exit  Sub
    End  If
     If Len(txtDestinationFile.Text) =   Then
          ErrorMsg ('You need to specify the destination file
              name')
         Exit  Sub
    End  If
     If Len(txtLineWidth.Text) =   Then
          ErrorMsg ('You need to specify the destination file line
              width')
         Exit  Sub
    End  If

     fileType = GetPCFileType(cmbFileType.Text)

     fFlags = DIS_WAIT
     If optDelete.Value = True Then
          fFlags = fFlags Or DIS_DELETE_SOURCE
     ElseIf optExpunge.Value = True Then
          fFlags = fFlags Or DIS_DELETE_EXPUNGE
    End  If
     If chkOverwrite.Value = True Then
          fFlags = fFlags Or DIS_OVERWRITE
    End  If
```

```
    Screen.MousePointer = 11
    retCode = DisGetObject
        (RTrim(txtSourceFile.Text) & Chr$( ),
        RTrim(txtDestinationFile.Text) & Chr$( ), fileType,
        pcbFile, Val(txtLineWidth.Text), fFlags, hdis)
    Screen.MousePointer =
    If retCode Then
        ErrorMsg ('Error retrieving DIS file to local machine')
   End  If


End Sub

Sub cmdQuit_Click ()
    Screen.MousePointer = 11
    retCode = DisDisconnect(DIS_WAIT,  &)
    Screen.MousePointer =
    retCode = DisTermClient( )
   Unload  Me
End Sub

Sub dirList_Change ()
    filList.Path = dirList.Path
End Sub

Sub drvList_Change ()
    dirList.Path = drvList.Drive
End Sub

Sub filList_Click ()
    txtDestinationFile.Text = filList.FileName
End Sub

Sub Form_Load ()
   SetPCFileType  Me.cmbFileType
    txtDestinationFile.Text = filList.Pattern

End Sub

Sub txtDestinationFile_KeyPress (KeyAscii As Integer)
    On Error GoTo ErrorHandler1
    If KeyAscii = 13 Then
        filList.FileName = txtDestinationFile.Text
   End  If
   Exit  Sub

ErrorHandler1:
    Select Case Err
        Case  53:
            ErrorMsg 'You have specified an invalid file name'
        Case  Else:
            Error  Err
```

```
    End  Select
    Resume  Next
End Sub
```

SUBCAP.FRM

The file *SUBCAP.FRM* includes the form illustrated below and these subroutines and functions:

Sub cmdExecute_Click

Sub cmdGetResult_Click

Sub cmdQuit_Click

Sub Form_Load

Function MakeTime

```
Begin Form frmSubmitCapsule
...
Sub cmdExecute_Click ()
Dim pBuf As Long
Dim cbBuf As Integer
Dim jobParm As DISCAPSULEJOBPARMS
Dim retCode As Integer
Dim idjob As Long

    If Len(txtCapsule.Text) =   Then
        InvalidInputMsg 'You need to specify the capsule
            name',  txtCapsule
        Exit  Sub
    End  If
    If Len(txtContainer.Text) =   Then
        InvalidInputMsg 'You need to specify the destination
            container',  txtContainer
        Exit  Sub
    End  If
    jobParm.cb = DisCFormatSize(DISCAPSULEJOBPARMS_FORMAT,
        DISVB_TOTAL_SIZE)
    jobParm.bPriority = 31
    jobParm.fbFlags =
    jobParm.pszDescription = txtCapsule.Text & Chr$( )
    jobParm.timeScheduled = MakeTime(txtSubmitTime)
    If jobParm.timeScheduled = -1 Then   Input error
        Exit  Sub
    End  If
    jobParm.pszAliasName = Chr$( )
    jobParm.flMask = DIS_CJP_PRIORITY Or DIS_CJP_DESCRIPTION
        Or  DIS_CJP_SCHEDULED

    pBuf = DisCreateBuffer(DISCAPSULEJOBPARMS_FORMAT, 1,
        cbBuf)
    If pBuf <=  & Then
```

```
        ErrorMsg ('Failed to create buffer')
    Else
        retCode = DisVBTypeToBuffer(pBuf, cbBuf, jobParm,
            Len(jobParm),  DISCAPSULEJOBPARMS_FORMAT)
        If retCode =   Then
            Screen.MousePointer = 11
            retCode = DisSubmitCapsule
                (txtCapsule.Text & Chr$( ),
                txtContainer.Text & Chr$( ), pBuf, idjob,
               DIS_WAIT,  hdis)
            Screen.MousePointer =
            If retCode Then
                ErrorMsg ('Failed to submit capsule')
            Else
                lblJobIdValue.Caption = Str(idjob)
            End  If
        End  If
        retCode = DisFreeBuffer(pBuf)
    End  If
End Sub

Sub cmdGetResult_Click ()
    Unload  Me
    frmGetObj.Show  1
End Sub

Sub cmdQuit_Click ()
    Screen.MousePointer = 11
    retCode = DisDisconnect(DIS_WAIT,  &)
    Screen.MousePointer =
    retCode = DisTermClient( )
    Unload  Me
End Sub

Sub Form_Load ()
    txtSubmitTime.Text = Format(Now, 'mm/dd/yy hh:mm:ss')
End Sub

Function MakeTime& (txtTime As Control)
    Dim T As DATEANDTIME
    Dim dayPos As Integer
    Dim yearPos As Integer
    Dim hourPos As Integer
    Dim minPos As Integer
    Dim secPos As Integer
    Dim convTime As Long

    dayPos = InStr(1, txtTime.Text, '/')
    If dayPos =   Then
        GoTo  InvalidFormat
    End  If
    yearPos = InStr(dayPos + 1, txtTime.Text, '/')
```

```
    If yearPos =   Then
        GoTo  InvalidFormat
    End  If
    hourPos = InStr(yearPos + 1, txtTime.Text, ' ')
    If hourPos =   Then
        GoTo  InvalidFormat
    End  If
    minPos = InStr(hourPos + 1, txtTime.Text, ':')
    If minPos =   Then
        GoTo  InvalidFormat
    End  If
    secPos = InStr(minPos + 1, txtTime.Text, ':')
    If secPos =   Then
        GoTo  InvalidFormat
    End  If
    T.month = Val(Mid(txtTime.Text, 1, dayPos - 1))
    T.day = Val(Mid(txtTime.Text, dayPos + 1,
         yearPos - dayPos - 1))
    T.year = Val(Mid(txtTime.Text, yearPos + 1,
         hourPos - yearPos - 1))
    T.hour = Val(Mid(txtTime.Text, hourPos + 1,
         minPos - hourPos - 1))
    T.min = Val(Mid(txtTime.Text, minPos + 1,
         secPos - minPos - 1))
    T.sec = Val(Mid(txtTime.Text, secPos + 1))
    If T.month < 1 Or T.month > 12 Then
        GoTo  InvalidFormat
    End  If
    If T.day < 1 Or T.day > 31 Then
        GoTo  InvalidFormat
    End  If
    If T.year <   Or T.year > 99 Then
        GoTo  InvalidFormat
    End  If
    If T.hour <   Or T.hour > 23 Then
        GoTo  InvalidFormat
    End  If
    If T.min <   Or T.min > 59 Then
        GoTo  InvalidFormat
    End  If
    If T.sec <   Or T.sec > 59 Then
        GoTo  InvalidFormat
    End  If
    convTime = DisMkTime(T)
    If convTime = -1 Then
        GoTo  InvalidFormat
    Else
        MakeTime = convTime
    End  If
    Exit  Function
InvalidFormat:
    InvalidInputMsg “Please enter correct time format
```

```
        (mm/dd/yy hh:mm:ss)", txtTime
    MakeTime = -1

End Function
```

Appendix E. Custom Visual Basic Controls Reference

The following sections provide detailed information on the use of the IDS Custom Visual Basic Controls.

Browser

The following sections provide information on the browser error codes, methods, events, and properties.

Error Codes

Browser returns three types of error codes:

Codes received from the ODAS server, which are defined in the DISERR.TXT file.

Visual Basic codes.

Internal codes, as listed in the following table.

Name	Value	Description
BRWS_ERR_TooLowColCount	32501	You tried to specify a Browser control with no columns; it must have at least one.
BRWS_ERR_TooHiColCount	32502	You tried to specify a Browser control with more than 32 columns; it can have no more than 32.
BRWS_ERR_TooLowListItemTopMargin	32503	You tried to specify a negative top margin for a list item. The top margin must be at least 0.
BRWS_ERR_TooHiListItemTopMargin	32504	You tried to specify too much top margin for a list item. The top margin must be no more than 10.
BRWS_ERR_TooLowListItemLeftMargin	32505	You tried to specify a negative left margin for a list item. The left margin must be at least 0.
BRWS_ERR_TooHiListItemLeftMargin	32506	You tried to specify too much left margin for a list item. The left margin must be no more than 10.
BRWS_ERR_TooLowSpacing	32507	You tried to specify too little column spacing. The column spacing must be 1 or more.
BRWS_ERR_TooHiSpacing	32508	You tried to specify too much column spacing. The column spacing must be no more than 10.
BRWS_ERR_TooLowLeftMargin	32509	You tried to specify a negative left margin (between the left edge of the Browser window and left edge of the first list box). The left margin must be at least 0.

© Copyright IBM Corp. 1996

Name	Value	Description
BRWS_ERR_TooHiLeftMargin	32510	You tried to specify too much left margin (between the left edge of the Browser window and left edge of the first list box). The left margin must be no more than 50.
BRWS_ERR_InvalidIndex	32511	You specified an invalid index value (column number). Use 0 for the first column through*n-1* for the nth column.
BRWS_ERR_SetRunTimeOnly	32512	This property can be set during execution only.
BRWS_ERR_UnknownActionProp	32513	You tried to set the Action property to an invalid value.
BRWS_ERR_UnknownActionIndexProp	32514	You tried to set the ActionIndex property to an invalid value.
BRWS_ERR_NoFileHsz	32515	No file could be loaded, because the File_Name property is not set.
BRWS_ERR_ErrorOpenConfigFile	32516	An error occurred when the control tried to open the configuration file (the file that contains configuration information, and is created by the IDS Browser tool to be read by the Visual Basic Browser).
BRWS_ERR_ErrorOpenFile	32517	An error occurred when the control tried to open the specified file.
BRWS_ERR_BadInternalState	32518	An unexpected value has been encountered.
BRWS_ERR_UnableToAllocateMemory	32519	The control was unable to allocate memory.
BRWS_ERR_SelectBeforeConstrain	32520	The user tried to constrain a data column before selecting a data value in that column.
BRWS_ERR_OneItemConstrain	32521	The user tried to constrain a data column while two or more data values were selected, but the selected constraint operator requires exactly one value.
BRWS_ERR_TwoItemConstrain	32522	The user tried to constrain a data column while three or more data values were selected, but the selected constraint operator requires exactly two values.
BRWS_ERR_ConstrainFailed	32523	The constraint failed.
BRWS_ERR_InvalidOperator	32524	The application programmer tried to specify an invalid constraint operator.
BRWS_ERR_ColCountTooHi	32525	You tried to specify the number of columns greater than the actual number of columns in the control.
BRWS_ERR_ColCountTooLow	32526	You tried to specify a negative or zero number of columns. The control must have at least one column.

Name	Value	Description
BRWS_ERR_ColWidthTooHi	32527	The user tried to specify too large a column width. The column cannot be more than 2000 pixels wide.
BRWS_ERR_ColWidthTooLow	32528	The user tried to specify too narrow a column width. The column cannot be less than 80 pixels wide.
BRWS_ERR_ColMinWidthTooHi	32529	The application program tried to specify too large a minimum column width. The column cannot be more than 2000 pixels wide.
BRWS_ERR_ColMinWidthTooLow	32530	The application program tried to specify too narrow a minimum column width. The column cannot be less than 80 pixels wide.
BRWS_ERR_ColWidthLessMinWidth	32531	The user tried to specify a column width narrower than the application program's minimum column width.

Methods

The Description column identifies the methods that are standard in Visual Basic; refer to the Microsoft Visual Basic documentation for details and examples.

Name	Standard	Description
Drag	Yes	Cancels, begins, or ends dragging, according to the action code.
Move	Yes	Moves the control.
Refresh	Yes	Repaints the control.

Events

The Description column identifies the events that are standard in Visual Basic; refer to the Microsoft Visual Basic documentation for details and examples.

Event	Parameters	Standard	Description
Click	Index As Integer	Yes	Occurs when the user clicks the mouse.
ConstrainButtonClicked	Column As Integer, Constraint As Integer	No	Occurs when the user clicks a command button on a column, to set the constraints for that column.
DragDrop	Index As Integer	Yes	Occurs after a drag-and-drop operation; that is, when the user drags another control through this control and releases the mouse button.

Event	Parameters	Standard	Description
DragOver	Index As Integer, Source As Control, X As Single, Y As Single	Yes	Occurs when a drag-and-drop operation is in progress. With this event, you can monitor when the pointer enters, stays on, or leaves this control.
GotFocus	Index As Integer	Yes	Occurs when the focus is on this control.
KeyDown	Index As Integer, KeyCode As Integer, Shift As Integer	Yes	Occurs if the user presses any key while the focus is on this control.
KeyPress	Index As Integer, KeyASCII As Integer	Yes	Occurs when the user presses and releases any character key.
KeyUp	Index As Integer, KeyCode As Integer, Shift As Integer	Yes	Occurs when the user releases any key, while the focus is on this control.
ListItemDraw	Column As Integer, ListItem As Integer, hDC As Long, rLeft As Integer, rTop As Integer, rRight As Integer, rBottom As Integer, Value As String	No	Occurs for each item in a data column whose Data Draw Event property is set to **True**. This event is provided for applications that draw list items in a particular way.
ListItemFormat	Column As Integer, ListItem As Integer, Text As String, Alignment As Integer	No	Occurs for each list item in a data column, when that column has the DataFillEvent property set to **True**. The event occurs before Browser assigns data to the list item.
LostFocus	Index As Integer	Yes	Occurs when the focus has been on this control but leaves.
Update	none	No	(For PowerBuilder only.) Occurs whenever the application's internal property values change, to force PowerBuilder to update its own property values. Your program should ignore this event.

Event Details

This section provides detailed descriptions of events in Browser that are unique to this control.

ConstrainButtonClicked

This event occurs when the user clicks a command button on a column to set the constraints for that column.

The syntax is:

Sub*Browser*_ConstrainButtonClicked (*Columm* As Integer, *Constraint* As Integer)

where:

Column	Specifies the column index of the command button that was clicked. Specify 0 (zero) for the first column.
Constraint	Specifies the current constraint value. Refer to the Column Constraint property for the operator value.

ListItemDraw

This event occurs for each item in a data column whose Data Draw Event property is set to True. ListItemDraw is provided for applications that draw list items in a particular way.

The syntax is:

Sub *Browser*_ListItemDraw (*Column* As Integer, *ListItem* As Integer, *hDC* As Long, *rLeft*As Integer, *rTop*As Integer, *rRight*As Integer, *rBottom*As Integer, *Value*As String)

where:

Column	An index to specify the column to be drawn. Specify 0 (zero) for the first column.
ListItem	An index to specify the list item to be drawn.
hDC	Specifies the handle to the device context. (For more information, refer to Microsoft Windows programming documentation.)
rLeft	Specifies the left-side screen coordinate of the list item to be drawn, in *twips*, or 1/1440ths of an inch.
rTop	Specifies the upper screen coordinate of the list item to be drawn, in twips.
rRight	Specifies the right-side screen coordinate of the list item to be drawn, in twips.
rBottomSub Browser_ListItemDraw (Column As Integer, ListItem As Integer, hDC As Long, rLeftAs Integer, rTopAs Integer, rRightAs Integer, rBottomAs Integer, *Value*As String)	Specifies the lower screen coordinate of the list item to be drawn, in twips.
Value	Specifies the text string to be drawn.

ListItemFormat

This event occurs for each list item in a data column, when that column has the DataFillEvent property set **True**. The event occurs before Browser assigns data to the list item.

The syntax is:

Sub *Browser*_ListItemFormat (*Column* As Integer, *ListItem*As Integer, *Text*As String, *Alignment*As Integer)

where:

Column	An index to specify the column whose data is to be assigned. Specify 0 (zero) for the first column.
ListItem	An index to specify the list item whose data is to be assigned.
Text	Specifies the text string of the list item.
Alignment	Specifies how the string is to be aligned. Valid values are: 1 = left 2 = center 3 = right

Update

For PowerBuilder only, this event occurs whenever the application's internal property values change. It forces PowerBuilder to update its own property values. Your application should ignore this event.

Properties

Some properties in Browser are unique to this control; detailed descriptions follow below. Other properties are standard in Visual Basic and the Description column identifies them as such.

Property	Data Type	Standard	Description
Action	integer	No	Specifies whether the control is to perform one of several possible actions, such as clearing data or saving data to a file.
Action_Index	integer	No	Specifies whether the control is to perform one of several possible actions, such as selecting and copying items from a data column to the clipboard.
Allow_ListBox_Dragging	Boolean	No	Specifies whether the width of the columns' list boxes can be changed by dragging their edges.
Background_Color	long	No	Specifies the control's background color.
BorderStyle	integer	Yes	Defines the type of border, if any.

Property	Data Type	Standard	Description
Caption	string	Yes	Specifies the text string to be displayed in or next to the control.
Column_Caption	string	No	Specifies or retrieves the title of the specified column.
Column_Constrain	Boolean	No	Constrains the specified column or returns its constraint status.
Column_Count	integer	No	Specifies or retrieves the number of columns that contain data.
Column_Min_Width	integer	No	Specifies or retrieves the smallest allowable width, in pixels, of the specified column.
Column_Operator	array	No	Specifies or retrieves the constraint operator for the specified column.
Column_Spacing	integer	No	Specifies or retrieves the amount of space between columns, in pixels.
Column_Width	integer	No	Specifies or retrieves the width, in pixels, of the specified column.
ConstrainButtonClicked_Event	Boolean	No	Specifies whether (TRUE) or not (FALSE) the ConstrainButtonClicked event is to occur when the user clicks on the constraint action button.
File_Name	string	No	Specifies the fully qualified path of the file to be used for a subsequent action. Use File_Name only when the Action property is: 1 (BRWS_ACTION_LOADBRWS_FILESZ) 2 (BRWS_ACTION_SAVEBRWS_FILESZ) 9 (BRWS_ACTION_LOADCONFIG_FILESZ) File_Name is ignored otherwise.
File_Num	integer	No	Specifies the file number to use when saving or loading a Browser object. Use File_Num only when the Action property is: 3 BRWS_ACTION_LOADBRWS_FILENUM) 4 (BRWS_ACTION_SAVEBRWS_FILENUM) File_Num is ignored otherwise.
Get_Column_Text	string	No	Retrieves a tab-delimited string of all data items from the specified column.

Property	Data Type	Standard	Description
Get_Selected_Text	string	No	Retrieves a tab-delimited string of selected data items from the specified column.
Height	single	Yes	Specifies the control's height, in twips, or 1/1440ths of an inch.
HWnd	Boolean	Yes	Specifies whether the control's focus box is to be drawn on the selected item in the list box (TRUE) or not drawn (FALSE).
Left	single	Yes	Specifies the distance, in pixels, between the internal left edge of the control and the left edge of the form it is in.
Left_Margin	integer	No	Specifies the distance, in pixels, between the internal left edge of the control and the start of the leftmost column of data.
ListItem_Count	Boolean	No	Specifies whether (TRUE) or not (FALSE) the ListItemFormatevent is to occur whenever a data item is about to be added to a specified column.
ListItemDraw_Event	Boolean	No	Specifies whether (TRUE) or not (FALSE) the ListItemDrawevent is to occur whenever a data item in a specified column is about to be drawn.
ListItemFormat_Event	Boolean	No	Specifies whether (TRUE) or not (FALSE) the ListItemFormatevent is to occur whenever a data item is about to be added to a specified column.
ListItem_Index	long	No	In conjunction with the Get_Selected_Textproperty, marks a data item to be retrieved.
ListItem_Index2	long	No	When used with ListItem_Index, marks a range of data items for retrieval.
ListItem_LeftMargin	integer	No	Specifies or retrieves the left margin, in pixels, of all data columns in the control.
ListItem_TopMargin	integer	No	Specifies or retrieves the upper margin, in pixels, of all data columns in the control.
Parent	form	Yes	Specifies which form is the parent of the control.
Redraw	Boolean	No	Specifies whether (TRUE) or not (FALSE) the screen is redrawn whenever any data item changes.
TabIndex	integer	Yes	Specifies the tabular order of the control within the form.
TabStop	Boolean	Yes	Specifies whether the user can use the Tab key to set the focus to this control.
Text	string	No	Inserts a tab-delimited row of data.

Property	Data Type	Standard	Description
TextBlock	string	No	Inserts two or more tab-delimited rows of data.
TextFile_Delimiter	integer	No	Specifies one fo five choices for the character used to separate fields of data within rows.
TextFile_WithCaptions	Boolean	No	Specifies whether the first row of a textual data file is column headings (TRUE) or data (FALSE).
Visible	Boolean	Yes	Specifies whether the control is hidden or visible.
Width	single	Yes	Specifies the width of the control.

Property Details

This section provides detailed descriptions of properties in Browser that are unique to this control. For details and examples of the standard Visual Basic properties, refer to the Microsoft Visual Basic documentation.

Action

Type: integer

This property specifies whether the control is to perform one of several possible actions, such as clearing data or saving data to a file.

The syntax is:

[*form.*]*browse.*Action[=*action*%]

where:

action% specifies the action to be taken:

0 or LOAD_TXTDATAFILE Load a data file which contains rows of tab-delimited data fields,using the File_Name property to specify the file.

1 or LOAD_FILENAME Load and reconstruct previously saved Browser data,using File_Name to specify the file.

2 or SAVE_FILENAME Save Browser data to a file, using File_Name to specify the file.

3 or LOAD_FILENUM Load and reconstruct previously saved Browser data,using File_Num to specify the file.

4 or SAVE_FILENUM Save Browser data to a file,using File_Num to specify the file.

5 or SELECT_ALL Select all data items currently visible in the Browser control.

6 or DESELECT_ALL Deselect all data items currently visible in the Browser control.

7 or CLIPBOARD_COPY Copy to the clipboard all data items currently visible in the control.

8 or CLEAR_ALL Clear all data items in the control.

9 or LOAD_CONFIGFILE Load a file containing the control's configuration and its data columns,using File_Name to specify the file.

This property is useful for performing actions on the entire control. Action can be assigned only. It is available during execution only. The Action_Index property is similar, but applies to specific columns.

For example, to load data from the file [verbar]Data1[verbar]Sales:

```
actionChoice= 2
File_Name = "[verbar]Data1[verbar]Sales"
myBrowser.Action= actionChoice
```

The Action property does not have a field in the control's private data structure, and therefore is not saved or loaded as a persistent property.

Action_Index

Type: integer

This property specifies whether the control is to perform one of several possible actions specific to a selected column, such as clearing data or saving data to a file.

The syntax is:

[*form.*]*browser.*Action_Index (*index%*)[= *action%*]

where:

index% A zero-based index indicating the column whose to be affected.

action% Specifies the action to be taken:

0	Select the data item indicated by the ListItem_Index property.
1	Deselect the data item indicated by the ListItem_Index property.
2	Select the block of data items indicated by the ListItem_Index and ListItem_Indexproperties.
3	Deselect the block of data items indicated by the ListItem_Indexand ListItem_Indexproperties.
4	Select all data items in the specified column.
5	Deselect all data items in the specified column.
6	Copy to the clipboard all data items in the specified column.

This property is useful for performing actions on specific columns. Action can be assigned only. It is available during execution only. The Action property is similar, but applies to the entire control.

For example, to load data from the file [verbar]Data1[verbar]Sales:

```
actionChoice=
ListItem_Index =3
myBrowser.Action_Index = actionChoice
```

The Action property does not have a field in the control's private data structure, and therefore is not saved or loaded as a persistent property.

Allow_ListBox_Dragging

Type: Boolean

This property specifies whether the width of the columns' list boxes can be changed by dragging their edges.

Its syntax is:

[*form.*]*browser.*Allow_ListBox_Dragging [=*status%*]

where:

status% specifies whether to let the user change column widths:

True	allowed
False	not allowed

Background_Color

Type: long

This property specifies the background color of the overall Browser control.

Its syntax is:

[*form.*]*browser*.Background_Color [=*color*]

where:

color	Specifies the color.

For example, either of these statements define the background color as orange:

```
myBrowser.Background_Color =RGB(255,165,66)
myBrowser.Background_Color= &H42A5FF
```

Column_Caption

Type: string

This property specifies or retrieves the title of the specified column.

Its syntax is:

[*form.*]*browser*.Column_Caption (*index*%)[=*string*$]

where:

index%	A zero-based index indicating the column whose title is to be specified or retrieved.
string$	The column title.

For example, to specify the title of the control's third column (column 2):

```
myBrowser.Column_Caption(2)= "Product Information"
```

To retrieve the title of the control's third column (column 2):

```
title= myBrowser.Column_Caption(2)
```

Column_Constrain

Type: Boolean

This property constrains the specified column or returns its constraint status.

Its syntax is:

[*form.*]*browser*.Column_Constrain(*index*%)[=*status*%]

where:

index%	is a zero-based index indicating the column whose constraint status is to be specified or retrieved.
status%	indicates whether the column is constrained: **True** constrained **False** not constrained

For example, to constrain the control's third column (column 2):

```
myBrowser.Column_Constrain(2)= True
```

To discover whether the control's third column (column 2) is constrained:

```
test = myBrowser.Column_Constrain(2)
```

Column_Count

Type: integer

This property specifies or retrieves the number of data columns the control is to contain.

If you use this property on an existing Browser control, that control is deleted and replaced by one that has the specified number of empty columns.

You can use this property during design and during execution.

Its syntax is:

[*form.*]*browser.*Column_Count[=*number%*]

where:

number% specifies how many data columns the control is to contain.

For example, to create a five-column Browser control:

```
myBrowser.Column_Count= 5
```

Column_Min_Width

Type: integer

This property specifies or retrieves the smallest allowable width, in pixels, of the specified column. Its syntax is:

[*form.*]*browser.*Column_Min_Width (*index%*)[=*width%*]

where:

index% is a zero-based index indicating the column whose minimum width is to be specified or retrieved.

width% indicates the minimum width, in pixels.

For example, to set the minimum width of the control's third column (column 2) to 50 pixels:

```
myBrowser.Column_Min_Width(2)= 5
```

Column_Operator

Type: array

This property specifies or retrieves the constraint operator for the specified column. Its syntax is:

[*form.*]*browser.*Column_Operator(*index%*)[=*operator%*]

where:

index%	is a zero-based index indicating the column to be constrained.
operator%	specifies the constraint operator: **0 or CnstEQ** equal to ( ) **1 or CnstNE** not equal to () **2 or CnstLT** less than (<) **3 or CnstLE** less than or equal to (<) **4 or CnstGT** greater than (>) **5 or CnstGE** greater than or equal to (>) **6 or CnstBT** between **7 or CnstNB** not between

For example, either of these statements constrains the control's third column (column 2) to values greater than a selected data value:

```
myBrowser.Column_Operator(2)= 4
myBrowser.Column_Operator(2) =CnstGT
```

To find the current operator for the control's third column:

```
columnOperator= myBrowser.Column_Operator(2)
```

Column_Spacing

Type: integer

This property specifies or retrieves the amount of space between columns, in pixels. Its syntax is:

[*form.*]*browser.*Column_Spacing [= *space%*]

where:

space%	specifies how many pixels' space to insert between columns.

For example, to set the spacing to 10 pixels:

```
myBrowser.Column_Spacing= 1
```

Column_Width
Type: integer

This property specifies or retrieves the actual width, in pixels, of the specified column. Its syntax is:

[*form*.]*browser*.Column_Width(*index*%)[=*width*%]

where:

index%	is a zero-based index indicating the column whose width is to be specified or retrieved.
width%	indicates the width, in pixels.

For example, to make the control's third column (column 2) to 50 pixels wide:

```
myBrowser.Column_Width(2)= 5
```

To retrieve that column's current width:

```
colWidth= myBrowser.Column_Width(2)
```

ConstrainButtonClicked_Event
Type: Boolean

This property specifies whether the ConstrainButtonClicked event is to occur when the user clicks on the constraint-action button.

Its syntax is:

[*form*.]*browser*.ConstrainButtonClicked_Event (*index*%)[=*status*%]

where:

index%	is a zero-based index indicating the column whose event status is to be specified or retrieved.
status%	indicates whether the event is to occur: **True** event is to occur when the user clicks on the button **False** event is not enabled

For example, to enable the event:

```
myBrowser.ConstrainButtonClicked_Event(2)= True
```

Constrain Designer
This property, unlike most others, is not a variable. When you double-click it during design, it opens a dialog box in in which you can manipulate the controls to set the font, typeface, and color of the text to be displayed for both constrained and unconstrained states of the column.

File_Name
Type: string

This property specifies the fully qualified path of the file to be used for a subsequent action. File_Name is ignored if Action is not 1, 2, or 9.

You can use this property during execution, not during design. Its syntax is:

[*form.*]*browser.*File_Name [=*string*$]

where:

string$ is the fully qualified path.

For example, to write data to the specified path:

```
myBrowser.File_Name= "D:\BRDATA\BROWSER.DAT"
myBrowser.Action= SAVE_FILENAME
```

File_Num
Type: integer

This property specifies the file number to use when saving or loading a Browser object. The application must have already opened the file, in binary mode. File_Num uses the same number as is used in the Open statement.

File_Num is ignored if Action is not 3 or 4.

You can use this property during execution, not during design. Its syntax is:

[*form.*]*browser.*File_Num [=1 t*number*%]

where:

number% specifies the identifying number of a file to be loaded or saved.

For example, to write data to the specified file:

```
Open"BROWSER.DAT" For Binary As #1
myBrowser.File_Num= 1
myBrowser.Action = SAVE_FILENUM
Close #1
```

Get_Column_Text
Type: string

This property retrieves a tab-delimited string of all data items from the specified column.

To retrieve all data from the column, you can use Get_Selected_Text, in conjunction with ListItem_Index and ListItem_Index2.

You can use this property only during execution, not during design. You cannot use it to assign data. Its syntax is:

[*string*$=][*form.*]*browser.*Get_Column_Text (*index*%)

where:

index% is a zero-based index indicating the column from which to retrieve data.

For example, to retrieve all of the data in the control's third column (column 2):

```
fetchText =myBrowser.Get_Column_Text(2)
```

Get_Selected_Text

Type: string

This property retrieves a tab-delimited string of selected data items from the specified column. Use it in conjunction with ListItem_Index to select a single item, or with both ListItem_Index and ListItem_Index2 to select a range of items.

To retrieve all data from the column, you can use Get_Column_Text.

You can use this property during execution, not during design. You cannot use it to assign data. Its syntax is:

[*string*$=][*form.*]*browser.Get_Selected_Text*(*index*%)

where:

index% is a zero-based index indicating the column from which to retrieve data.

For example, to retrieve the fifth data item in the control's third column (column 2):

```
ListItem_Index = 4
fetchText= myBrowser.Get_Selected_Text(2)
```

Left_Margin

Type: integer

This property specifies the distance, in pixels, between the internal left edge of the control and the start of the leftmost column of data. Its syntax is:

[*form.*]*browser.*Left_Margin [= *space*%]

where:

space% specifies how many pixels' space to insert, from a minimum of 0 (zero) to a maximum of 50.

For example, to set the spacing to 10 pixels:

```
myBrowser.Left_Margin= 1
```

ListItem_Count

Type: integer

This property stores the number of visible data items in a specified column.

You can use this property during execution, not during design. Its syntax is:

items=[*form*.]*browser*.ListItem_Count (*index*%)

where:

items	is the number of data items in the column.
index%	is a zero-based index indicating the column to be counted.

For example, to find how many data items are visible in the control's third column (column 2):

```
howMany= myBrowser.ListItem_Count(2)
```

ListItemDraw_Event

Type: Boolean

This property specifies whether the ListItemDrawevent is to occur whenever a data item in a specified column is about to be drawn. The event is useful if you want to control when data items are redrawn or not redrawn. When enabled, the event occurs for every data item within the specified column when the application is reading the data.

You can use this property only during execution, not during design. Its syntax is:

[*form*.]*browser*.ListItemDraw_Event (*index*%)[=*status*%]

where:

index%	is a zero-based index indicating the column whose event status is to be specified or retrieved.
status%	indicates whether the event is to occur: **True** event is to occur when a data item is about to be drawn **False** event is not enabled The default is False for each column.

For example, to enable the event:

```
myBrowser.ListItemDraw_Event(2)= True
```

ListItemFormat_Event

Type: Boolean

This property specifies whether or not the ListItemFormatevent is to occur whenever a data item is about to be added to a specified column. The event is useful if you want to alter data before writing the data to the column. When enabled, the event occurs for every data item within the specified column when the application is reading the data.

You can use this property only during execution, not during design. Its syntax is:

[*form.*]*browser.*ListItemFormat_Event (*index*%)[=*status*%]

where:

index% is a zero-based index indicating the column whose event status is to be specified or retrieved.

status% indicates whether the event is to occur:

True event is to occur whenever a data item is about to be added

False event is not enabled

The default is False for each column.

For example, to enable the event:

```
myBrowser.ListItemFormat_Event(2)= True
```

ListItem_Index

Type: long

This property marks a data item for retrieval or other action with the Action_Index, Get_Column_Text, or Get_Selected_Text property.

You can use this property only during execution, not during design. Its syntax is:

[*form.*]*browser.*ListItem_Index [=*index*%]

where:

index% is a zero-based index indicating an item within a column.

For example, to retrieve the fifth data item in the control's third column (column 2):

```
ListItem_Index = 4
fetchText= myBrowser.Get_Selected_Text(2)
```

ListItem_Index2

Type: long

This property, when used with ListItem_Index, marks the end of a range of data items for retrieval or other action, in conjunction with the Action_Index, Get_Column_Text, or Get_Selected_Textproperty. ListItem_Index2 must be larger than ListItem_Index.

You can use this property during execution, not during design. Its syntax is:

[*form.*]*browser.*ListItem_Index [= *index%*]

where:

index%	is a zero-based index indicating an item within a column.

For example, to retrieve the fifth through seventh data items in the control's third column (column 2):

```
ListItem_Index = 4
ListItem_Index2 = 6
fetchText= myBrowser.Get_Selected_Text(2)
```

ListItem_LeftMargin

Type: integer

This property specifies or retrieves the left margin, in pixels, of all data columns in the control. Its syntax is:

[*form.*]*browser.*ListItem_LeftMargin [=*space%*]

where:

space%	specifies how many pixels' space to insert, from a minimum of 0 (zero) to a maximum of 10.

For example, to set the spacing to 10 pixels:

```
myBrowser.ListItem_LeftMargin= 1
```

ListItem_TopMargin

Type: integer

This property specifies or retrieves the upper margin, in pixels, of all data columns in the control. Its syntax is:

[*form.*]*browser.*ListItem_TopMargin [= *space%*]

where:

space%	specifies how many pixels' space to insert, from a minimum of 0 (zero) to a maximum of 10.

For example, to set the spacing to 10 pixels:

```
myBrowser.ListItem_TopMargin= 1
```

Property Designer

This property, unlike most others, is not a variable. When you double-click on it during design, it opens a dialog box.

In that window, you can manipulate the controls to set the number of columns, size of each column, default column operator, and other properties.

Redraw

Type: Boolean

This property specifies whether or not the screen is redrawn whenever any data item changes. Its typical use is to delay updating the screen until all desired changes have been made. If the application performs a lengthy series of operations that would cause the control to redraw (such as font changes), this could cause a flickering screen.

Its syntax is:

[*form.*]*browser*.Redraw [=*status*%]

where:

status% indicates whether to redraw the screen:

- **True** Redraw the screen whenever needed.
- **False** Disable redrawing the screen until Redraw is reset.

For example, to disable redrawing:

```
myBrowser.Redraw= False
myBrowser.Column_Caption( ) ="Product Information"
myBrowser.Column_Caption(1)= "Region"
myBrowser.Column_Caption(2)= "City"
myBrowser.Redraw =True
```

Text

Type: string

This property inserts one tab-delimited row of data. To insert more than one row at a time, you can use TextBlock. The syntax of Text is:

[*form.*]*browser*.Text [= *string*$]

where:

string$ includes the data for every column in the control, separated by Chr$(9) (tab character). Chr$(13) (return character) must terminate the entire string.

TextBlock
Type: string

This property inserts two or more tab-delimited rows of data. To insert only one row at a time, you can use Text. The syntax of TextBlockis:

[*form*.] *browser*.TextBlock [= *string*$]

where:

string$ includes the data for every column in the control, separated by Chr$(9) (tab character). Chr$(13) (return character) must terminate each row's data including the last row in the entire string.

TextFile_Delimiter
Type: integer

This property specifies one of five choices for the character used to separate fields of data within rows. Its syntax is:

[*form*.] *browser*.TextFile_Delimiter[= *number*%]

where:

number% specifies the delimiting character to be used:

0	tab
1	semicolon (;)
2	comma (,)
3	space
4	colon (:)

This property is useful when you expect data to contain the previously set delimiting character. The default delimiter is a tab character.

For example, if you want to insert comma-delimited data:

```
myBrowser.TextFile_Delimiter= 3
myBrowser.Text = "Soap, Western, Seattle" + Chr$(13)
```

TextFile_WithCaptions
Type: Boolean

This property specifies whether the first row of a textual data file is column headings or data. The default value is False. Its syntax is:

[*form*.] *browser*.TextFile_WithCaptions [= *status*%]

where:

status% indicates whether the first row is column headings or data:

True column headings

False data

For example, to declare that the first row is column headings:

```
myBrowser.TextFile_WithCaptions =True
```

DisCapQ

The following sections provide information on DisCapQ limitations, assumptions, errors, methods, and properties.

Limitations

DisCapQ does not support batch mode.

DisCapQ does not support help.

DisCapQ cannot define users. If you want only the capsule job owner to be able change the capsule job status, you must provide this security at the application level.

DisCapQ does not support any filter for the job listing.

Assumptions

DisCapQ assumes that the client and server are already connected.

Error Codes

DisCapQ returns three types of error codes:

Codes received from the ODAS server, which are defined in the DISERR.TXT file.

Visual Basic codes

Internal codes as listed in the following table.

Name	Value	Description
CAP_ERR_MEMALLOCFAILURE	32000	The computer does not have enough memory for DisCapQ to perform the requested action.

Methods

The only method used by DisCapQ is standard in Visual Basic; refer to the Microsoft Visual Basic documentation for its details and example.

Name	Standard	Description
Refresh	Yes	Refreshes the listing of the contents of the Capsule job queue.

Events

The Description column identifies the events that are standard in Visual Basic; refer to the Microsoft Visual Basic documentation for their details and examples.

Event	Parameters	Standard	Description
Click	Index As Integer	Yes	Occurs when the user clicks the mouse.
DblClick	Index As Integer	Yes	Occurs when the user double-clicks the mouse.
DragDrop	Index As Integer	Yes	Occurs when the user drags another control through this control and releases the mouse button.
DragOver	Index As Integer, Source As Control, X As Single, Y As Single	Yes	Occurs when a drag-and-drop operation is in progress. With this event, you can monitor when the pointer enters, stays on, or leaves this control.
GotFocus	Index As Integer	Yes	Occurs when the focus is on this control.
KeyDown	Index As Integer, KeyCode As Integer, Shift As Integer	Yes	Occurs when the user presses any key while the focus is on this control.
KeyPress	Index As Integer, KeyAscii As Integer	Yes	Occurs when the user presses and releases any character key.
KeyUp	Index As Integer, KeyCode As Integer, Shift As Integer	Yes	Occurs when the user releases any key while the focus is on this control.
LostFocus	Index As Integer	Yes	Occurs when the focus has been on this control but leaves.
MouseDown	Index As Integer, Button As Integer, Shift As Integer, X As Integer, Y As Integer	Yes	Occurs when the user presses a mouse button.
MouseMove	Index As Integer, Button As Integer, Shift As Integer, X As Integer, Y As Integer	Yes	Occurs when the user moves the mouse.

Event	Parameters	Standard	Description
MouseUp	Index As Integer, Button As Integer, Shift As Integer, X As Integer, Y As Integer	Yes	Occurs when the user releases a mouse button.
Update	none	No	(For PowerBuilder only.) Occurs whenever the application's internal property values change, to force PowerBuilder to update its own property values. Your program should ignore this event.

Properties

Some properties in DisCapQ are unique to this control; detailed descriptions follow below. Other properties are standard in Visual Basic and the Description column identifies them as such; refer to the Microsoft documentation for their details and examples.

Property	Data Type	Standard	Description
About	(none)	No	Displays a pop-up window that shows copyright and version information about DisCapQ.
BackColor	long	Yes	Specifies the control's background color.
CurCapId	long	No	Specifies the job number that the control must use to retrieve the Status property.
DragIcon	bitmap	Yes	Specifies what icon, if any, to display as the pointer in drag-and-drop operations.
DragMode	integer	Yes	Specifies whether to use manual (FALSE) or automatic (TRUE) dragging mode for drag-and-drop operations.
Enabled	Boolean	Yes	Specifies whether the control can (TRUE) or cannot (FALSE) respond to user-generated events.
ErrCode	long	No	(For PowerBuilder only.) Returns the error code after any operation.
FontBold	Boolean	Yes	Specifies whether the control's text is boldface (TRUE) or not (FALSE).
FontItalic	Boolean	Yes	Specifies whether the control's text is italic (TRUE) or roman (FALSE).
FontName	string	Yes	Specifies the font family used for the control's text.
FontSize	integer	Yes	Specifies the font size used for the control's text.
FontStrikethru	Boolean	Yes	Specifies whether the control's text is in the strike-through style (TRUE) or not (FALSE).

Property	Data Type	Standard	Description
FontUnderline	Boolean	Yes	Specifies whether the control's text is underscored (TRUE) or not (FALSE).
ForeColor	long	Yes	Specifies the control's foreground color.
Height	single	Yes	Specifies the control's height, in twips, or 1/1440ths of an inch.
HideFocusBox	Boolean	Yes	Specifies whether the control's focus box is to be drawn on the selected item in the list box (TRUE) or not drawn (FALSE).
Index	integer	Yes	Specifies a number that uniquely identifies the control in a control array. Valid values are from 0 (zero) through 32767.
Left	single	Yes	Specifies the distance between the internal left edge of the control and the left edge of the form it is in.
List	string	Yes	Lists the name of a capsule in the queue.
ListAliasName	string	No	Stores the scheduling alias, if any, with which the currently selected capsule is scheduled.
ListCount	integer	Yes	Stores the total number of capsule listed in the list box (through scrolling; not just the items visible at one time).
ListId	long	No	Stores the job number of the currently selected capsule in the list.
ListIndex	integer	Yes	Stores the index of the currently selected capsule in the list; that is, it identifies which item is selected.
ListMaxExecution	long	No	Stores the capsule's maximum allowable execution time, in seconds.
ListMaxStartDelay	long	No	Stores the capsule's maximum allowable delay past its scheduled starting time, in seconds.
ListModify	single	No	Specifies whether and how to modify the state of the currently selected capsule in the queue.
ListOwner	string	No	Stores the name of the owner of the currently selected capsule in the list.
ListPriority	single	No	Stores the priority of the currently selected capsule in the list.
ListTimeScheduled	string	No	Stores the time and date when the currently selected capsule is next scheduled for execution.
ListState	single	No	Stores the state of the currently selected capsule in the list.
ListTimeSubmitted	string	No	Stores the time and date when the currently selected capsule was submitted.

Property	Data Type	Standard	Description
MDIChildWindow	Boolean	Yes	Specifies whether the control is to behave as an independent "MDI child window" (TRUE) or not (FALSE).
MousePointer	integer	Yes	Specifies the type and shape of the pointer (cursor) whenever it is over this control.
Name	string	Yes	Specifies the name of the control.
Parent	form	Yes	Specifies which form is the parent of the control.
PowerBuilder	Boolean	No	Specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.
Status	long	No	Stores the completion status of the Capsule job, using the job number specified in the CurCapId property.
TabIndex	integer	Yes	Specifies the tabular order of the control within the form.
TabStop	Boolean	Yes	Specifies whether the user can use the Tab key to set the focus to this control.
Tag	string	Yes	Stores any extra data needed by your application. Developers usually use this property to identify the code object.
Text	string	Yes	Specifies the window caption when the MDIChildWindow property = TRUE.
Top	string	Yes	Specifies the distance between the internal top edge of the control and the top edge of the form. This value applies during the design phase only, because the control is invisible when the user runs your application.
Visible	Boolean	Yes	Specifies whether the control is hidden or visible.
Width	single	Yes	Specifies the width of the control.

Property Details

This section provides detailed descriptions of properties in DisCapQ that are unique to this control. For details and examples of the standard Visual Basic properties, refer to the Microsoft Visual Basic documentation.

About

This property displays a pop-up window that shows copyright and version information about DisCapQ.

CurCapId
Type: long

This property specifies the job number that the control must use to retrieve the Status property.

For example, to find the status of the first job in the queue:

```
index =
myDisCapQ.ListId = index
jobStatus = myDisCapQ.Status
```

ErrCode
Type: long

(For PowerBuilder only.) This property returns the error code after any operation.

For example, to cancel the first job in the capsule job queue and check the result of the operation:

```
myDisCapQ.List Modify( ) = 1
If myDisCapQ.ErrCode <>
Then ReportError()
EndIf
```

ListAliasName
Type: string

This property stores the alias, if any, with which the currently selected capsule job is scheduled.

For example:

```
aliasName = myDisCapQ.ListAliasName(index)
```

or:

```
myDisCapQ.ListAliasName(index) = txtAlias
myDisCapQ.ListModify(index) = 5
```

ListId
Type: long

This property stores the job number of the currently selected capsule job in the list. This property cannot be changed during capsule execution.

For example:

```
jobNumber = myDisCapQ.ListId(index)
```

ListMaxExecution
Type: long

This property stores the capsule job's maximum allowable execution time in seconds.

For example, to give the fourth job in the queue a maximum execution time of five minutes:

```
index = 3
maximumTime = 3
myDisCapQ.ListMaxStartDelay(index)= maximumTime
myDisCapQ.ListModify(index)= 5
```

ListMaxStartDelay
Type: long

This property stores the capsule job's maximum allowable delay past its scheduled starting time, in seconds.

For example, to give the fourth job in the queue a maximum delay of five minutes:

```
index = 3
maximumDelay = 3
myDisCapQ.ListMaxStartDelay(index)= maximumDelay
myDisCapQ.ListModify(index)= 5
```

ListModify
Type: single

This property specifies whether and how to modify the state of the currently selected capsule job in the list. Valid values are:

1 Cancel the job and remove it from the queue.

2 Purge a job that is scheduled for periodic execution.

3 Hold the job until it is released.

4 Release a job that is being held.

5 Modify one or more job parameters by sending the current settings of the corresponding properties. You can use this feature with the ListPriority, ListAliasName, ListDescription, ListTimeScheduled, ListExecution, and ListMaxStartDelay properties.

For example, to cancel the first job in the queue:

```
index =
myDisCapQ.ListModify(index)= 1
```

To modify the fourth job's scheduled execution time:

```
index = 3
newTime = "1 /12/94 18:3 :  "
myDisCapQ.ListTimeScheduled(index) = newTime
myDisCapQ.ListModify(index)= 5
```

ListOwner

Type: string

This property stores the name of the owner of the currently selected capsule job in the list. This property cannot be changed during capsule execution.

For example:

```
jobOwner = myDisCapQ.ListOwner
```

ListPriority

Type: single

This property stores the priority of the currently selected capsule job in the list.

For example, to modify the priority of the fourth job in the queue:

```
index = 3
newPriority = 8
myDisCapQ.ListPriority(index)= newPriority
myDisCapQ.ListModify(index)= 5
```

ListTimeScheduled

Type: string

This property stores the time and date when the currently selected capsule job is next scheduled for execution.

Each submission should have a value for either ListAliasName or ListTimeScheduled, but not both. The format is:

mm/dd/yy hh:mm:ss

where:

mm the month of the year (1 through 12)

dd the day of the month (1 through 31)

yy the last two digits of the year (94 through 99)

hh the hour of the 24-hour day (0 through 23)

mm the minute (0 through 59)

ss the second (0 through 59)

For example, to modify the scheduled execution time of the fourth job in the queue:

```
index = 3
newTime = "1 /12/94 18:3 :  "
myDisCapQ.ListTimeScheduled(index) =
newTime myDisCapQ.ListModify(index)= 5
```

ListState

Type: single

This property stores the state of the currently selected capsule job in the list. This property cannot be changed during capsule execution.

Valid values are:

1 The job is waiting for its scheduled time.

2 The job is waiting for an execution resource.

3 The job is being held.

4 The job is being executed.

5 The job has abnormally terminated due to either cancellation or an execution error.

For example:

```
jobState = myDisCapQ.ListState(index)
Select Case jobState
 Case1: lblStatus = "WAITING"
 Case2: lblStatus = "READY"
 Case3: lblStatus = "HELD"
 Case4: lblStatus = "RUNNING"
 Case5: lblStatus = "ABENDED"
End
Select
```

ListTimeSubmitted

Type: string

This property stores the time and date when the currently selected capsule job was submitted. (Read-only, during execution)

The format is:

mm/dd/yy hh:mm:ss

where:

mm the month of the year (1 through 12)

dd the day of the month (1 through 31)

yy the last two digits of the year (94 through 99)

hh the hour of the 24-hour day (0 through 23)

mm the minute (0 through 59)

ss the second (0 through 59)

For example:

```
jobTime = myDisCapQ.ListTimeSubmitted
```

PowerBuilder

Type: Boolean

This property specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.

For example, to specify that you are not using PowerBuilder:

```
myDisCapQ.PowerBuilder = FALSE
```

Status

Type: long

This property stores the completion status of the capsule job, using the job number specified in the CurCapId property. This property cannot be changed during capsule execution.

Valid values are:

0 = The job is completed.

1 = The job is not completed.

For example:

```
myDisCapQ.CurCapId = myJobId
jobStatus = myDisCapQ.Status
```

DisCom and ODisCom

The following sections provide information on DisCom and ODisCom limitations, errors, events, and properties.

Limitations

If the ODAS server drops the connection, the control's behavior is unpredictable, because it has no way to query the server to discover the connection state between the client and the server.

The control does not support batch mode.

The control does not support help.

Only one instance of the control can be running at any one time.

Assumptions

The control assumes that:

The client computer is disconnected from the server when the application is loaded into memory.

Any connection to the server is through use of DisCom. If the application connects to the server by itself, and then tries to log on to a desktop using the control, the control's behavior is unpredictable.

Error Codes

The control returns three types of error codes:

Codes received from the ODAS server, which are defined in the DISERR.TXT file.

Visual Basic codes.

Internal code as listed in the following table.

Name	Value	Description
DISCOM_ERR_BADFORMAT	32000	The property value has the wrong format.
DISCOM_ERR_ALREADY_CONNECTED	32001	The application tried to connect to the ODAS server when the connection was already established.
DISCOM_ERR_NOTCONNECTED	32002	The application tried to disconnect from the ODAS server when the connection was already broken.
DISCOM_ERR_NODEFINEDDESKTOP	32003	The application tried to log on before an IDS desktop was specified.
DISCOM_ERR_NODEFINEDPASSWORD	32004	The application tried to log on before a desktop password was specified.
DISCOM_ERR_ALREADYLOGON	32005	The application tried to log on when it was already logged onto an IDS desktop.
DISCOM_ERR_ALREADYLOGOFF	32006	The application tried to log off when it was not logged onto any IDS desktop.

Events

The Standard column identifies the events that are standard in Visual Basic; refer to the Microsoft Visual Basic documentation for their details and examples.

Event	Parameters	Standard	Description
AFDExecBegin	ErrorCode As Integer, FileSendAs Integer, FileRcvAs Integer, ByteSend As Integer, ByteRcv As Integer	No	ODisCom only: occurs when AFD begins execution.
AFDExecEnd	ErrorCode As Integer, FileSendAs Integer, FileRcvAs Integer, ByteSend As Integer, ByteRcv As Integer	No	ODisCom only: occurs when AFD finishes execution.
AFDInitBegin	none	No	ODisCom only: occurs when AFD begins initializing.
AFDInitEnd	ErrorCode As Integer, FileSendAs Integer, FileRcvAs Integer, ByteSend As Integer, ByteRcv As Integer	No	ODisCom only: occurs when AFD finishes initializing.
Connected	none	No	Occurs when connection to a ODAS or Global Access server is established.
FileBegin	FileSizeAs Long, FileName As String	No	ODisCom only: occurs when AFD begins transferring a file.
FileProgress	ElapsedTimeAs Long, XferSizeAs Long, FileSizeAs Long	No	ODisCom only: occurs after every 8 kilobytes while AFD is transferring a file.
FileEnd	ErrorCodeAs Long, SourceName As String	No	ODisCom only: occurs when AFD finishes transferring a file.
Update	none	No	(For PowerBuilder only.) Occurs whenever the application's internal property values change, to force PowerBuilder to update its own property values. Your program should ignore this event.

Event Details

This section provides detailed descriptions of events in DisCom and ODisCom that are unique to these controls. For details and examples of the standard Visual Basic events, refer to the Microsoft Visual Basic documentation.

AFDExecBegin

ODisCom only: this event occurs when AFD begins execution. The syntax is:

Sub *ODisCom*_AFDExecBegin (*ErrorCode* As Integer, *FileSend*As Integer, *FileRcv*As Integer, *ByteSend* As Integer, *ByteRcv* As Integer)

where:

ErrorCode	Returns the error code from this process.
FileSend	Returns the number of files to be sent.
FileRcv	Returns the number of files to be received.
ByteSend	Returns the number of bytes to be sent.
ByteRcv	Returns the number of bytes to be received.

AFDExecEnd

ODisCom only: this event occurs when AFD finishes execution. The syntax is:

Sub *ODisCom*_AFDExecEnd (*ErrorCode* As Integer, *FileSend*As Integer, FileRcvAs Integer, *ByteSend* As Integer, *ByteRcv* As Integer)

where:

ErrorCode	Returns the error code from this process.
FileSend	Returns the number of files to be sent.
FileRcv	Returns the number of files to be received.
ByteSend	Returns the number of bytes to be sent.
ByteRcv	Returns the number of bytes to be received.

AFDInitBegin

ODisCom only: this event occurs when AFD begins initializing. If your application starts AFD, you can cancel it only during the AFD initialization phase. You cannot cancel AFD before it receives the AFDInitBegin event or after it receives the AFDInitEnd event.

The syntax is:

Sub *ODisCom*_AFDInitBegin ()

AFDInitEnd

ODisCom only: this event occurs when AFD finishes initializing. You cannot cancel AFD after it receives the AFDInitEnd event.

The syntax is:

Sub *ODisCom*_AFDInitEnd (*ErrorCode* As Integer, *FileSend*As Integer, *FileRcv*As Integer, *ByteSend* As Integer, *ByteRcv* As Integer)

where:

ErrorCode	Returns the error code from AFD.
FileSend	Returns the number of files to be sent.
FileRcv	Returns the number of files to be received.
ByteSend	Returns the number of bytes to be sent.
ByteRcv	Returns the number of bytes to be received.

FileBegin

ODisCom only: this event occurs when AFD begins transferring a file.

The syntax is:

Sub *ODisCom*_FileBegin (*FileSize*As Long, *FileName* As String)

where:

FileSize	Specifies the size, in bytes, of the file being transferred.
FileName	Specifies the name of the file being transferred.

FileEnd

ODisCom only: this event occurs when AFD finishes transferring a file.

The syntax is:

Sub *ODisCom*_FileEnd (*ErrorCode*As Long, *SourceName* As String)

where:

ErrorCode	Returns the error code from the file transfer.
SourceName	Specifies the name of the file being transferred.

FileProgress

ODisCom only: this event occurs while AFD is transferring a file.

The syntax is:

Sub *ODisCom*_FileProgress (*ElapsedTime*As Long, *XferSize*As Long, *FileSize*As Long)

where:

ElapsedTime	Returns the elapsed time, in seconds, since the file transfer started.
XferSize	Returns the number of bytes that were transferred.
FileSize	Specifies the size, in bytes, of the file being transferred.

Update

This event occurs for PowerBuilder only. It occurs whenever the application's internal property values change, to force PowerBuilder to update its own property values. Your program should ignore this event.

Properties

Some properties in DisCom and ODisCom are unique to this control. Other properties are standard in Visual Basic and the Standard column identifies them as such; refer to the Microsoft documentation for their details and examples.

Property	Data Type	Standard	Description
About	(none)	No	Displays a pop-up window that shows copyright and version information about the control.
AutoAFD	Boolean	Yes	ODisCom only, specifies whether to enable (TRUE) or disable (FALSE) AFD.
BaudRate	integer	No	Specifies the asynchronous connection baud rate for connection to a Global Access server. For normal ODAS use, this property is ignored. If your application is to connect with a Global Access server, see “BaudRate” on page 244.
CommPort	integer	No	Specifies which communications port is to be used for connection to a Global Access server. For normal ODAS use, this property is ignored. If your application is to connect with a Global Access server, see “CommPort” on page 244.
CommType	integer	No	Specifies the type of connection to the ODAS server. For normal ODAS use, specify 2. If your application is to connect with a Global Access server, see “CommType” on page 245.
Connect or ConnectDIS	Boolean	Yes	Specifies whether the control is to initiate (TRUE) or disconnect (FALSE) the connection to the ODAS server. Use ConnectDIS for PowerBuilder or Connect for Visual Basic.
Desktop	string	No	Specifies the name of the IDS desktop to be logged on.
Enabled	Boolean	Yes	Specifies whether the control can respond to user-generated events, such as mouse events: TRUE = Default value; control responds. FALSE = All buttons are dimmed and disabled.

Property	Data Type	Standard	Description
ExecAFD	Boolean	Yes	ODisCom only: specifies whether to start (TRUE) or not start (FALSE) a single instance of AFD.
ErrCode	long	No	(For PowerBuilder only.) Returns the error code after any operation.
Init	Boolean	No	When set to TRUE, calls DisInitClient() to initialize the session before the control connects to the server, and calls DisTermClient() to terminate the session when the control disconnects from the server.
Left	single	Yes	Specifies the distance between the internal left edge of the control and the left edge of the form it is in.
Logon	Boolean	No	Specifies whether the control is to log on to (TRUE) or off from (FALSE) the specified IDS desktop.
Name	string	Yes	Specifies the name of the control.
Password	string	No	Specifies the user's IDS password. The user must enter this string; you cannot write it into the application.
PhoneNumber	string	No	Specifies the telephone number to be used for connection to a Global Access server. For normal ODAS use, this property is ignored. If your application is to connect with a Global Access server, see "PhoneNumber" on page 247.
PowerBuilder	Boolean	No	Specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.
ServerName	string	No	Specifies the name of the ODAS or Global Access server with which the client is to connect; applicable only if CommType = 2.
Tag	string	Yes	Stores any extra data needed by your application. Typically used to identify the code object.
TimeOut	integer	No	Specifies the connection timeout in seconds; applicable only if CommType = 2.
Top	single	Yes	Specifies the distance between the internal top edge of the control and the top edge of the form. This value applies during the design phase only, because the control is invisible when the user runs your application.

Property Details

This section provides detailed descriptions of those properties in DisCom that are unique to this control. For details and examples of the standard Visual Basic properties, refer to the Microsoft documentation.

About

This property displays a pop-up window that shows copyright and version information about the control. This property is read only.

AutoAFD

Type: Boolean

ODisCom only: this property specifies whether to enable or disable AFD. The program must be logged on to a valid desktop before AFD can be started. If your application starts AFD, you can cancel it only during the AFD initialization phase. You cannot cancel AFD before it receives the AFDInitBegin event or after it receives the AFDInitEnd event.

For example, to enable AFD:

```
MyODisCom.AutoAFD= TRUE
```

BaudRate

Type: integer

This property specifies the asynchronous connection baud rate; it is applicable only if your application is to connect with a Global Access server and CommType = 1. For normal connection to an ODAS server, the user establishes the telephone connection though an independent communications program.

Valid values of this enumerated data type are:

0	default
1	300
2	1200
3	2400
4	4800
5	9600
6	19,200
7	38,400

For example:

```
myDisCom.BaudRate = GetAsyncBaudRate%()
```

where GetAsyncBaudRate is a control in which the user chooses a baud rate.

CommPort

Type: integer

This property specifies which communications port is to be used for connection to the Global Access server; it is applicable only if your application is to connect with a Global Access server and CommType = 1.

Valid values of this enumerated data type are:

0 default

1 COM1

2 COM2

3 COM3

4 COM4

For example:

```
myDisCom.CommPort = GetPort%()
```

where GetPort is a control in which the user chooses a port.

CommType

Type: integer

This property specifies the type of connection to the ODAS or Global Access server. For a Global Access server, valid values are:

1 asynchronous connection

2 LAN connection

3 IDS LAN connection

For an ODAS server, the only valid value is 2.

For example:

```
myDisCom.CommType = GetCommType%()
```

Desktop

Type: string

This property specifies the name of the IDS desktop to be logged on to.

For example:

```
myDisCom.Desktop = txtDesktop.Text
```

ErrCode

Type: long

(For PowerBuilder only.) This property returns the error code after any operation.

For example:

```
myDisCom.Logon =TRUE
IfmyDisCom.ErrCode <>
Then ReportError()
EndIf
```

ExecAFD

Type: Boolean

ODisCom only, this property specifies whether to start a single instance of AFD. The program must be logged on to a valid desktop before AFD can be started. If your application starts AFD, you can cancel it only during the AFD initialization phase. You cannot cancel AFD before it receives the AFDInitBegin event or after it receives the AFDInitEnd event.

For example:

```
MyODisCom.ExecAFD = TRUE
```

Init

Type: Boolean

This property specifies whether the control should automatically initialize and terminate the session:

TRUE The control calls DisInitClient() to initialize the session as soon as the control connects to the server, and calls DisTermClient() to terminate the session when the control disconnects from the server.

FALSE The control does not initialize or terminate the session; your application must call DisInitClient() and DisTermClient().

For example:

```
myDisCom.Init =TRUE
```

Logon

Type: Boolean

This property specifies the connection state:

True The control logs on to the specified IDS desktop.

FALSE The control logs off the IDS desktop.

For example:

```
myDisCom.Init =TRUE
myDisCom.Desktop="user_name"
myDisCom.Password="password"
myDisCom.Logon=TRUE
```

Password
Type: string

This property specifies the user's IDS password. The user must enter this string; you cannot write it into the application.

For example:

```
myDisCom.Password = txtPassword.Text
```

PhoneNumber
Type: Boolean

This property specifies the telephone number to be used for connection to the ODAS server; it is applicable only if CommType = 1.

For example:

```
myDisCom.PhoneNumber = "(4 8)258-1412"
```

PowerBuilder
Type: Boolean

This property specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.

For example, to specify that you are not using PowerBuilder:

```
myDisCom.PowerBuilder= FALSE
```

ServerName
Type: string

This property specifies the name of the ODAS server with which the client is to connect; it is applicable only if CommType = 2.

For example:

```
myDisCom.ServerName = "Sep 14"
```

TimeOut
Type: integer

This property specifies the connection timeout in seconds; it is applicable only if CommType = 2.

For example:

```
myDisCom.TimeOut = 1
```

DisConv

The following sections provide information on DisConv limitations, errors, events, and properties.

Limitations

DisConv does not support batch mode.

DisConv does not support help.

DisConv cannot retrieve the default conversion table for the system.

Assumptions

DisConv assumes that the client and server are connected.

Error Codes

DisConv returns three types of error codes:

Codes received from the ODAS server, which are defined in the DISERR.TXT file.

Visual Basic codes.

Internal code as listed in the following table.

Name	Value	Description
DISCONV_ERR_BADEXTFORMAT	32000	The specified file extension has the wrong format. File extensions must be no more than three characters long.
DISCONV_ERR_BUFFERTOOSMALL	32001	The internal buffer is too small.
DISCONV_ERR_BADFILETYPE	32002	The specified file type is not a valid type.
DISCONV_ERR_BADOBJTYPE	32003	The specified IDS object type is not a valid type.
DISCONV_ERR_ILLEGALACTION	32004	The specified type of action is not a valid action value.
DISCONV_ERR_MEMALLOCFAILURE	32005	The computer does not have enough memory for DisConv to perform the requested action.
DISCONV_ERR_CREATEFILEFAILED	32006	Your attempt to create a conversion table failed.
DISCONV_ERR_OPENFILEFAILED	32007	Your attempt to open a conversion table failed.
DISCONV_ERR_READFILEFAILED	32008	Your attempt to read a conversion table failed.
DISCONV_ERR_WRITEFILEFAILED	32009	Your attempt to write a conversion table failed.
DISCONV_ERR_BADFILENAME	32010	The specified file name is not valid.

Methods

The Standard column identifies the events that are standard in Visual Basic; refer to the Microsoft Visual Basic documentation for their details and examples.

Name	Standard	Description
Clear	Yes	Clears all elements from the destination and source within the control.
Refresh	Yes	Refreshes the contents of DisConv and updates all information about open containers.
RemoveItem	Yes	Removes the selected entry from the control's list.

Events

The Standard column identifies the events that are standard in Visual Basic; refer to the Microsoft Visual Basic documentation for their details and examples.

Event	Parame-ters	Standard	Description
Update	none	No	(For PowerBuilder only.) Occurs whenever the applica-tion's internal property values change, to force PowerBuilder to update its own property values. Your program should ignore this event.

Properties

Some properties in DisConv are unique to this control. Other properties are standard in Visual Basic and the Descriptioncolumn identifies them as such; refer to the Microsoft Visual Basic documentation for details and examples.

Property	Data Type	Standard	Description
About	(none)	No	Displays a pop-up window that shows copyright and version information about DisConv and IBM.
Action	integer	No	Specifies whether the control is to reset, choose, read, or write a conversion table.
Count	integer	Yes	Specifies the number of entries in the conversion table.
ErrCode	long	No	(For PowerBuilder only.) Returns the error code after any operation.
FileExt	string	No	Specifies the local file extension of the file to be read or written.
FileName	string	No	Specifies the fully qualified path of the file. FileName is ignored if Action is neither 3 (reading) nor 4 (writing).
FileType	integer	No	Specifies the file-type number for the table entry. Type numbers are listed in *Developing Applications with Open Data Access Service*.

Property	Data Type	Standard	Description
Index	integer	Yes	Specifies a number that uniquely identifies the control in a control array. Valid values are from 0 through 32767.
Left	single	Yes	Specifies the distance between the internal left edge of the control and the left edge of the form it is in.
LineWidth	integer	No	For text files only; specifies the maximum width for lines in the destination file.
ListIndex	integer	Yes	Stores the index of the currently selected item in the list; that is, it identifies which item is selected.
Name	string	Yes	Specifies the name of the control.
ObjType	integer	No	Specifies the IDS object-type number for the table entry. Type numbers are listed in *Developing Applications with Open Data Access Service*.
Parent	form	Yes	Specifies which form is the parent of the control.
PowerBuilder	Boolean	No	Specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.
Tag	string	Yes	Stores any extra data needed by your application. Typically used to used this property to identify the code object.
Top	single	Yes	Specifies the distance between the internal top edge of the control and the top edge of the form. This value applies during the design phase only, because the control is invisible when the user runs your application.

Property Details

This section provides detailed descriptions of properties in DisConv that are unique to this control. For details and examples of the standard Visual Basic properties, refer to the Microsoft Visual Basic documentation.

About

This property displays a pop-up window that shows copyright and version information about DisConv. This property cannot be changed.

Action

Type: integer

This property specifies what the control is to do to a conversion table. Valid values are:

0 = Reset all of the table's entries to the default values. 1 = Specify the table to be used. 2 = Read the table from a file on the client computer. 3 = Write the table to a file on the client computer. Writing the table overwrites any previous contents.

For example:

```
myDisConv.FileName = "convert.bin"
myDisConv.Action= 1
```

This example reads a table to be converted.

ErrCode
Type: long

(For PowerBuilder only.) This property returns the error code after any operation.

For example:

```
If myDisConv.ErrCode <>
Then ReportErr()
EndIf
```

FileExt
Type: string

This property specifies the local file extension of the file to be read or written. (Write-only, during execution only)

For example:

```
myDisConv.FileExt= ".TXT"
```

FileName
Type: string

This property specifies the fully qualified path of the file. FileName is used only if Action value is 3 (reading) or 4 (writing).

For example:

```
myDisConv.FileName= "CONVERT.TBL"
```

FileType
Type: integer

This property specifies the file-type number for the table entry. Type numbers are listed in *Development with Open Data Access Service*. This property cannot be changed during capsule executions.

For example:

```
myDisConv.FileType= 2
```

LineWidth
Type: integer

For text files only; specifies the maximum width for lines in the destination PC file. (During execution only)

For example:

```
myDisConv.LineWidth= 1
```

ObjType

Type: integer

This property specifies the IDS object-type number for the table entry. Type numbers are listed in *Developing Applications with Open Data Access Service*. This property can only be changed during capsule execution.

For example:

```
myDisConv.ObjType= 5
```

PowerBuilder

Type: Boolean

This property specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.

For example, to specify that you are not using PowerBuilder:

```
myDisConv.PowerBuilder= FALSE
```

DisDir

The following sections provide information on DisDir limitations, errors, events, and properties.

Limitations

DisDir does not support batch mode.

DisDir does not support help.

Assumptions

If ViewMode isTRUE, DisDir assumes that the client and server are connected.

Error Codes

DisDir returns three types of error codes:

Codes received from the ODAS server, which are defined in the DISERR.TXT file.

Visual Basic codes.

Internal codes as listed in the following table.

Name	Value	Description
DISDIR_ERR_BADFORMAT	32000	The property value has the wrong format.
DISDIR_ERR_MEMALLOCFAILURE	32001	The computer does not have enough memory for DisDir to perform the requested action.
DISDIR_ERR_DELETEMODE	32002	The delete method can only be used when ViewMode is TRUE.
DISDIR_ERR_INVALID_INDEX	32003	Either the RemoveItem method requires an index argument, or the index provided is not valid.
DISDIR_ERR_DELETE_INDEX_OUTOFBOUND	32004	The index for the RemoveItem method is incorrect.
DISDIR_ERR_UNABLETODELETE_ITEM	32005	The specified item cannot be deleted.
DISDIR_ERR_DYNAMIC_MODE_CHANGE	32006	The ViewMode cannot be changed when PathActivate is TRUE.

Methods

The only method used by DisDir is standard in Visual Basic; refer to the Microsoft documentation for its details and example.

Name	Standard	Description
Refresh	Yes	Refreshes the contents of DisDir and updates all information about open containers.

Events

The Description column identifies the events that are standard in Visual Basic; refer to the Microsoft documentation for their details and examples.

Event	Parameters	Standard	Description
Click	Index As Integer	Yes	Occurs when the user clicks the mouse.
Close	Index As Integer	No	Occurs when the user tries to close an IDS container.
Completion	CompCode As Long	No	Occurs after a directory query is performed.
DblClick	Index As Integer	Yes	Occurs when the user double-clicks the mouse.
DragDrop	Index As Integer	Yes	Occurs when the user drags another control through this control and releases the mouse button.

Event	Parameters	Standard	Description
DragOver	Index As Integer, Source As Control, X As Single, Y As Single	Yes	Occurs when a drag-and-drop operation is in progress. With this event, you can monitor when the pointer enters, stays on, or leaves this control.
GotFocus	Index As Integer	Yes	Occurs when the focus is on this control.
KeyDown	Index As Integer, KeyCode As Integer, Shift As Integer	Yes	Occurs when the user presses any key while the focus is on this control.
KeyPress	Index As Integer, KeyAscii As Integer	Yes	Occurs when the user presses and releases any character key.
KeyUp	Index As Integer, KeyCode As Integer, Shift As Integer	Yes	Occurs when the user releases any key, while the focus is on this control.
LostFocus	Index As Integer	Yes	Occurs when the focus has been on this control but leaves.
MouseDown	Index As Integer, Button As Integer, Shift As Integer, X As Integer, Y As Integer	Yes	Occurs when the user presses a mouse button.
MouseMove	Index As Integer, Button As Integer, Shift As Integer, X As Integer, Y As Integer	Yes	Occurs when the user moves the mouse.
MouseUp	Index As Integer, Button As Integer, Shift As Integer, X As Integer, Y As Integer	Yes	Occurs when the user releases a mouse button.
Open	Index As Integer	No	Occurs when the user opens an IDS container.
Update	none	No	(For PowerBuilder only.) Occurs whenever the application's internal property values change, to force PowerBuilder to update its own property values. Your program should ignore this event.

Properties

Some properties in DisDir are unique to this control. Other properties are standard in Visual Basic and the Description column identifies them as such; refer to the Microsoft Visual Basic documentation for details and examples.

Property	Data Type	Standard	Description
About	(none)	No	Displays a pop-up window that shows copyright and version information about DisDir and IBM.
Align	integer	Yes	Specifies the list box's location in the form and how its size is set.
BackColor	long	Yes	Specifies the control's background color.
BorderStyle	integer	Yes	Defines the type of border, if any.
DragIcon	bitmap	Yes	Specifies what icon, if any, to display as the pointer in drag-and-drop operations.
DragMode	integer	Yes	Specifies whether to use manual (FALSE) or automatic (TRUE) dragging mode for drag-and-drop operations.
Enabled	Boolean	Yes	Specifies whether the control can (TRUE) or cannot (FALSE) respond to user-generated events.
ErrCode	long	No	(For PowerBuilder only.) Returns the error code after any operation.
FontBold	Boolean	Yes	Specifies whether the control's text is boldface (TRUE) or not (FALSE).
FontItalic	Boolean	Yes	Specifies whether the control's text is italic (TRUE) or roman (FALSE).
FontName	string	Yes	Specifies the font family used for the control's text.
FontSize	integer	Yes	Specifies the font size used for the control's text.
ForeColor	long	Yes	Specifies the control's foreground color.
Height	single	Yes	Specifies the control's height, in Microsoft twips, or 1/1440ths of an inch.
HelpContextID	integer	Yes	Specifies a context number to be associated with the control. The control uses this number to provide context-sensitive help for the application.
Index	integer	Yes	Specifies a number that uniquely identifies the control in a control array. Valid values are from 0 (zero) through 32767.
ItemInvert	Boolean	No	Inverts the background and foreground colors.

Property	Data Type	Standard	Description
Left	single	Yes	Specifies the distance between the internal left edge of the control and the left edge of the form it is in.
List	string	Yes	Specifies one object in the IDS container as an array element.
ListCount	integer	Yes	Stores the total number of items in the list box.
ListDate	long	No	Stores the date and time when the currently selected item was most recently modified.
ListIndex	integer	Yes	Stores the index of the currently selected item in the list; that is, it identifies which item is selected.
ListFid	long	No	Stores the IDS file number of the currently selected item in the list.
ListMetPath	string	No	Stores the MET@number identifier of the currently selected item in the list. This is a unique identifier, which is required because an IDS desktop can hold multiple items with the same name.
ListOpen	Boolean	No	Stores the state of the currently selected IDS desktop item in the list: open (TRUE) or closed (FALSE).
ListPath	string	No	Stores the path of the currently selected IDS desktop item in the list.
ListSize	long	No	Stores the size, in bytes, of the currently selected IDS desktop item in the list.
ListType	long	No	Stores the type number of the currently selected IDS desktop item in the list. Type numbers are listed in *Development Applications with Open Data Access Service*.
LocalObject	string	No	Specifies the full path of the local directory in which the IDS container's listing is to be stored; applicable only if ViewMode = 1.
Name	string	Yes	Specifies the name of the control.
Path	string	Yes	Specifies the path of the IDS container whose contents are to be listed.
PathActivation	Boolean	No	Specifies whether (TRUE) or not (FALSE) to query IDS container information when the Path property is set; applicable only if ViewMode = 0.
PictureClosedCapsule	bitmap	No	Specifies a graphic symbol to be displayed to represent a closed Capsule icon.
PictureClosedEnvelope	bitmap	No	Specifies a graphic symbol to be displayed to represent a closed Envelope icon.

Property	Data Type	Standard	Description
PictureClosedFolder	bitmap	No	Specifies a graphic symbol to be displayed to represent a closed Folder icon.
PictureLeaf	bitmap	No	Specifies a graphic symbol to be displayed to represent an IDS desktop item that is not a container.
PictureMinus	bitmap	No	Specifies a graphic symbol to be displayed to represent an open desktop item.
PicturePlus	bitmap	No	Specifies a graphic symbol to be displayed to represent a closed desktop item.
PictureOpenCapsule	bitmap	No	Specifies a graphic symbol to be displayed to represent an open Capsule icon.
PictureOpenEnvelope	bitmap	No	Specifies a graphic symbol to be displayed to represent an open Envelope icon.
PictureOpenFolder	bitmap	No	Specifies a graphic symbol to be displayed to represent an open Folder icon.
PowerBuilder	Boolean	No	Specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.
ProgTrigger	Boolean	No	Specifies whether to open the containers in a list when either the user double-clicks on it (FALSE) or through a command in the application (TRUE).
Sort	integer	No	Specifies the order in which the IDS container's list is to be sorted.
TabIndex	integer	Yes	Specifies the tabular order of the control within the form.
TabStop	Boolean	Yes	Specifies whether the user can use the **Tab** key to set the focus to this control.
Tag	string	Yes	Stores any extra data needed by your application. Typically used to identify the code object.
Top	single	Yes	Specifies the distance between the internal top edge of the control and the top edge of the form. This value is in effect during the design phase only because the control is invisible when the user runs your application.
ViewMode	integer	Yes	Specifies whether the IDS container's list is to be stored in a local file.
Visible	Boolean	Yes	Specifies whether the control is hidden or visible.
Width	single	Yes	Specifies the width of the control.

Property Details

This section provides detailed descriptions of properties in DisDir that are unique to this control. For details and examples of the standard Visual Basic properties, refer to the Microsoft documentation.

About

This property displays a pop-up window that shows copyright and version information about DisDir. This property cannot be changed.

ErrCode

Type: long

(For PowerBuilder only.) This property returns the error code after any operation.

For example:

```
If myDisDir.ErrCode <>
Then ReportError()
EndIf
```

ItemInvert

Type: Boolean

Set this property to TRUE if you want to invert the background and foreground colors.

For example:

```
myDisDir.ItemInvert = TRUE
```

ListDate

Type: long

This property stores the date and time when the currently selected item was most recently modified. This property cannot be changed during capsule execution.

For example:

```
date = myDisDir.ListDate(1)
```

ListFid

Type: long

This property stores the IDS file number of the currently selected item in the list. This property cannot be changed during capsule execution.

For example:

```
fid = myDisDir.ListFid(1)
```

ListMetPath
Type: string

This property stores the MET@number identifier of the currently selected item in the list. This is a unique identifier, which is required because an IDS desktop can hold multiple items with the same name. This property cannot be changed during capsule execution.

For example:

```
MetPath = myDisDir.ListMetPath(1)
```

ListOpen
Type: Boolean

This property stores the state of the currently selected IDS desktop item in the list: open (TRUE) or closed (FALSE). This property cannot be changed during capsule execution.

For example:

```
If (myDisDir.ListOpen (1)
)
Then status = "Open"
EndIf
```

ListPath
Type: string

This property stores the path of the currently selected IDS desktop item in the list. This property cannot be changed during capsule execution.

For example:

```
Path = myDisDir.ListPath(1)
```

ListSize
Type: long

This property stores the size, in bytes, of the currently selected IDS desktop item in the list. This property cannot be changed during capsule execution.

For example:

```
Size = myDisDir.ListSize(1)
```

ListType
Type: long

This property stores the type number of the currently selected IDS desktop item in the list. Type numbers are listed in *Developing Applications with Open Data Access Service*. This property cannot be changed during capsule execution.

1 asynchronous connection

2 LAN connection

3 IDS LAN connection

For example:

```
Type = myDisDir.ListType(1)
```

LocalObject

Type: string

This property specifies the full path of the local directory in which the IDS container's list is to be stored; it is applicable only if ViewMode = 1.

For example:

```
myDisDir.ViewMode = 1
myDisDir.LocalObject= "Dir.dat"
```

PathActivation

Type: Boolean

This property specifies whether (TRUE) or not (FALSE) to query IDS container information when the Path property is set; it is applicable only if ViewMode = 0.

PathActivation exists primarily for backward compatibility with the previous version of DisDir control. Use the Action property to trigger this control.

For example:

```
myDisDir.ViewMode = 1
myDisDir.PathActivation = FALSE
```

PictureClosedCapsule

Type: bitmap

This property specifies a graphic symbol to be displayed to represent a closed Capsule icon.

For example:

```
myDisDir.PictureClosedCapsule =pic_closed_cap
```

PictureClosedEnvelope

Type: bitmap

This property specifies a graphic symbol to be displayed to represent a closed Envelope icon.

For example:

```
myDisDir.PictureClosedEnvelope = pic_closed_env
```

PictureClosedFolder
Type: bitmap

This property specifies a graphic symbol to be displayed to represent a closed Folder icon.

For example:

```
myDisDir.PictureClosedFolder = pic_closed_fldr
```

PictureLeaf
Type: bitmap

This property specifies a graphic symbol to be displayed to represent a IDS desktop item that is not a container.

For example:

```
myDisDir.PictureLeaf = picleaf
```

PictureMinus
Type: bitmap

This property specifies a graphic symbol to be displayed to represent an open desktop item.

For example:

```
myDisDir.PictureMinus = picminus
```

PicturePlus
Type: bitmap

This property specifies a graphic symbol to be displayed to represent a closed desktop item.

For example:

```
myDisDir.PicturePlus = picplus
```

PictureOpenCapsule
Type: bitmap

This property specifies a graphic symbol to be displayed to represent an open Capsule icon.

For example:

```
myDisDir.PictureOpenCapsule = pic_op_cap
```

PictureOpenEnvelope

Type: bitmap

This property specifies a graphic symbol to be displayed to represent an open Envelope icon.

For example:

```
myDisDir.PictureOpenEnvelope = pic_op_env
```

PictureOpenFolder

Type: bitmap

This property specifies a graphic symbol to be displayed to represent an open Folder icon.

For example:

```
myDisDir.PictureOpenFolder =
pic_op_fldr
```

PowerBuilder

Type: Boolean

This property specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.

For example, to specify that you are not using PowerBuilder:

```
myDisDir.PowerBuilder = FALSE
```

ProgTrigger

Type: Boolean

This property specifies how to open containers in the list, by user action of through a command in the application:

TRUE Control opening through the application.

FALSE Automatically open any container when the user double-clicks on it.

For example:

```
myDisDir.ProgTrigger = TRUE
```

Sort
Type: integer

This property specifies the order in which the IDS container's listing is to be sorted:

0 unsorted

1 ascending order by file name

2 ascending order by file type

3 ascending order by file size

4 ascending order by the date of the file in the container

5 descending order by file name

6 descending order by file type

7 descending order by file size

8 descending order by the date of the file in the container

For example:

```
myDisDir.Sort =
```

ViewMode
Type: integer

This property specifies how the IDS container's listing is to be treated:

0 retrieved and immediately displayed

1 retrieved through the DisXfr or ODisXfr controls and stored as specified in the LocalObjectproperty

For example:

```
myDisDir.ViewMode = 1
```

DisSetVs

The following sections provide information on DisSetVs limitations, errors, events, and properties.

Limitations

DisSetVs does not support batch mode.

DisSetVs does not support help.

Assumptions

DisSetVs assumes that the client and server are connected.

Error Codes

DisSetVs returns three types of error codes:

Codes received from the ODAS server, which are defined in the DISERR.TXT file.

Visual Basic codes.

Internal codes as listed in the following table.

Name	Value	Description
DISSETVS_ERR_OUTOFMEMORY	32000	The computer does not have enough memory for DisSetVs to perform the requested action.
DISSETVS_ERR_ILLEGALATVAR	32001	The @-variable specified in AtVar is not in the acceptable range.
DISSETVS_ERR_ARRAYOUTOFBOUND	32002	The index of the requested list element is greater than the actual size of the list.
DISSETVS_ERR_DESCTOOLARGE	32003	The specified @-variable description exceeded the maximum size (50 characters).
DISSETVS_ERR_VALUETOOLARGE	32004	The specified @-variable value string exceeded the maximum size (255 characters).
DISSETVS_ERR_NEEDVARTOSTORE	32005	You tried to write a value or description without specifying its @-variable.
DISSETVS_ERR_INVALIDACTION	32006	The specified type of action is not a valid action value.
DISSETVS_ERR_CAPSULENAMENEEDED	32007	You tried to perform some action without specifying the Capsule application's name.

Methods

The only method used by DisSetVs is standard in Visual Basic; refer to the Microsoft documentation for its details and example.

Name	Standard	Description
Clear	No	Clears all elements from the destination and source within the control.

Events

The Standard column identifies the events that are standard in Visual Basic; refer to the Microsoft Visual Basic documentation for details and examples.

Event	Parameters	Standard	Description
Update	None	No	(For Power Builder only.) Occurs whenever the application's internal property values change, to force Power Builder to update its own property values. Your program should ignore this event.

Properties

Some properties in DisSetVs are unique to this control. Other properties are standard in Visual Basic and the Description column identifies them as such; refer to the Microsoft Visual Basic documentation for details and examples.

Property	Data Type	Standard	Description
About	(none)	No	Displays a pop-up window that shows copyright and version information about DisSetVs and IBM.
Action	integer	No	Specifies whether the control is to retrieve or save @-variables.
AtVar	string	No	Specifies which @-variable the Value and Description properties belong to.
CapsuleName	string	No	Specifies which Capsule application the @-variables belong to.
Description	string	No	Specifies a descriptive label for an @-variable.
ErrCode	long	No	(For Power Builder only.) Returns the error code after any operation.
Index	integer	Yes	Specifies a number that uniquely identifies the control in a control array. Valid values are from 0 (zero) through 32,767.
Left	single	Yes	Specifies the distance between the internal left edge of the control and the left edge of the form it is in.
List	array string	No	Specifies an array of defined @-variables. To delete an @-variable from the list, set its Value and Description properties to empty strings (" ").
ListCount	integer	No	Specifies the number of @-variables in the list.
Name	string	Yes	Specifies the name of the control.
PowerBuilder	Boolean	No	Specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.
Replace	Boolean	No	Specifies whether (TRUE) or not (FALSE) to overwrite the @-variable.
Tag	string	Yes	Stores any extra data needed by your application. Typically used to identify the code object.

Property	Data Type	Standard	Description
Top	single	Yes	Specifies the distance between the internal top edge of the control and the top edge of the form. This value is in effect during the design phase only, because the control is invisible when the user runs your application.
Value	string	No	Specifies the value for an @-variable.

Property Details

This section provides detailed descriptions of properties in DisSetVs that are unique to this control. For details and examples of the standard Visual Basic properties, refer to the Microsoft documentation.

About

This property displays a pop-up window that shows copyright and version information about DisSetVs. This property cannot be changed.

Action

Type: integer

This property specifies whether the control is to retrieve or save @-variables. Valid values are:

0 Retrieve the current values and descriptions from the specified Capsule application.

1 Save the specified values and descriptions to the specified Capsule application.

For example:

```
myDisSetAtVars.Action =
```

AtVar

Type: string

This property specifies to which @-variable the Value and Description properties belong.

For example:

```
myDisSetAtVars.AtVar = "A[rdqu]
myDisSetAtVars.Value= "2 "
myDisSetAtVars.Description= "Price"
```

CapsuleName

Type: string

This property specifies which Capsule application the @-variables belong to.

For example:

```
myDisSetAtVars.CapsuleName= "Capsule"
```

Description
Type: string

This property specifies a descriptive label for an @-variable.

For example:

```
myDisSetAtVar = "A"
myDisSetAtVars.Description = "Price"
```

ErrCode
Type: long

(For PowerBuilder only.) This property returns the error code after any operation.

For example:

```
myDisSetAtVars.Action =
If ErrCode <>
Then PrintErr()
EndIf
```

List
Type: array string

This property specifies an array of defined @-variables. To delete an @-variable from the list, set its Value and Description properties to empty strings (" ").

If you defined variable @A, then atVarString would be A. For example:

```
atVarString= myDisSetAtVars.List( )
```

ListCount
Type: integer

This property specifies the number of @-variables in the list. (Read-only, during execution)

For example:

```
Count = myDisSetAtVars.ListCount
```

PowerBuilder
Type: Boolean

This property specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.

For example, to specify that you are not using PowerBuilder:

```
myDisSetAtVars.PowerBuilder= FALSE
```

Replace

Type: Boolean

This property specifies whether to overwrite the @-variable. Specify the DIS_OVERWRITE flag to replace the Capsule application's current variables instead of merging the new and old lists of variables. For example, assume that the existing list has @A and @B and pcapv specifies only @B. With DIS_OVERWRITE, the resulting variable list has only @B. Without DIS_OVERWRITE, the resulting variable list has the old @A and the new @B.

TRUE Overwrite the @-variable specified in AtVar with the specified value and description.

FALSE Do not overwrite the @-variable specified in AtVar.

For example:

```
myDisSetAtVars.Replace= TRUE
```

Value

Type: string

This property specifies the value for the @-variable specified in AtVar.

For example:

```
myDisSetAtVars.Value = OSCO
```

DisSubCp

The following sections provide information on DisSubCp limitations, errors, events, and properties.

Limitations

DisSubCp does not support batch mode.

DisSubCp does not support help.

Assumptions

DisSubCp assumes that the client and server are connected.

Error Codes

DisSubCp returns three types of error codes:

Codes received from the ODAS server, which are defined in the DISERR.TXT file.

Visual Basic codes.

Internal codes, as listed here:

Name	Value	Description
DISSUBCAP_ERR_OUTOFMEMORY	32000	The computer does not have enough memory for DisSubCp to perform the requested action.
DISSUBCAP_ERR_INVALIDDATETIMEFORMAT	32001	The date or time specified in the ScheduledTime property is in the wrong format.
DISSUBCAP_ERR_BADCAPNAME	32002	The CapsuleName property is either empty or unspecified.
DISSUBCAP_ERR_BADDESTINATION	32003	The DestContainer property is either empty or specified with an incorrect format.

Events

The Standard column identifies the events that are standard in Visual Basic; refer to the Microsoft Visual Basic documentation for details and examples.

Event	Parameters	Standard	Description
Update	None	No	(For PowerBuilder only.) Occurs whenever the application's internal property values change, to force PowerBuilder to update its own property values. Your program should ignore this event.

Properties

Some properties in DisSubCp are unique to this control; detailed descriptions follow below. Other properties are standard in Visual Basic and the Description column identifies them as such; refer to the Microsoft Visual Basic documentation for details and examples.

Property	Data Type	Standard	Description
About	(none)	No	Displays a pop-up window that shows copyright and version information about DisSubCp and IBM.

Property	Data Type	Standard	Description
CapsuleName	string	No	Specifies the fully qualified path of the Capsule application that is to be submitted for execution. Set all other job-control properties before you set this property.
DestContainer	string	No	Specifies the fully qualified path of the container that is to receive the executed copy of the Capsule application.
Description	string	No	Describes the Capsule job.
ErrCode	long	No	(For PowerBuilder only.) Returns the error code after any operation.
Index	integer	Yes	Specifies a number that uniquely identifies the control in a control array. Valid values are from 0 (zero) through 32767.
Left	single	Yes	Specifies the distance between the internal left edge of the control and the left edge of the form it is in.
JobId	long	No	Stores the job number, provided by the Capsule Service.
Name	string	Yes	Specifies the name of the control.
MaxStartDelayTime	long	No	Specifies the maximum allowable delay past the scheduled starting time, in seconds.
MaxExecuteTime	long	No	Specifies the maximum allowable execution time, in seconds.
PowerBuilder	Boolean	No	Specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.
Priority	integer	No	Specifies the priority of the submitted Capsule application.
ScheduledAlias	string	No	Specifies the predefined alias for scheduling the job. See the Capsule Services chapter of the *System Administration Guide and Reference* for discussion of aliases.
ScheduledTime	string	No	Specifies the scheduled starting time.
Tag	string	Yes	Stores any extra data needed by your application. Typically used to identify the code object.
Top	single	Yes	Specifies the distance between the internal top edge of the control and the top edge of the form. This value applies during the design phase only, because the control is invisible when the user runs your application.

Property Details

This section provides detailed descriptions of properties in DisSubCp that are unique to this control. For details and examples of the standard Visual Basic properties, refer to the Microsoft Visual Basic documentation.

About

This property displays a pop-up window that shows copyright and version information about DisSubCp. This property cannot be changed.

CapsuleName

Type: string

This property specifies the IDS path of the Capsule application that is to be submitted for execution. The path is not relative; it must be fully qualified.

Set all other job-control properties before you set this property. As soon as you set a value for CapsuleName, the application submits the capsule.

For example:

```
DisSubCp.CapsuleName = "\Cap\Capsule"
```

This example submits a Capsule in the DisDir control listing.

DestContainer

Type: string

This property specifies the IDS path of the container that is to receive the executed copy of the Capsule application. The path is not relative; it must be fully qualified.

For example:

```
MySubmitCapsule.DestContainer = "\"
```

Description

Type: string

This property describes the Capsule job.

For example:

```
curIndex = frmCapQ.DisCapQ.ListIndex
DisSubCpDescription = "Report for manager"
```

ErrCode

Type: long

(For PowerBuilder only.) This property returns the error code after any operation.

For example:

```
If DisCapQ.ErrCode <>
Then PrintErr()
EndIf
```

JobId
Type: long

This property stores the job number, provided by capsule service, in the execution queue. This property cannot be changed during capsule execution.

For example:

```
pnlStatus = "Job
submitted. Job ID:" & Str$(MySubmitCapsule.JobId)
MaxStartDelayTime
```

MaxStartDelayTime
Type: long

This property specifies the maximum allowable delay past the scheduled starting time, in seconds.

For example:

```
DisSubCp.MaxStartDelayTime =Val(DelayTime)
```

MaxExecuteTime
Type: long

This property specifies the maximum allowable execution time, in seconds.

For example:

```
MySubmitCapsule.MaxExecuteTime = Val(MaxExecTime)
```

PowerBuilder
Type: Boolean

This property specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.

For example, to specify that you are not using PowerBuilder:

```
MySubmitCapsule.PowerBuilder= FALSE
```

Priority
Type: integer

This property specifies the priority of the submitted Capsule application, within the execution queue. The lowest priority is 1; the highest is 31.

For example:

```
MySubmitCapsule.Priority = 1
```

ScheduledAlias

Type: string

This property specifies the predefined alias for scheduling the job. The administrator of the capsule service can establish aliases for single or periodic execution at any time, and day, or date. See the *System Administration Guide and Reference* for discussion of aliases. Each capsule job submitted submission should use either ScheduledAlias or ScheduledTime. You cannot specify both.

For example:

```
DisSubCp.ScheduledAlias ="OVERNIGHT"
```

ScheduledTime

Type: string

This property specifies the scheduled starting time for the job. Each capsule jub submitted should use either ScheduledAlias or ScheduledTime. You cannot specify both. The format must be:

mm/dd/yy hh:mm:ss

where:

mm	the month of the year (1 through 12)
dd	the day of the month (1 through 31)
yy	the last two digits of the year (94 through 99)
hh	the hour of the 24-hour day (0 through 23)
mm	the minute (0 through 59)
ss	the second (0 through 59)

For example:

```
DisSubCp.ScheduledTime = " 6/ 5/94  :  :  "
```

DisTxfer and ODisTxfr

The following sections provide information on DisTxfer and ODisTxfr limitations, errors, events, and properties.

Limitations

These controls do not support help.

All files are transferred sequentially. If an error occurs during transfer of one file, the operation ends and no more files are transferred.

If you use wildcards (asterisks) to transfer IDS files to the client computer, the IDS file name must meet the limitations of MS-DOS file names.

Assumptions

DisTxfer assumes that the client and server are connected.

Error Codes

The control returns three types of error codes:

Codes received from the ODAS server, which are defined in the DISERR.TXT file.

Visual Basic codes.

Internal codes as listed in the following table.

Name	Value	Description
DISTXFR_ERR_OUTOFMEMORY	32000	The computer does not have enough memory for DisTxfr to perform the requested action.
DISTXFR_ERR_ARRAYOUTOFBOUND	32001	The index of the requested list element is two or more greater than the actual size of the list.
(none)	32002	Reserved.
DISTXFR_ERR_UNDEFINEDMODE	32003	The Mode property was specified as something other than 0 or 1.
DISTXFR_ERR_NOTHING2DO	32004	No element is set in the Destination or Source property, so DisTxfr cannot transfer anything.
DISTXFR_ERR_OBJTYPE4GETMODE	32005	The Objtype or FileType property is set, but Mode is set to 0 (for retrieval from IDS).
DISTXFR_ERR_INVALIDWIDTH	32006	The line width exceeded the maximum, 160 characters.
DISTXFR_ERR_NODESTNATION	32007	You must set the Destination or DestDir property.
DISTXFR_ERR_NOSOURCE	32008	You must set the Source or SourceDir property.
DISTXFR_ERR_RENAME	32009	An error occurred during transfer of IDS files to the local computer with wildcard (asterisks). One cause is an IDS name that does not meet the limitations of MS-DOS file names.
DISTXFR_ERR_RENAMEEXISTEDFILE	32010	DisTxfr could not rename a downloaded file because the proposed file name already exists.

Name	Value	Description
DISTXFR_ERR_RENAMEFILENOTFOUND	32011	DisTxfr could not rename a downloaded file because it could not find the file.

Methods

All of the methods used by DisTxfer are standard in Visual Basic; refer to the Microsoft documentation for their details and examples.

Name	Standard	Description
Clear	Yes	Clears all elements from the destination and source within the control.
RemoveItem	Yes	Removes the selected entry from the control's list.

Events

The Standard column identifies the events that are standard in Visual Basic; refer to the Microsoft Visual Basic documentation for details and examples.

Event	Parameters	Standard	Description
FileBegin	FileSize As Long, FileName As String	No	ODisTxfr only: occurs when file transfer begins.
FileEnd	ErrorCode As Long, SourceName As String	No	ODisTxfr only: occurs when file transfer finishes.
FileProgress	ElapsedTime As Long, XferSize As Long, FileSize As Long	No	With ODisTxfr only: occurs during file transfer.
Update	None	No	(For PowerBuilder only.) Occurs whenever the application's internal property values change, to force PowerBuilder to update its own property values. Your program should ignore this event.

Event Details

This section provides detailed descriptions of events in DisTxfer and ODisTxfr that are unique to these controls. For details and examples of the standard Visual Basic events, refer to the Microsoft Visual Basic documentation.

FileBegin

ODisTxfer only: this event occurs when automated file distribution begins transferring a file.

The syntax is:

Sub *ODisTxfer*_FileBegin (*FileSize*As Long, *FileName* As String)

where:

FileSize	Specifies the size, in bytes, of the file being transferred.
FileName	Specifies the name of the file being transferred.

FileEnd

ODisTxfer only: this event occurs when automated file distribution finishes transferring a file.

The syntax is:

Sub *ODisTxfer*_FileEnd (*ErrorCode*As Long, *SourceName* As String)

where:

ErrorCode	Returns the error code from this process.
SourceName	Specifies the name of the file being transferred.

FileProgress

With ODisTxfer only, this event occurs while automated file distribution is transferring a file.

The syntax is:Sub *ODisTxfer*_FileProgress (*ElapsedTime*As Long, *XferSize*As Long, *FileSize*As Long)

where:

ElapsedTime	Returns the elapsed time, in seconds, since the file transfer started.
TransferSize	Returns the number bytes that were transferred.
FileSize	Specifies the size, in bytes, of the file being transferred.

Update

This event exists for PowerBuilder only. It occurs whenever the application's internal property values change, to force PowerBuilder to update its own property values. Your program should ignore this event.

Properties

Some properties in DisTxfer are unique to this control; detailed descriptions follow below. Other properties are standard in Visual Basic and the Description column identifies them as such; refer to the Microsoft Visual Basic documentation for details and examples.

Property	Data Type	Standard	Description
About	(none)	No	Displays a pop-up window that shows copyright and version information about the controland IBM.
Action	integer	Yes	Triggers some action, according to the Mode setting.
Cancel	Boolean	No	With ODisTxfr only, stops (TRUE) or does not stop (FALSE) the transfer.
DestDir	string	No	Specifies the directory or IDS container where the file copies are to go.
Destination	string	No	Specifies the destination name (for transfer in either direction). Whether it is an IDS object name or a file name, the path can be either fully qualified or relative to the DestDirproperty. If a full path is specified, then DestDir is ignored. You can specify the full path without specifying DestDir.
ErrCode	long	No	(For PowerBuilder only.) Returns the error code after any operation.
FileType	integer	No	Specifies the type number of the file to be transferred to the IDS desktop. Type numbers are listed in *Developing Applications with Open Data Access Service*.
Index	integer	Yes	Specifies a number that uniquely identifies the control in a control array. Valid values are from 0 (zero) through 32767.
Left	single	Yes	Specifies the distance between the internal left edge of the control and the left edge of the form it is in.
LineWidth	integer	No	For text files only; specifies the maximum width for lines in the destination file.
ListCount	integer	No	Specifies the number of destination and source files to be listed. The maximum depends on how much memory is available.
ListErrorIndex	integer	No	Specifies the index, within the DisTxfer control, of the destination and source file that have caused an error.
ListIndex	integer	No	Specifies the index, within the DisTxfer control, of the currently selected destination and source files.
Mode	integer	No	Specifies the direction of the file transfer: 0 = from the IDS desktop to the client 1 = from the client to the IDS desktop
Name	string	Yes	Specifies the name of the control.
ObjType	integer	No	Specifies the IDS object-type number when a file is transferred from the client computer to the IDS desktop. Type numbers are listed in the *Developing Applications with Open Data Access Service* manual.

Property	Data Type	Standard	Description
PowerBuilder	Boolean	No	Specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.
Source	string	No	Specifies the name of the file to be transferred (in either direction). Whether it is an IDS object name or a file name, the path can be either fully qualified or relative to the SourceDir property. If a full path is specified, then SourceDir is ignored.
SourceDir	string	No	Specifies the directory or IDS container from which the file copies are to be transferred. SourceDir is ignored if Source is fully qualified.
Tag	string	Yes	Stores any extra data needed by your application. Developers usually use this property to identify the code object.
Top	single	Yes	Specifies the distance between the internal top edge of the control and the top edge of the form. This value applies during the design phase only, because the control is invisible when the user runs your application.

Property Details

This section provides detailed descriptions of those properties in DisTxfer that are unique to this control. For details and examples of the standard Visual Basic properties, refer to the Microsoft Visual Basic documentation.

About

This property displays a pop-up window that shows copyright and version information about DisTxfer. This property cannot be changed.

Cancel

Type: Boolean

With ODisTxfr only, this property can stop a transfer that is in progress.

TRUE Stop transferring files.

FALSE Do not interrupt the transfer.

For example:

```
MyODisTxfr.Cancel = TRUE
```

DestDir

Type: string

This property specifies the directory or IDS container where the file copies are to be stored. DestDir is ignored if Destination is fully qualified.

For example:

```
myDisTxfer.DestDir = "c:\"
```

Destination
Type: string

This property specifies the destination name (for transfer in either direction). Whether it is an IDS object name or a file name, the path can be either fully qualified or relative to the DestDir property. If a full path is specified, then DestDir is ignored. This property cannot be changed during capsule execution.

For example:

```
myDisTxfer.Destination = "dest.doc"
```

ErrCode
Type: long

(For PowerBuilder only.) This property returns the error code after any operation.

For example:

```
If DisTxfr.ErrCode <>
Then ReportErr()
EndIf
```

FileType
Type: integer

This property specifies the type number of the file to be transferred to the IDS desktop. Type numbers are listed in *Developing Applications with Open Data Access Service*. This property cannot be changed during capsule execution.

For example:

```
myDisTxfer.FileType = 2
```

LineWidth
Type: integer

For text files only, this property specifies the maximum width for lines in the destination file. (During execution only)

For example:

```
myDisTxfer.LineWidth = 1
```

ListCount
Type: integer

This property specifies the number of destination and source files to be listed. The maximum depends on how much memory is available. (Read-only, during execution)

For example:

```
count = myDisTxfer.ListCount
```

ListErrorIndex

Type: integer

This property specifies the index, within the DisTxfer control, of the destination and source file that caused an error. (Read-only, during execution)

For example:

```
indexErr = myDisTxfer.ListErrorIndex
```

ListIndex

Type: integer

This property specifies the index, within the DisTxfer control, of the currently selected destination and source files. (Read-only, during execution)

For example:

```
index = myDisTxfer.ListIndex
```

Mode

Type: integer

This property specifies the direction of the file transfer. Valid values are:

0 from the IDS desktop to the client computer

1 from the client computer to the IDS desktop

For example:

```
myDisTxfer.Mode =
```

ObjType

Type: integer

This property specifies the IDS object-type number when a file is transferred from the client computer to the IDS desktop. Type numbers are listed in the *Developing Applications with Open Data Access Service*. This property can only be changed during capsule execution.

For example:

```
myDisTxfer.ObjType = 1
```

PowerBuilder
Type: Boolean

This property specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.

For example, to specify that you are not using PowerBuilder:

```
myDisTxfer.PowerBuilder = FALSE
```

Source
Type: string

This property specifies the name of the file to be transferred (in either direction). Whether it is an IDS object name or a file name, the path can be either fully qualified or relative to the SourceDir property. If a full path is specified, then SourceDir is ignored.

For example:

```
myDisTxfer.Source = "envelope"
```

SourceDir
Type: string

This property specifies the directory or IDS container from which the file copies are to be transferred. SourceDir is ignored if Source is fully qualified.

For example:

```
myDisTxfer.SourceDir = 1
```

DisXfr and ODisXfr

The following sections provide information on DisXfr and ODisXfr limitations, errors, events, and properties.

Limitations

The control does not support batch mode.

The control does not support help.

Assumptions

DisXfr assumes that the client and server are connected.

Error Codes

The control returns three types of error codes:

Codes received from the ODAS server, which are defined in the DISERR.TXT file.

Visual Basic codes.

Internal codes as listed in the following table.

Name	Value	Description
DISTXFR_ERR_OUTOFMEMORY	32000	The computer does not have enough memory for the control to perform the requested action.
DISTXFR_ERR_ARRAYOUTOFBOUND	32001	The index of the requested list element is two or more greater than the actual size of the list.
(none)	32002	Reserved.
DISTXFR_ERR_UNDEFINEDMODE	32003	The Mode property was specified as something other than 0 or 1.
DISTXFR_ERR_NOTHING2DO	32004	No element is set in the Destination or Source property, so the controlcannot transfer anything.
DISTXFR_ERR_OBJTYPE4GETMODE	32005	The Objtype or FileType property is set, but Mode is set to 0 (for retrieval from IDS).
DISTXFR_ERR_INVALIDWIDTH	32006	The line width exceeded the maximum, 160.
DISTXFR_ERR_NODESTNATION	32007	You must set the Destination or DestDir property.
DISTXFR_ERR_NOSOURCE	32008	You must set the Source or SourceDir property.
DISTXFR_ERR_RENAME	32009	An error occurred during transfer of IDS files to the local computer with wildcards. One possible cause is an IDS name that does not meet the limitations of MS-DOS file names.
DISTXFR_ERR_RENAMEEXISTEDFILE	32010	The control could not rename a downloaded file because the proposed file name already exists.
DISTXFR_ERR_RENAMEFILENOTFOUND	32011	The control could not rename a downloaded file because it could not find the file.
DISXFR_ERR_DESTINATIONNOTEXIST	32014	The control could not find the specified destination directory.
DISXFR_ERR_DESTINATIONNOTDIR	32015	The control found the specified destination name, but it is not a directory.
DISXFR_ERR_DATETIMEFORMAT	32016	The date or time was specified in an invalid format.
DISXFR_ERR_CONVERTFILE	32017	The control could not find the specified conversion table.
DISCACHE_ERR_OPENARCHIVE_FAILED	42000	The control could not open the object-information file.

Name	Value	Description
DISCACHE_ERR_CREATEARCHIVEFAILED	42001	The control could not create the object-information file.
DISCACHE_ERR_READARCHIVEFAILED	42002	The control opened but could not read from the object-information file.
DISCACHE_ERR_WRITEARCHIVEFAILED	42003	The control opened but could not write to the object-information file.
DISCACHE_ERR_MEMORYEXCEPTION	42004	The action required more memory than was available.
DISCACHE_ERR_INVALID_MODE	42005	The specified mode is invalid.
DISCACHE_ERR_NONEXISTING_DIRECTORY	42006	The controlcould not find the specified directory.
DISCACHE_ERR_NOT_A_DIRECTORY	42007	The controlfound the specified directory name, but it is not a directory.
DISCACHE_ERR_SHARE_VIOLATED	42008	Another application has the destination file opened for write mode.
DISCACHE_ERR_INVALID_POINTER	42009	An internal error occurred.
DISCACHE_ERR_FULLPATHNAME_NEEDED	42010	The source property must be fully qualified.
DISCACHE_ERR_INUSE	42011	The destination directory is being used by another application.
DISCACHE_ERR_OPENEDALREADY	42012	The control tried to open a file that was already open.
DISCACHE_ERR_LMMFILE_NOTOPENED	42013	The local object cannot be located.
DISCACHE_ERR_CLOSE_UNOPENDFILE	42014	An internal error occurred.
DISCACHE_ERR_REFRESH	42015	An error occurred while updating information for a container.
DISCACHE_ERR_STALE_FILE	42016	The local file has been modified since it was last opened.
DISCACHE_ERR_FOLDERNOTFOUND	42017	An internal error occurred while reading an IDS container.
DISCACHE_ERR_EXCLUSIVESAVE	42018	An error occurred while saving the downloaded IDS objects.
DISCACHE_ERR_NOTSUPPORTED	42019	An internal error occurred.

Events

The Standard column identifies the events that are standard in Visual Basic; refer to the Microsoft Visual Basic documentation for their details and examples.

Event	Parameters	Standard	Description
Error	ErrorCodeAs Long, SourceNameAs String	No	Occurs when an error happens during transfer.
FileBegin	FileSizeAs Long, FileName As String	No	ODisXfr only: occurs when file transfer begins.
FileCount	FileCountAs Long	No	Specifies the number of files to be transferred.
FileEnd	ErrorCodeAs Long, SourceName As String	No	ODisXfr only: occurs when file transfer finishes.
FileProgress	ElapsedTimeAs Long, XferSizeAs Long, FileSizeAs Long	No	ODisXfronly: occurs during file transfer.
Update	none	No	(For PowerBuilder only.) Occurs whenever the application's internal property values change, to force PowerBuilder to update its own property values. Your program should ignore this event.

Event Details

This section provides detailed descriptions of events in DisXfr and ODisXfr that are unique to these controls. For details and examples of the standard Visual Basic events, refer to the Microsoft Visual Basic documentation.

Error

This event occurs when there is an error during transfer.

The syntax is:

Sub *DisXfr*_Error (*ErrorCode*As Long, *SourceName*As String)

where:

ErrorCode	Specifies the code number of the error that occurred. See “Error Codes” on page 281.
SourceName	Specifies the name of the file being transferred.

FileBegin

ODisXfr only: this event occurs when automated file distribution begins transferring a file.

The syntax is:

Sub *ODisXfr*_FileBegin (*FileSize*As Long, *FileName* As String)

where:

FileSize	Specifies the size, in bytes, of the file being transferred.
FileName	Specifies the name of the file being transferred.

FileCount

This event occurs when automated file distribution begins transferring a file.

The syntax is:

Sub *ODisXfr*_FileCount (*FileCount*As Long)

where:

FileCount	Specifies the number of files or objects being transferred.

FileEnd

ODisXfr only: this event occurs when automated file distribution finishes transferring a file.

The syntax is:

Sub *ODisXfr*_FileEnd (*ErrorCode*As Long, *SourceName* As String)

where:

ErrorCode	Returns the error code from this process.
SourceName	Specifies the name of the file being transferred.

FileProgress

ODisXfr only: this event occurs while automated file distribution is transferring a file.

The syntax is:

Sub *ODisXfr*_FileProgress (*ElapsedTime*As Long, *XferSize*As Long, *FileSize*As Long)

where:

ElapsedTime	Returns the elapsed time, in seconds, since the file transfer started.
TransferSize	Returns the number bytes that were transferred.
FileSize	Specifies the size, in bytes, of the file being transferred.

Update

This event exists for PowerBuilder only. It occurs whenever the application's internal property values change, to force PowerBuilder to update its own property values. Your program should ignore this event.

Properties

Some properties in DisXfr are unique to this control; detailed descriptions follow below. Other properties are standard in Visual Basic and the Description column identifies them as such; refer to the Microsoft documentation for their details and examples.

Property	Data Type	Standard	Description
About	(none)	No	Displays a pop-up window that shows copyright and version information about the control and IBM.
Cancel	Boolean	No	With ODisXfr only, stops (TRUE) or does not stop (FALSE) the transfer.
ConvertTbl	string	No	Specifies the full path name of the conversion table to be used during the transfer.
Date	string	No	Specifies a date in the format MM/DD/YY limiting which files or objects are to be transferred. If you do not specify a date, everything is transferred.
Destination	string	No	Specifies the destination name (for transfer in either direction). Whether it is an IDS object name or a PC file name, the path must be fully qualified.
ErrCode	long	No	(For PowerBuilder only.) Returns the error code after any operation.
Index	integer	Yes	Specifies a number that uniquely identifies the control in a control array. Valid values are from 0 (zero) through 32767.
Left	single	Yes	Specifies the distance between the internal left edge of the control and the left edge of the form it is in.
Mode	integer	No	Specifies the direction of the file transfer: 0 = from the IDS desktop to the client 1 = from the client to the IDS desktop
Name	string	Yes	Specifies the name of the control.
Parent	form	Yes	Specifies which form is the parent of the control.
PowerBuilder	Boolean	Yes	Specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.
Purge	Boolean	No	Specifies whether (TRUE) or not (FALSE) to delete the source object after it is transferred.
Recursive	Boolean	No	Specifies whether to copy all the files within the container, including any nested containers and their contents (TRUE), or to merely copy the files in the container itself (FALSE).

Property	Data Type	Standard	Description
Source	string	No	Specifies the name of the file to be transferred (in either direction). Whether it is an IDS object name or a PC file name, the path must be fully qualified.
Tag	string	Yes	Stores any extra data needed by your application. Developers usually use this property to identify the code object.
Top	single	Yes	Specifies the distance between the internal top edge of the control and the top edge of the form. This value applies during the design phase only, because the control is invisible when the user runs your application.

Property Details

This section provides detailed descriptions of properties in DisXfr that are unique to this control. For details and examples of the standard Visual Basic properties, refer to the Microsoft documentation.

About

This property displays a pop-up window that shows copyright and version information about DisXfr and IBM. This property cannot be changed.

Cancel

Type: Boolean

ODisXfr only, this property can stop a transfer that is in progress.

TRUE = Stop transferring files. FALSE = Do not interrupt the transfer.

For example:

```
MyODisXfr.Cancel = TRUE
```

ConvertTbl

Type: string

This property specifies the name of the conversion table to be used during the transfer. Your application must first define this table with the DisConv control.

For example:

```
myDisXfr.ConvertTbl = "\conv.tbl"
```

Date

Type: string

This property specifies a date limiting which files or objects are to be transferred. If you do not specify a date, everything is transferred. If you do specify a date, only the files bearing that date or a newer one are transferred.

The format is:

mm/dd/yy

where:

mm the month of the year (1 through 12)

dd the day of the month (1 through 31)

yy the last two digits of the year (94 through 99)

For example, to copy objects dated Dec. 12, 1994, or later:

```
transferDate = "12/12/94"
myDisXfr.Date = transferDate
```

Destination

Type: string

This property specifies the destination name (for transfer in either direction). Whether it is an IDS object name or a file name, the path must be fully qualified.

For example:

```
myDisXfr.Destination = "c:\temp"
```

ErrCode

Type: long

(For PowerBuilder only.) This property returns the error code after any operation.

For example:

```
If myDisXfr.ErrCode <>
Then ReportErr()
EndIf
```

Mode

Type: integer

This property specifies the direction of the file transfer and is a trigger to activate the file transfer process. Valid values are:

0 from the IDS desktop to the client

1 from the client to the IDS desktop

For example, to transfer files to the client from IDS:

```
setDirection =
myDisXfr.Mode = setDirection
```

PowerBuilder
Type: Boolean

This property specifies that your application is developed in the PowerBuilder environment. Otherwise, set this property to FALSE.

For example, to specify that you are not using PowerBuilder:

```
myDisXfr.PowerBuilder = FALSE
```

Purge
Type: Boolean

This property specifies whether to delete the source object contents after it is transferred.

TRUE Delete the original source object contents.

FALSE Do not delete the original source object contents.

For example, to move the file, rather than just copying it:

```
setPurge = FALSE
myDisXfr.Purge= setPurge
```

Recursive
Type: Boolean

This property specifies whether to copy all the files and objects within the container, including any nested containers and their contents.

TRUE Copy all of the container's contents.

FALSE Copy only the name and date of the container itself.

For example, to copy everything:

```
setRecursive= FALSE
myDisXfr.Recursive = setRecursive
```

Source
Type: string

This property specifies the name of the file or object to be transferred (in either direction). Whether it is an IDS object name or a file name, the path must be fully qualified.

For example, to copy an IDS object named July:

```
setSource="[verbar]Data1[verbar]Sales[verbar]Reports[verbar]July"
myDisXfr.Source = setSource
```

Glossary

client. A workstation that interacts with an ODAS service for access to desktop functions such as Capsule applications and mail.

cluster. A set of execution engines. A service such as Capsule Services can have several clusters in different computers, but each cluster is in a single computer.

cluster manager. A process used by certain services, such as Capsule Services, to control one cluster, under direction from a service manager;the service manager can control several cluster managers in different computers.

container. An icon, such as the Folder and File Drawer icons, that can hold other icons.

desktop. The screen space occupied by DIS.

Ethernet. A type of local area network used for DIS connections. An Ethernet network uses a bus topology (that is, all devices are connected to a single cable). An Ethernet network can include any of several kinds of physical transmission media.

execution engine. A process used by certain services, such as Capsule Services, to execute specific actions under the control of a service manager through a cluster manager.

file drawer. A container, created or renamed by the system administrator, that can be shared among desktops. A file drawer typically contains folders.*See also* folder.

file server. The computer on which the file service runs.

file service. In DIS, a software process that manages disk storage of the system software and user data.

folder. An organizing container that resides on a single desktop and is used to store icons and other folders. Folders are represented on the DIS desktop by the Folder icon, on which a file folder is depicted.

icon. A graphic representation of a tool or service or object.

internetwork. Two or more local area networks (LANs)interconnected for exchange of data.

LAN. *See* local area network.

local area network (LAN). A data communications network, spanning a limited geographical area, in which serial transmission is used for direct communication among computers or other devices.

local network. The network to which a specified user or resource is directly connected.

message bar. The rectangular area along the bottom of the desktop, where advisory or error messages are displayed.

network. A set of workstations, servers, and other devices connected together for the interchange of information.

Network Device Interface Specification (NDIS). A protocal standard used in IBM NETBIOS software.

remote network. A network other than the local network, but which is accessible to the local network by some means.

server. The computer in which a service runs.

service. A software process that provides facilities to other parts of DIS, such as a file service, a mail service, a mail gateway service, and a data access service.

service manager. A process used by certain services, such as Capsule Service, to control one or more sets of execution engines through one or more cluster managers.

terminate-and-stay-resident (TSR). To occupy memory, ready to run, after initial execution is completed.

token-ring network. A network that uses a ring topology, in which tokens are passed in a circuit from node to node. A node that is ready to send can capture the token and insert data for transmission.

tool. A DIS workstation program, such as Text, Reporter, or Query. Every tool is represented by an icon.

TSR. Terminate-and-stay-resident.

user id. The name you use to access a desktop, system, gateway, or database service.

© Copyright IBM Corp. 1996

workstation. A computer at which the user can use DIS to perform work. Workstation components include the monitor, keyboard, mouse or trackball, and system unit, which houses one or more disk drives.

Bibliography

This bibliography includes books referenced in this book and a list of all the books in the IDS library. To order copies of the books listed here, or to get more information about IBM publications, contact an IBM representative.

Referenced Publications

These publications describe the Visual Basic language and programming environment.

Microsoft Visual Basic Programmers Guide

Microsoft Visual Basic Language Reference

IDS Publications

This section lists IDS publications, including order numbers. To order copies of the books listed here, or to get more information about a book, see your IBM representative.

General Publications

Getting Started with the Intelligent Decision Server (SC26-8892)

Working with the Intelligent Decision Server (SC26-8893)

Installation, Configuration, and Administration Publications

Installing IDS LAN Components (GC26-8877)

System Administration Guide and Reference (SC26-8890)

Messages and Codes (GC26-8878)

Database Gateway Services Publications

Installing Database Gateway Services for UNIX Environments (GC26-8884)

Configuring and Administering Database Gateway Services (GC26-8885)

Administering Databases with IDS (SC26-8891)

Host Services Publications

Planning MVS Host-to-LAN Connectivity (GC26-8880)

Installing Host Services and Configuring Host-to-LAN Communications for MVS (GC26-8881)

Administering Host Services for MVS: DB2 and Cooperative Application Services (GC26-8882)

Developing Applications with Cooperative Application Services (GC26-8883)

Configuring and Administering Mail Gateway Services (GC26-8923)

Open Data Access Service Publications

Developing Applications with Open Data Access Service (SC26-8888)

Data Access Tools Publications

Browser User's Guide (SC26-8900)

Data Entry User's Guide (SC26-8897)

DBA Tool User's Guide (SC26-8894)

SQL Entry User's Guide (SC26-8898)

© Copyright IBM Corp. 1996

Data Analysis Tools Publications

BASIC Tool User's Guide (SC26-8908)

Query User's Guide (SC26-8901)

Spreadsheet User's Guide (SC26-8904)

Presentation Tools Publications

Layout User's Guide (SC26-8902)

Plot User's Guide (SC26-8905)

Reporter User's Guide (SC26-8903)

Text User's Guide (SC26-8896)

Application Development Tools Publications

Transformers Guide (SC26-8907)

Capsule User's Guide (SC26-8906)

Forms Design User's Guide (SC26-8899)

Interacting with PC Applications (SC26-8895)

Installing and Configuring IDS Open Clients (GC26-8889)

Index

A

About property 230, 238, 242, 249, 255, 265, 269, 276, 286
Action property 211, 249, 265, 276
Action_Index property 211, 224
administration console 5
AFDExecBegin event 238, 255
AFDExecEnd event 238, 255
AFDInitBegin event 238, 255
AFDInitEnd event 238, 255
Align property 238, 255
API functions
  DisAFD 40, 119
  DisAFDList 41, 119
  DisBufferToVBArray 94, 101, 120
  DisBufferToVBType 96, 97, 99
  DisCancelRequest 41, 123
  DisCFormatSize 92, 123
  DisConnect 42, 91, 92, 124
  DisCopyObject 43, 125
  DisCreateBuffer 92, 94, 96, 97, 99, 101, 102, 103, 104, 127
  DisDeleteObject 44, 127
  DisDisconnect 33, 44, 93, 107, 128
  DisExecuteAFD 45
  DisFreeBuffer 92, 94, 96, 97, 99, 102, 103, 104, 129
  DisGetCapsuleVars 46, 101, 129
  DisGetErrorInfo 47, 131
  DisGetMultObjects 35, 48, 96, 109, 132
  DisGetObject 35, 51, 95, 109, 135
  DisGetPerfInfo 53, 137
  DisGetSysMessage 55, 139
  DisInitAFD 56
  DisInitAFDList 57
  DisInitBatch 35, 58, 110, 140
  DisInitClient 59, 91, 92, 141
  DisLogoff 33, 59, 93, 107, 141
  DisLogon 33, 60, 91, 92, 107, 142
  DisMkTime 103, 143
  DisMoveObject 61, 98, 100, 144
  DisObjectCapsuleInfo 67, 151
  DisPutMultObjects 35, 62, 98, 109, 145
  DisPutObject 35, 63, 97, 98, 99, 109, 147
  DisQueryCapsuleInfo 30, 31, 33, 64, 104, 105, 107, 148
API functions *(continued)*
  DisQueryObjectInfo 33, 93, 94, 103, 107
  DisQueryResult 31, 69, 105, 154
  DisQuerySessionInfo 70, 155
  DisSetCapsuleStatus 72, 157
  DisSetCapsuleVars 74, 86, 102, 159, 170
  DisSetConversionTable 75, 96, 97, 99, 162
  DisSetCurrentContainer 34, 77, 108, 164
  DisSetEventHandler 77, 164
  DisSetPassword 81, 165
  DisSetSessionInfo 81, 165
  DisSubmitBatch 35, 83, 110, 167
  DisSubmitCapsule 84, 104, 168
  DisTermAFD 88
  DisTermBatch 36, 88, 110, 170
  DisTermClient 89, 93, 173
  DisVBArrayToBuffer 102, 173
  DisVBFormatSize 175
  DisVBTypeToBuffer 92, 104, 175
  DisWaitCapsuleStatus 90, 104, 176
  list of 37, 38, 39, 112, 114
application development tools publications, list of 9
AtVar property 265
AutoAFD property 242, 255
AutoMenu 7

B

BackColor property 230, 238, 255
Background_Color property 211
batch submissions
  checking results 154
  continuing after error 16
  detailed results information 131
  submitting 167
  terminating 172
BaudRate property 242, 255
bit flags 14, 16
BorderStyle property 211, 238, 255
Browser control 206
Browser window 180

C

Cancel property 276, 286
capsule applications

© Copyright IBM Corp. 1996

capsule applications *(continued)*
API functions for 38, 113
awaiting completion of 90, 104, 176
checking current job status 104, 148
getting variables from 46, 101, 129
job-parameter flags for 16, 17, 103
setting status of 72, 157
setting variables in 74, 100, 159, 170
submitting for execution 84, 102, 168, 170
Capsule job-state codes 66, 150
capsule variables, setting 100
CapsuleName property 265, 269
Caption property 211
classes in IDS 15
Clear method 249, 275
Click event 208, 229, 253
client manager 5
Close event 253
Column_Caption property 211
Column_Count property 211
Column_Min_Width property 211
Column_Operator property 211
Column_Spacing property 211
Column_Width property 211
CommPort property 242, 255
CommType property 242, 255
communication manager 6
completion codes for DisQueryResult function 69, 154
Completion event 253
components 1, 4
Connect property 242, 255
ConnectDIS property 242, 255
Connected event 238, 255
console, administration 5
ConstrainButtonClicked event 208, 211, 220
Constraint Designer window 220, 226
controls
Browser 206
DisCapQ 181
DisCom 182, 255
DisConv 182, 184
DisDir 183, 184
DisSetVs 183
DisSubCap 183
DisTxfer 183
DisXfr 184
ODisCom 182, 255
ODisTxfr 183
ODisXfr 184
ConvertTbl property 286
Count property 249
CurCapId property 230

D

data analysis tools publications, list of 9
data types, list of 39
Database Gateway Services publications, list of 17
date format 236, 288
Date property 286
DblClick event 229, 253
Description property 265, 269
desktop objects
API functions for manipulating 38, 113
types, list of 13
Desktop property 242, 255
DestContainer property 269
DestDir property 276
Destination property 276, 286
detail selection codes
DisQueryCapsuleInfo 65, 149
DisQueryObjectInfo 67, 152
DIS_DELETE_EXPUNGE flag 16
DIS_DELETE_IFEMPTY flag 15
DIS_DELETE_SOURCE flag
description 15
with DisPutMultObjects 62, 145
with DisPutObject 63, 147
DIS_DISCARD_OUTPUTS flag 16
DIS_GMO_ID_NAME flag 50, 134
DIS_GMO_OBJECT_INFO flag 50, 134
DIS_GMO_SOURCE_NAME flag 50, 134
DIS_IGNORE_ERROR flag 16
DIS_LOFF_AFD_NOW flag 59, 142
DIS_LOFF_AFD_SUPPRESS flag 59, 142
DIS_LON_AFD_DEFER flag 59, 60, 142, 143
DIS_LON_AFD_NOW flag 60, 143
DIS_LON_AFD_SUPPRESS flag 60, 143
DIS_NOWAIT flag 15
DIS_OVERWRITE flag 16
DIS_PRESERVE_TIME flag 16
DIS_QCI_OWNED flag 66, 150
DIS_SB_SUBRESULTS flag 83, 167
DIS, classes in 15, 16
DisAFD API function 40, 119
DisAFDList API function 41, 119
DisBufferToVBArray API function 94, 101, 120
DisBufferToVBType API function 96, 97, 99

DisCancelRequest API function 41, 123
DisCapQ control 181
DisCapQ list box 181
DisCFormatSize API function 92, 123
DisCom control 182, 255
DisCom icon 255
DISCOM.VBX file 255
DisConnect API function 42, 91, 92, 124
DisConv control 182, 184
DisCopyObject API function 43, 125
DisCreateBuffer API function 92, 94, 96, 97, 99, 101, 102, 103, 104, 127
DisDeleteObject API function 44, 127
DisDir control 183, 184
DisDir list box 183
DisDisconnect API function 33, 44, 93, 107, 128
DisExecuteAFD API function 45
DisFreeBuffer API function 92, 94, 96, 97, 99, 102, 103, 104, 129
DisGetCapsuleVars API function 46, 101, 129
DisGetErrorInfo API function 47, 131
DisGetMultObjects API function 35, 48, 96, 109, 132
DisGetObject API function 35, 51, 95, 109, 135
DisGetPerfInfo API function 53, 137
DisGetSysMessage API function 55, 139
DisInitAFD API function 56
DisInitAFDList API function 57
DisInitBatch API function 35, 58, 110, 140
DisInitClient API function 59, 91, 92, 141
diskettes, validation 18
  default settings on 92
DisLogoff API function 33, 59, 93, 107, 141
DisLogon API function 33, 60, 91, 92, 107, 142
DisMkTime API function 103, 143
DisMoveObject API function 61, 98, 100, 144
DisPutMultObjects API function 35, 62, 98, 109, 145
DisPutObject API function 35, 63, 97, 98, 99, 109, 147
DisQueryCapsuleInfo API function 30, 31, 33, 64, 104, 105, 107, 148
DisQueryObjectInfo API function 33, 67, 93, 94, 103, 107, 151
DisQueryResult API function 31, 69, 105, 154
DisQuerySessionInfo API function 70, 155
DisSetCapsuleStatus API function 72, 157
DisSetCapsuleVars API function 74, 86, 102, 159, 170
DisSetConversionTable API function 75, 96, 97, 99, 162
DisSetCurrentContainer API function 34, 77, 108, 164
DisSetEventHandler API function 77, 164
DisSetPassword API function 81, 165
DisSetSessionInfo API function 81, 165
DisSetVs control 183
DisSubCap control 183
DisSubmitBatch API function 35, 83, 110, 167
DisSubmitCapsule API function 84, 104, 168
DisTermAFD API function 88
DisTermBatch API function 36, 88, 110, 170
DisTermClient API function 89, 93, 173
DisTxfer control 183
DisVBArrayToBuffer API function 102, 173
DisVBFormatSize API function 175
DisVBTypeToBuffer API function 92, 104, 175
DisWaitCapsuleStatus API function 90, 104, 176
DisXfr control 184
Drag method 208
DragDrop event 208, 229, 253
DragIcon property 230, 238, 255
DragMode
  property 255
DragMode property 230, 238
DragOver event 208, 229, 253
dynamic-link library 9

E

education in IDS 16
electronic mail
  see mail 8, 37, 112
Enabled property 230, 238, 242, 255
ErrCode property 230, 238, 242, 249, 255, 265, 269, 276, 286
Error event 283
events
  AFDExecBegin 238, 255
  AFDExectEnd 238, 255
  AFDInitBegin 238, 255
  AFDInitEnd 238, 255
  Click 208, 229, 253
  Close 253
  Completion 253
  Connected 238, 255
  ConstrainButtonClicked 208, 211, 220
  DblClick 229, 253
  DragDrop 208, 229, 253
  DragOver 208, 229, 253
  Error 283
  FileBegin 238, 255, 275, 283
  FileCount 283
  FileProgress 238, 255, 275, 283

events *(continued)*
GotFocus 208, 229, 253
KeyDown 208, 229, 253
KeyPress 208, 229, 253
KeyUp 208, 229, 253
ListItemDraw 208, 211, 223
ListItemFormat 208, 211, 224
LostFocus 208, 229, 253
MouseDown 229, 253
MouseMove 229, 253
MouseUp 229, 253
Open 253
Update 208, 229, 238, 249, 253, 255, 264, 269, 275, 283
ExecAFD property 242, 255

F

file format, TEXT 22, 23, 95, 99
file transferal, API functions for 37, 113
file types, list of 12
FileBegin event 238, 255, 275, 283
FileCount event 283
FileExt property 249
FileName property 211, 249
FileNum property 211
FileProgress event 238, 255, 275, 283
files
DISCOM.VBX 255
moving to the desktop 98
ODISCOM.VBX 255
retrieving from the desktop 100
FileType property 249, 276
flags
bit flags 14, 16
DIS_DELETE_EXPUNGE 16
DIS_DELETE_IFEMPTY 15
DIS_DELETE_SOURCE
description 15
with DisPutMultObjects 62, 145
with DisPutObject 63, 147
DIS_DISCARD_OUTPUTS 16
DIS_GMO_ID_NAME 50, 134
DIS_GMO_OBJECT_INFO 50, 134
DIS_GMO_SOURCE_NAME 50, 134
DIS_IGNORE_ERROR 16
DIS_LOFF_AFD_NOW 59, 142
DIS_LOFF_AFD_SUPPRESS 59, 142
DIS_LON_AFD_DEFER 59, 60, 142, 143
DIS_LON_AFD_NOW 60, 143
flags *(continued)*
DIS_LON_AFD_SUPPRESS 60, 143
DIS_NOWAIT 15
DIS_OVERWRITE 16
DIS_PRESERVE_TIME 16
DIS_QCI_OWNED 66, 150
DIS_SB_SUBRESULTS 83, 167
job-parameter flags 16, 17
FontBold property 230, 238, 255
FontItalic property 230, 242, 255
FontName property 230, 242, 255
FontSize property 230, 242, 255
FontStrikethru property 230
FontUnderline property 230
ForeColor property 230, 242, 255
formats
date 236, 288
time 236

G

Get_Column_Text property 211, 224
Get_Selected_Text property 211, 224
GotFocus event 208, 229, 253

H

handle, returning upon connection 40, 41, 42, 44, 45, 47, 48, 50, 52, 54, 57, 58, 59, 60, 62, 63, 64, 66, 69, 70, 72, 73, 75, 76, 77, 80, 81, 82, 83, 85, 87, 88, 89, 90, 120, 123, 125, 138
Height property 211, 230, 242, 255
help, classroom 16
HelpContextID property 242, 255
HideFocusBox property 230
Host Services publications, list of 12
HWnd property 211

I

icons
DisCom 255
ODisCom 255
IDSkettes, validation
generating 6
Index property 230, 242, 249, 255, 265, 269, 276, 286
information selection codes
DisQueryCapsuleInfo 65, 124, 149, 175
DisQuerySessionInfo function 71, 155
DisSetSessionInfo function 82, 166

Init property 242, 255
instruction in IDS 16
instruction inIDS 14
ItemInvert property 242, 255

J

job-parameter flags, list of 16, 17
JobId property 269

K

KeyDown event 208, 229, 253
KeyPress event 208, 229, 253
KeyUp event 208, 229, 253

L

Left property 211, 230, 242, 249, 255, 265, 269, 276, 286
Left_Margin property 211
libraries
  compatibility with 10
  dynamic-link (DLL) 9
  import 9
LineWidth property 249, 276
list boxes
  DisCapQ 181
  DisDir 183
List method 264
List property 230, 242, 255, 265
ListAliasName property 230
ListCount property 230, 242, 255, 265, 276
ListDate property 242, 255
ListErrorIndex property 276
ListFid property 242, 255
ListId property 230
ListIndex property 230, 242, 249, 255, 276
ListItem_Count property 211
ListItem_Index property 211, 224
ListItem_LeftMargin property 211
ListItem_TopMargin property 211
ListItemDraw event 208, 211, 223
ListItemDraw_Event property 211
ListItemFormat event 208, 211, 224
ListItemFormat_Event property 211
ListMaxExecution property 230
ListMaxStartDelay property 230
ListMetPath property 242
ListModify property 230
ListOpen property 242, 255, 263
ListOwner property 230
ListPath property 242, 255
ListPriority property 230
ListSize property 242, 255
ListState property 230
ListTimeScheduled property 230
ListTimeSubmitted property 230
ListType property 242, 255
logging module 5
Logon property 242, 255
LostFocus event 208, 229, 253

M

mail
  API functions for 37, 113
  listing 93
  retrieving 95
  sending 96, 98
manager
  client 5
  communication 6
  network communication 5
  service 5
  session 5
MaxExecuteTime property 269
MaxStartDelayTime property 269
MDIChildWindow property 230
methods
  Clear 249, 275
  Drag 208
  List 264
  Move 208
  Refresh 208, 228, 249, 253
  RemoveItem 249, 275
Mode property 276, 286
MouseDown event 229, 253
MouseMove event 229, 253
MouseUp event 229, 253
Move method 208

N

Name property 242, 249, 255, 265, 269, 276, 286
network communication manager 5

O

object types, list of 13
ObjType property 249, 276
ODisCom control 182, 255
ODisCom icon 255
ODISCOM.VBX file 255
ODisTxfr control 183
ODisXfr control 184
Open event 253

P

parameter flags for capsule jobs 16, 17
Parent property 211, 230, 249, 286
Password property 242, 255
path
  checking current container 70, 155
  querying for desktop object 152
  separator character 164, 165
Path property 230, 242, 255
PathActivation property 255
PhoneNumber property 242, 255
PictureClosedCapsule property 182, 255
PictureClosedEnvelope property 182, 255
PictureClosedFolder property 255
PictureLeaf property 255
PictureMinus property 255
PictureOpenCapsule property 255
PictureOpenEnvelope property 255
PictureOpenFolder property 255
PicturePlus property 255
PowerBuilder property 230, 242, 249, 255, 265, 269, 276, 286
Priority property 269
product education 16
ProgTrigger property 255
properties
  About 230, 238, 242, 249, 255, 265, 269, 276, 286
  Action 211, 249, 265, 276
  Action_Index 211, 224
  Align 238, 255
  AtVar 265
  AutoAFD 242
  BackColor 230, 238, 255
  Background_Color 211
  BaudRate 242, 255
  BorderStyle 211, 238, 255
  Cancel 276, 286
  CapsuleName 265, 269
properties *(continued)*
  Caption 211
  Column_Caption 211
  Column_Count 211
  Column_Min_Width 211
  Column_Operator 211
  Column_Spacing 211
  Column_Width 211
  CommPort 242, 255
  CommType 242, 255
  Connect 242, 255
  ConnectDIS 242, 255
  ConvertTbl 286
  Count 249
  CurCapId 230
  Date 286
  Description 265, 269
  Desktop 242, 255
  DestContainer 269
  DestDir 276
  Destination 276, 286
  DragIcon 230, 238, 255
  DragMode 230, 238, 255
  Enabled 230, 238, 242, 255
  ErrCode 230, 238, 242, 249, 255, 265, 269, 276, 286
  ExecAFD 242, 255
  FileExt 249
  FileName 211, 249
  FileNum 211
  FileType 249, 276
  FontBold 230, 238, 255
  FontItalic 230, 242, 255
  FontName 230, 242, 255
  FontSize 230, 242, 255
  FontStrikethru 230
  FontUnderline 230
  ForeColor 230, 242, 255
  Get_Column_Text 211, 224
  Get_Selected_Text 211, 224
  Height 211, 230, 242, 255
  HelpContextID 242, 255
  HideFocusBox 230
  HWnd 211
  Index 230, 242, 249, 255, 265, 269, 276, 286
  Init 242, 255
  ItemInvert 242, 255
  JobId 269
  Left 211, 230, 242, 249, 255, 265, 269, 276, 286
  Left_Margin 211

properties *(continued)*
LineWidth 249, 276
List 230, 242, 255, 265
ListAliasName 230
ListCount 230, 242, 255, 265, 276
ListDate 242, 255
ListErrorIndex 276
ListFid 242, 255
ListId 230
ListIndex 230, 242, 249, 255, 276
ListItem_Count 211
ListItem_Index 211, 224
ListItem_LeftMargin 211
ListItem_TopMargin 211
ListItemDraw_Event 211
ListItemFormat_Event 211
ListMaxExecution 230
ListMaxStartDelay 230
ListMetPath 242, 255
ListModify 230
ListOpen 242, 255, 263
ListOwner 230
ListPath 242, 255
ListPriority 230
ListSize 242, 255
ListState 230
ListTimeScheduled 230
ListTimeSubmitted 230
ListType 242, 255
Logon 242, 255
MaxExecuteTime 269
MaxStartDelayTime 269
MDIChildWindow 230
Mode 276, 286
Name 230, 242, 249, 255, 265, 269, 276, 286
ObjType 249, 276
Parent 211, 230, 249, 286
Password 242, 255
Path 230, 242, 255
PathActivaton 255
PhoneNumber 242, 255
PictureClosedCapsule 182, 255
PictureClosedEnvelope 182, 255
PictureClosedFolder 255
PictureLeaf 255
PictureMinus 255
PictureOpenCapsule 255
PictureOpenEnvelope 255
PictureOpenFolder 255
PicturePlus 255
properties *(continued)*
PowerBuilder 230, 242, 249, 255, 265, 269, 276, 286
Priority 269
ProgTrigger 255
Purge 286
Recursive 286
Redraw 211
ScheduledAlias 269
ScheduledTime 269
ServerName 242, 255
Sort 255
Source 276, 286
SourceDir 276
Status 230
TabIndex 211, 230, 255
TabStop 211, 230, 255
Tag 230, 242, 249, 255, 265, 269, 276, 286
Text 211, 230
TextBlock 211
TextFile_Delimiter 211
TextFile_WithCaptions 211
TimeOut 242, 255
Top 230, 242, 249, 255, 265, 269, 276, 286
Value 265
ViewMode 255
Visible 211, 230, 255
Width 211, 230, 255
publications
administration 17
application development tools 9
data analysis tools 9
general 16
Host Services 8
OpenDIS (ODAS) 10
planning 16
presentation tools 9
Purge property 286

R

Recursive property 286
Redraw property 211
Refresh method 208, 228, 249, 253
RemoveItem method 249, 275
Replace property 265

S

ScheduledAlias property 269
ScheduledTime property 269
ServerName property 242, 255
service manager 5
session control, API functions for 37, 112
session handle, returning upon connection 40, 41, 42, 44, 45, 47, 48, 50, 52, 54, 57, 58, 59, 60, 62, 63, 64, 66, 69, 70, 72, 73, 75, 76, 77, 80, 81, 82, 83, 85, 87, 88, 89, 90, 120, 123, 125, 138
session manager 5
Sort property 255
Source property 276, 286
SourceDir property 276
Status property 230

T

TabIndex property 211, 230, 255
TabStop property 211, 230, 255
Tag property 230, 242, 249, 255, 265, 269, 276, 286
TEXT format 22, 23, 95, 99
Text property 211, 230
TextBlock property 211
TextFile_Delimiter property 211
TextFile_WithCaptions property 211
time format 236
time function (DisMkTime) 143
TimeOut property 242, 255
Top property 230, 242, 249, 255, 265, 269, 276, 286
training in IDS 15, 16
types
  data 39
  desktop objects 13
  file 12

U

Update event 208, 229, 238, 249, 253, 255, 264, 269, 275, 283

V

validation diskette 18
  default settings on 92
validation IDSkette
  generating 6
Value property 265
ViewMode property 255
Visible property 211, 230, 255

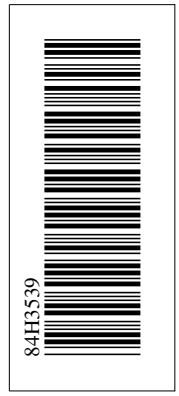
W

Width property 211, 230, 255
windows
  Browser 180
  Constraint Designer 220, 226

IBM®

Part Number: 84H3539

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.



SC26-8888-

Spine information:

IBM Intelligent Decision Server

Developing Applications with Open Data Access Service

Version 1