

ImageBASIC for NIC 1:View

User's Guide

ImageBASIC 3.1

IMAGE  *BASIC*

Diamond Head Software, Inc.
1217 Digital Drive Ste. 125
Richardson, Texas 75081-1970
(972) 479-9205

Copyright Notices

Under the copyright laws, neither the documentation nor the software may be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of Diamond Head Software, Inc., except in the manner described in the documentation.

This software product contains proprietary software components developed by a number of different software companies, referred herein as "Third Party Licensors". This documentation and the software that you purchased are protected by one or more of the following copyright notices:

Portions of this product, ©1993 - 1997 Diamond Head Software, Inc. All rights reserved.

Portions of this product, ©1990 - 1997 Network Imaging Systems Corporation, All rights reserved.

Company and product names mentioned in this documentation are trademarks or registered trademarks of their respective companies. Lotus and Lotus Notes are registered trademarks of Lotus Development Corporation. Windows is a trademark and Microsoft is a registered trademark of Microsoft Corporation.

DIAMOND HEAD SOFTWARE INC. AND ITS THIRD PARTY LICENSORS MAKE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. DIAMOND HEAD SOFTWARE, INC. AND ITS THIRD PARTY LICENSORS DO NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME JURISDICTIONS. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

IN NO EVENT WILL DIAMOND HEAD SOFTWARE INC. OR ITS THIRD PARTY LICENSORS AND/OR THEIR DIRECTORS, OFFICERS, EMPLOYEES OR AGENTS BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF DIAMOND HEAD SOFTWARE INC. OR ITS THIRD PARTY LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. Diamond Head Software Inc.'s and its Third Party Licensors' liability to you for actual damages from any cause whatsoever, and regardless of the form of the action (whether in contract, tort (including negligence), product liability or otherwise),

03/24/97 10:30

will be limited to \$50.

Contents

Preface: Installation	1
Installing and Configuring the NIC ActiveX's.....	1
Software Installation.....	1
Licensing Configuration.....	1
Chapter 1: Introduction to the NIC ActiveX's	3
Introduction to the NIC ActiveX's.....	3
Storing and Managing Information with 1: View Object Manager.....	3
Chapter 2: NIC Login Object	5
Using the NICLgn Control.....	5
Logging into the Server.....	5
Logging out of the Server.....	6
Chapter 3: NIC Simple Object Control	7
Using the NICObj Control.....	7
Storing an Object.....	7
Retrieving a Simple Object.....	8
Deleting a Simple Object.....	9
Working with the ObjectHandle Property.....	9
Chapter 4: NIC Group Object	11
Using the NICGrp Component.....	11
Storing a Group Object.....	11
Retrieving a Group Object.....	12
Deleting a Group Object.....	12
Appending a Group Object.....	13
Removing an Object from a Group.....	13
Querying for Object Attributes.....	13
Chapter 5: NIC Login Control Reference	17
NICLgn Reference.....	17
Chapter 6: NIC Simple Object Reference	21
NICObj Reference.....	21
Chapter 7: NIC Group Control Reference	27

NICGrp Reference.....27

Appendix A: Error Codes 35

Possible Error Codes35

Index 41

Preface: Installation

Installing and Configuring the NIC ActiveX's

Software Installation

Installation is performed with Windows on the client PC. The installation of the client controls does not also configure the client PC to communicate with the NIC Server. This setup must be performed according to the instructions provided with your network software.

To install the NIC ActiveX's:

1. Run Microsoft Windows.
2. If using Window's 95, run NICSETUP.EXE from the "Start" button or Explorer from the NIC Installation Diskette.
3. If using Window's 3.1, run NICSETUP.EXE from the Program Manager or File Manager, with the IB for NIC Installation Diskette.

You will be prompted to identify the installation directory. If ImageBASIC is already installed on this machine, it is recommended that ImageBASIC for NIC be installed in the same directory.

4. You will be notified when installation is complete.

The following files are installed. All of these files must be installed in a directory that is on the PATH:

dslib.dll	vmllib.dll	sslib.dll
wmcommon.dll	wmobj.dll	wmsecure.dll
wmcommon.ini	nictm32.dll	nicgrp.vbx
nicobj.vbx	nicscs.vbx	scom32.dll
score32.dll	scom.ini	

Licensing Configuration { XE "Licensing Configuration" }

Each ImageBASIC component can be separately licensed for use in development or at runtime. A development license can operate as a runtime license, allowing

the developer to test the application without acquiring an additional runtime license. Each ImageBASIC toolkit is shipped with a hardware key containing development licenses for the originally purchased components. This small device must be attached to a parallel port on the development PC. Another parallel device, such as a printer, can then be connected to the hardware key, and both devices will function normally. Licensing is verified a little differently at design time and at runtime, as detailed below.

Design Time Licensing Verification{ XE "Verification: Design Time Licensing " }{ XE "Design Time Licensing Verification " }

As each ImageBASIC control is loaded in a development environment, it will attempt to find and lock a single design license. This licensing verification request occurs immediately when an ImageBASIC control is added to a project, or a project containing an ImageBASIC control is loaded.

Runtime Licensing Verification{ XE "Verification: Runtime Licensing " }{ XE "Runtime Licensing Verification " }

All ImageBASIC controls employed in a compiled application must have access to a runtime license. A design time license can also be used at this time. Once a license is locked, it will not be released until the application terminates.

If no license is available, the control will not function and a Visual Basic runtime error will occur. The error number generated is 20001. This error is trappable with an **On Error ...** statement.

The Active Property

Active { XE "Active Property " }{ XE "Properties: Active" }{ XE "Runtime Licensing Verification: Active Property " } is a Boolean (True/False) property that specifies when licensing verification is requested during runtime. When a control is loaded as an executable is initialized, all of the ImageBASIC controls whose **Active** property is True will request licensing verification. If **Active** is False on initialization, that control will not request a license until it is used.

Because of the ability to selectively request licenses for each component, it is possible to design an application that selectively activates ImageBASIC components at each seat, perhaps based on a user identification process. This technique will help the system administrator purchase only the precise number of runtime licenses that are necessary for each ImageBASIC component in a large application.

{ XE "Introduction to the NIC ActiveX's"

}Chapter 1: Introduction to the NIC ActiveX's

Introduction to the NIC ActiveX's

Diamond Head Software has incorporated the technology from Network Imaging Corporation to give ImageBASIC users a flexible and reliable choice for storing images. NIC's IView represents a highly integratable server which can be integrated into your ImageBASIC application to provide you easy and accurate storage of you image files. NIC's IView: Object Manager is a system of client/server object management services which can be designed to meet the specific needs of your organization.

ImageBASIC for NIC is made up of three ActiveX's. The components are the Login, Simple Object{ XE "Simple Object" \r "D2HBSimple_Objec1" } and Group Object components. When developing an application with these components, you will first need to the utilize the Login ActiveX so that user can log into and out of the Object Manager server. Once logged on, users can then attach simple and group objects{ XE "Group Objects" \r "D2HBGroup_Object1" } to the server and retrieve remove previously saved objects.

Before you begin integrating the NIC server into your ImageBASIC application there are several I:View terms in which you need to be familiar. The *IView: Object Manager Administrator's Guide* provides a thorough explanation of these terms. If you need a brief refreshing of these terms or the *Administrator's Guide* is not available to you, the following explanation will give you a concise and brief explanation of the terms that are needed in order to understand the ImageBASIC for NIC ActiveX's.

{ XE "Storing and Managing Information with 1: View Object Manager" }Storing and Managing Information with 1: View Object Manager

{ XE "Units of Storage" }Units of Storage

The IView: Object Manager stores information in a hierarchy. The biggest unit of storage is the **Volume**.{ XE "Volume" \r "D2HBVolume2" } There can be many volumes with the storage system. They generally correspond to magnetic disk partitions or an optical disk. The next unit of storage on the IView: Object Manager system are the **Group Objects**. Group objects are generally a set of

related data. Each piece of data is known as a simple object. Group objects can store just one **Simple Object** or many group objects.

This page intentionally left blank

Chapter 2: NIC Login Object

Using the NICLgn Control

If the security on the server is enable, users can store and retrieve objects on the 1View: Object Manager, but they must first complete a successful login. The NICLgn ActiveX is the component which allows users to tap into the server.

Several security measures are exposed within this ActiveX. For your convenience, this component provides a built-in dialog box from which users can automatically login. If you prefer, you can customize your own login dialog box by setting a series of properties.

The NIC security component performs two simple functions: login and out of the NIC 1:View Object Manager Server.

{ XE "Logging on to the Server" }Logging into the Server

The Login component provides you with two ways to log into the server. The Login component provides a built-in dialog box which can easily be deployed. However, if you choose to design your own login, this can be accomplished through the setting of several properties.

Using the Built-in Login{ XE "Using the Built-in Login" }{ XE "Logging on to the Server: Using the Built-in Login" }

To utilize the built-in login dialog box, simply place the NICLgn ActiveX on the Visual Basic form. By setting this control's Login method, the control will display as a "Login" button during runtime if the **LoginDlgFlag** property{ XE "LoginDlgFlag Property" }{ XE "Properties: LoginDlgFlag" }{ XE "LoginDlgFlag Property: Using the Built-in Login" }{ XE "Using the Built-in Login: LoginDlgFlag Property" } is set to True. The **LoginDlgFlag** property is a Boolean (True/False) property which turns the login dialog box on or off based on its value. Once a user clicks the "Login" button (NICES control) a dialog box will prompt the user for username, password, and the server name that the user will want to log onto.

Customizing a Login Dialog Box{ XE "Logging on to the Server: Customizing a Login Dialog Box" }{ XE "Customizing a Login Dialog Box" }

If you need to provide a customized interface for your users, you can do so by making sure the following NICLgn properties are satisfied:

- ServerName property{ XE "Volume Property" }{ XE "Properties: Volume" }{ XE "Customizing a Login Dialog Box: Volume Property" }{ XE "Volume Property: Customizing a Login Dialog Box" }--Specifies the server name to log into.
- UserName property{ XE "UserName Property" }{ XE "Properties: UserName" }{ XE "Customizing a Login Dialog Box: UserName Property" }{ XE "UserName Property: Customizing a Login Dialog Box" }--The name of the user logging on.
- UserPassword property{ XE "UserPassword Property" }{ XE "Properties: UserPassword" }{ XE "Customizing a Login Dialog Box: UserPassword Property" }{ XE "UserPassword Property: Customizing a Login Dialog Box" }--The designated user password.

Once these properties are satisfied, the **Login** method can then be called.

{ XE "Connected Event" }{ XE "Logging on to the Server: Connected Event" }The Connected Event

After a successful login session, with either the built in dialog box or a customized one, the **Connected** event { XE "Events: Connected" }{ XE "Logging on to the Server: Connect Event" } will be triggered. Since this event is triggered after the login is successful, this is a good place to place code which informs access to the server has been granted, any startup instructions, messages from the server, etc.

{ XE "Logging out of the Server" }Logging out of the Server

The NIC Session object's **Logout** method can be called to disconnect as user from the database. Once the user has successfully logged out of the database the **Disconnected** event{ XE "Disconnected Event" }{ XE "Events: Disconnected" }{ XE "Disconnected Event: Logging out using the Logout Method" } is triggered. This is a good place to put code which informs the user that the NIC IView: Object Manager Server can no longer be accessed without logging back in.

Chapter 3: NIC Simple Object Control

{ XE "Using the Simple Object Component" }{ XE "Simple Object" \r "D2HBSimple_Objec4" }Using the NICObj Control

The Simple Object Component, NICObj ActiveX, allows users to create simple objects on the NIC server. Simple objects is an actual file such as a document, an image or sound file. When saving simple objects to the server there are several properties that are requested.

Simple objects, also known as files, can be stored either by themselves or in Groups on a Volume.{ XE "Volume" \r "D2HBVolume5" }

Storing an Object

Storing an object is similar to storing a single file on a database. The NIC ActiveX gives you several choices for storing objects. You can store an object by setting a few parameter properties and then calling the method.

When storing a simple object there are several properties that are required:

- **VolumeName**--{ XE "VolumeName Property" }{ XE "Properties: VolumeName" }{ XE "VolumeName Property: Creating a Simple Object" }{ XE "Creating a Simple Object: VolumeName Property" }Specifies the volume in which to store the newly created object. If you only have one volume specified, the Object Manager will default any new objects to this volume.
- **ObjectName**--{ XE "ObjectName Property" }{ XE "Properties: ObjectName" }{ XE "ObjectName Property: Creating a Simple Object" }{ XE "Creating a Simple Object: ObjectName Property" }This is name (string) that you must assign to the file that you are going to store on the server. The **ObjectHandle** property --{ XE "ObjectHandle Property" }{ XE "Properties: ObjectHandle" }{ XE "ObjectHandle Property: Creating a Simple Object" }{ XE "Creating a Simple Object: ObjectHandle Property" }assign a number to each object based on the **ObjectName**.
- **FileName**--{ XE "FileName Property" }{ XE "Properties: FileName" }{ XE "FileName Property: Creating a Simple

Object" }{ XE "Creating a Simple Object: FileName Property" }The name of the file on the local machine. This property must specify the fully qualified path of the object to store on the database.

After these properties are set, you can then call the method **StoreObject** { XE "StoreObject Method" }{ XE "Methods: StoreObject" }{ XE "StoreObject Method: Creating a Simple Object" }{ XE "Creating a Simple Object: StoreObject Method" }. The Object Manager will then take the information given to it and write the specified file to the server. It will then assign an object handle. **ObjectHandle** { XE "ObjectHandle Property" }{ XE "Properties: ObjectHandle" }{ XE "ObjectHandle Property: Creating a Simple Object" }{ XE "Creating a Simple Object: ObjectHandle Property" } is a string property that identifies the file on the server. By this identification number, the server knows the location and the object name of the file. When retrieving, deleting, or appending files to a group object, this property is required. In addition to the **ObjectHandle** property, the **DateTime** property { XE "DateTime Property" }{ XE "Properties: DateTime" }{ XE "DateTime Property: Creating a Simple Object" }{ XE "Creating a Simple Object: DateTime Property" } is also set. This property is a string that returns the date and time that object was created on the server.

After the file has been successfully written to the server, the **ObjectCreated** { XE "ObjectCreated Event" }{ XE "Events: ObjectCreated" }{ XE "ObjectCreated Event: Creating a Simple Object" }{ XE "Creating a Simple Object: ObjectCreated Event" } event is triggered. This event is a convenient place to store code that informs the user the file has been successfully saved.

You do not necessarily have to set the **VolumeName** property { XE "VolumeName Property" }{ XE "Properties: VolumeName" }{ XE "VolumeName Property: Creating a Simple Object" }{ XE "Creating a Simple Object: VolumeName Property" } if you have only one volume on your server. In this case, the **VolumeName** property would automatically use the default Volume. Also, if a volume is active, meaning you have just stored, deleted or retrieved a file from a particular volume, you would not have to set the name in the **VolumeName** property when saving to the volume.

There are also other property that can be set when storing a file on the server. These properties are strictly optional, but may help you manage data on the server:

- **ObjectClass**--{ XE "ObjectClass Property" }{ XE "Properties: ObjectClass" }{ XE "ObjectClass Property: Creating a Simple Object" }{ XE "Creating a Simple Object: ObjectClass Property" } If you want to divide your files into classes, the name of the class that a particular file belongs to would be set in this property. Classes can be

used to narrow a group of files into a particular category. For example, you may have a several simple objects saved in a group object called resumes. You could then give each of the resume files saved to the group object an object class of "Resumes."

- **ObjectCompression**--{ XE "ObjectCompression Property" }{ XE "Properties: ObjectCompression" }{ XE "ObjectCompression Property: Creating a Simple Object" }{ XE "Creating a Simple Object: ObjectCompression Property" }Specifies the amount of compression of the file while saving it to the server. This property can hold any value between 0 and 255.
- **ObjectEncryption**--{ XE "ObjectEncryption Property" }{ XE "Properties: DateTime" }{ XE "ObjectEncryption Property: Creating a Simple Object" }{ XE "Creating a Simple Object: ObjectEncryption Property" }Specifies the type of encryption that will be used when saving the file. This property can be set to any value between 0 and 255.

{ XE "Retrieving a Simple Object" }Retrieving a Simple Object

The NIC Simple Object will also retrieve objects from the server. When retrieving objects there are only four properties that must be set in order to have a successful retrieval:

- **ObjectHandle**--{ XE "ObjectHandle Property" }{ XE "Properties: ObjectHandle" }{ XE "ObjectHandle Property: Retrieving a Simple Object" }{ XE "Retrieving a Simple Object: ObjectHandle Property" }A number assigned by the server which identifies an object on the server.
- **FileName**--{ XE "FileName Property" }{ XE "Properties: FileName" }{ XE "FileName Property: Retrieving a Simple Object" }{ XE "Retrieving a Simple Object: FileName Property" }In retrieval, the **FileName** property specifies the fully qualified path on the client machine that the file from the server is to be placed.
- **VolumeName**--{ XE "VolumeName Property" }{ XE "Properties: VolumeName" }{ XE "VolumeName Property: Retrieving a Simple Object" }{ XE "Retrieving a Simple Object: VolumeName Property" }If the server contains more than one volume, the **VolumeName** property must specify which volume to point the client to. However, if the

client machine has just saved, removed, or retrieved a file from that the same volume, the server will default to that volume and the **VolumeName** property does not necessarily have to be set.

In addition to these properties, the **RetrieveObject** { XE "RetrieveObject Method" }{ XE "Methods: RetrieveObject" }{ XE "RetrieveObject Method: Retrieving a Simple Object" }{ XE "Retrieving a Simple Object: RetrieveObject Method" }method must be called.

{ XE "Deleting a Simple Object" }Deleting a Simple Object

Removing files from the server is the simplest function performed by the client. In order to remove a file only one property is required in addition to calling the **DeleteObject** method:

- **ObjectHandle** property--{ XE "ObjectHandle Property" }{ XE "Properties: ObjectHandle" }{ XE "ObjectHandle Property: Deleting a Simple Object" }{ XE "Deleting a Simple Object: ObjectHandle Property" }A number assigned by the server which identifies a file's name and location on the server.

After setting this property, you then need to call the **DeleteObject** --{ XE "DeleteObject Method" }{ XE "Methods: DeleteObject" }{ XE "DeleteObject Method: Deleting a Simple Object" }{ XE "Deleting a Simple Object: DeleteObject Method" }method.

Working with the ObjectHandle Property { XE "Working with the ObjectHandle Property" }

The **ObjectHandle** property, as stated before, is a string which is assigned by the NIC IView: Object Manager. This string not only identifies the file, but it also keeps track of the location of the file on the server.

There is no way to directly query the object handles of each file. One of the best way of tracking this information is to track this information in a database. This will allow the user to give meaningful names for objects.

This page intentionally left blank

Chapter 4: NIC Group Object

{ XE "Using the Group Object Component" }Using the NICGrp Component

The Group Object allows users to store files from a client machine on to the NIC server. A group object is a container which can contain multiple objects or one simple object. Group objects{ XE "Group Objects" \r "D2HBGroup_Object11"} are necessary because they help users organize data. Files with similar information can be contained within one group object. Group objects are then stored in Volumes.

The group object control is a non-visual control which means that it is invisible at runtime. It allows users to create, delete and append group objects. In addition, it also allows users to remove already existing group objects on the NIC Server. Additionally, it provides information about group objects such as volume name in which it is residing, number of objects it contains, and attributes.

{ XE "Storing a Group Object" }Storing a Group Object

After the log in session, users can create group object on the server. Group objects help users keep track and organize information. They must first be created on the server before simple objects can be stored to them. When creating a group object a sequence must be followed:

1. Specify the Volume on which to create the group through the **VolumeName** { XE "VolumeName Property" }{ XE "Properties: VolumeName" }{ XE "VolumeName Property: Creating a Group Object" }{ XE "Creating a Group Object: VolumeName Property" }property.
2. Specify a name for the group through the **ObjectName** { XE "ObjectName Property" }{ XE "Properties: ObjectName" }{ XE "ObjectName Property: Creating a Group Object" }{ XE "Creating a Group Object: ObjectName Property" }property.
3. Call the **StoreGroup** { XE "StoreGroup Method" }{ XE "Methods: StoreGroup" }{ XE "StoreGroup Method: Storing a Group Object" }{ XE "Creating a Group Object: StoreGroup Method" }method.

The server will then return a number which it assigns to the newly created group. This number can be queried through the **GroupHandle** { XE "GroupHandle

Property" }{ XE "Properties: GroupHandle" }{ XE "GroupHandle Property: Creating a Group Object" }{ XE "Creating a Group Object: GroupHandle Property" }property.

{ XE "Specifying the Volume" }Specifying the Volume

The Volume is the server onto which the newly created group is to be created. If there is only one volume specified by the system administrator, the **VolumeName** property does not have to be set. However, if there is more than one Volume, you must specify the correct one in the **VolumeName** property.

Specifying the Object Name{ XE "Specifying the Object Name" }

The **ObjectName** { XE "ObjectName Property" }{ XE "Properties: ObjectName" } property must be specified by the user creating the group on the server. If this property is not set, a group object can not be created on the server.

{ XE "Setting Class, Compression and Encryption" }Setting Class, Compression and Encryption (optional)

The **ObjectClass**, **ObjectCompression** and **ObjectEncryption** { XE "ObjectClass Property" }{ XE "Properties: ObjectClass" }{ XE "ObjectCompression Property" }{ XE "Properties: ObjectCompression" }{ XE "ObjectEncryption Property" }{ XE "Properties: ObjectEncryption" }properties are defined by the user. Either of these properties can be set to a value between 0 and 255. Both of these properties can be used as an identifier as to what type of file is being retrieved or stored. For example, all Word documents can be saved out with a Compression value of 15. This would let other user's know that files with a Compression value of 15 are Microsoft® Word documents.

Calling the StoreGroup Method { XE "Calling the StoreGroup Method" }

After you have set all of the necessary parameters, call the **StoreGroup** { XE "StoreGroup Method" }{ XE "Methods: StoreGroup" }method to create the group object on the server. Once a group object has been created, you can then attach files (simple objects) to it. A group object and the attaching of files to it can not be created at the same time.

Once a group object is created on the server, the server then assigns a group handle to it. The **GroupHandle** { XE "GroupHandle Property" }{ XE "Properties: GroupHandle" }is an identification number that allows users to add and remove simple objects from groups, and retrieve and remove groups from the server.

Because the **GroupHandle** is an alphanumerically value, a database should be kept which tracks the GroupHandle with its user assigned **GroupName**. This will allow users to assign meaningful names to group object.

{ XE "Retrieving a Group Object" }Retrieving a Group Object

Retrieving group objects from the IView Server requires several things be specified by the user:

1. The user must specified where on its machine it needs to place the retrieved group object. This is specified in the **FileName** { XE "FileName Property" }{ XE "Properties: FileName" }{ XE "FileName Property: Retrieving a Group Object" }{ XE "Retrieving a Group Object: FileName Property" }property. The **FileName** property must contain the full qualified name.
2. The **VolumeName** { XE "VolumeName Property" }{ XE "Properties: VolumeName" }{ XE "VolumeName Property: Retrieving a Group Object" }{ XE "Retrieving a Group Object: VolumeName Property" }property must specify the correct volume where the group object is located.
3. The **GroupHandle** property{ XE "GroupHandle Property" }{ XE "Properties: GroupHandle" }{ XE "GroupHandle Property: Retrieving a Group Object" }{ XE "Retrieving a Group Object: GroupHandle Property" } must specify the group to be retrieved.
4. Call the **RetrieveGroup** { XE "RetrieveGroup Method" }{ XE "Methods: RetrieveGroup" }{ XE "RetrieveGroup Method: Retrieving a Group Object" }{ XE "Retrieving a Group Object: RetrieveGroup Method" }method.

{ XE "Deleting a Group Object" }Deleting a Group Object

Deleting a group object will remove the object and all of its contents. To delete a group object you must first specify its handle through the **GroupHandle** { XE "GroupHandle Property" }{ XE "Properties: GroupHandle" }{ XE "GroupHandle Property: Deleting a Group Object" }{ XE "Deleting a Group Object: GroupHandle Property" }property. You must then call the **DeleteGroup** method.

{ XE "Appending a Group Object" }Appending a Group Object

Simple objects can be appended to a group object. Since group objects can not be created and populated at the same time, you will use this option often. To append a simple object{ XE "Simple Object" \r "D2HBSimple_Objec13" } to a group:

1. You must first specify the group object in which to save the simple object. This is specified through the **GroupHandle** { XE "GroupHandle Property: Appending a Group Object" }{ XE "Appending a Group Object: GroupHandle Property" }property. The server defaults to the last group object which received the action unless that action was a deletion. If you are wanting to save to this group object, then the **GroupHandle** property does not have to be specified.
2. Specify the object through the **ObjectHandle** property { XE "ObjectHandle Property" }{ XE "Properties: ObjectHandle" }{ XE "ObjectHandle Property: Appending a Group Object" }{ XE "Appending a Group Object: ObjectHandle Property" }for the object that needs to be added. You must first save a simple object to the database before it can be added to a group object. For more information on this process, please refer to the section "Storing an Object" in Chapter 3.
3. After the simple object has been identified, call the **AppendToGroup** { XE "AppendToGroup Method" }{ XE "Methods: AppendToGroup" }{ XE "AppendToGroup Method: Appending a Group Object" }{ XE "Appending a Group Object: AppendToGroup Method" }method.

{ XE "Removing an Object from a Group" }Removing an Object from a Group

Simple objects must be removed from a group one file at a time. This process operates in a similar manner to appending a simple object to a Group. To remove an object:

1. Specify the volume{ XE "Volume" \r "D2HBVolume14" } on which the group resides. This is set through the **VolumeName** { XE "VolumeName Property" }{ XE "Properties: VolumeName" }{ XE "VolumeName Property: Removing an Object from a Group" }{ XE "Removing an Object from a Group: VolumeName Property" }property.

The GroupObj ActiveX will automatically default to the volume with last received the action. If there is only one volume created on the server, it will be the default.

2. Specify the group object from which you want to remove the simple object. This is set through the **GroupHandle** { XE "GroupHandle Property" }{ XE "Properties: GroupHandle" }{ XE "GroupHandle Property: Removing an Object from a Group" }{ XE "Removing an Object from a Group: GroupHandle Property" }property. As with the **VolumeName** property, the **GroupHandle** defaults to the group which last received the action.
3. Specify the simple object to be deleted. This is set through the **ObjectHandle** { XE "ObjectHandle Property" }{ XE "Properties: ObjectHandle" }{ XE "ObjectHandle Property: Removing an Object from a Group" }{ XE "Removing an Object from a Group: ObjectHandle Property" }property.
4. Call the **RemoveFromGroup** { XE "RemoveFromGroup Method" }{ XE "Methods: RemoveFromGroup" }{ XE "RemoveFromGroup Method: Removing an Object from a Group" }{ XE "Removing an Object from a Group: RemoveFromGroup Method" }method.

Querying for Object Attributes

Once you have created a object in a group you can query for its attributes. There are several items of information that can be queried:

- object names
- object handles
- object classes
- object compression
- object encryption

Querying for the Object Name

Before an querying for an object name, the group must first be retrieved by utilizing the **RetrieveGroup** { XE "RetrieveGroup Method" }{ XE "Methods: RetrieveGroup" }{ XE "RetrieveGroup Method: Querying for the Object Name " }{ XE "Querying for the Object Name: RetrieveGroup Method" }method. Once the group has been successfully retrieved, you can query for the name of an object within that group by completing the following steps:

1. Query the **ObjectCount** { XE "ObjectCount Property" }{ XE "Properties: ObjectCount" }{ XE "ObjectCount Property: Querying for the Object Name " }{ XE "Querying for the Object Name: ObjectCount Property" }property to obtain the number of items existing in that group.
2. The **GetHandleList** { XE "GetHandleList Method" }{ XE "Methods: GetHandleList" }{ XE "GetHandleList Method: Querying for the Object Name " }{ XE "Querying for the Object Name: GetHandleList Method" }method should be then be called to retrieve the handle for each of the items in the group.
3. Set the **ItemIndex** { XE "ItemIndex Property" }{ XE "Properties: ItemIndex" }{ XE "ItemIndex Property: Querying for the Object Name " }{ XE "Querying for the Object Name: ItemIndex" }to any value between 0 and 1 minus the **ObjectCount** property.
4. Query the **ObjectName** { XE "ObjectName Property" }{ XE "Properties: ObjectName" }{ XE "ObjectName Property: Querying for the Object Name " }{ XE "Querying for the Object Name: ObjectName Property" }property to retrieve the name.

Querying for the Object Handle

Before an querying for an object's handle, the group must first be retrieved by utilizing the **RetrieveGroup** { XE "RetrieveGroup Method: Querying for the ObjectHandle" }{ XE "Querying for the ObjectHandle: RetrieveGroup Method" }method. Once the group has been successfully retrieved, you can query for the handle of an object within that group by completing the following steps:

1. Query the **ObjectCount** { XE "ObjectCount Property: Querying for the ObjectHandle" }{ XE "Querying for the ObjectHandle: ObjectCount Property" }property to obtain the number of items existing in that group.
2. The **GetHandleList** { XE "GetHandleList Method" }{ XE "Methods: GetHandleList" }{ XE "GetHandleList Method: Querying for the Object Count " }{ XE "Querying for the Object Count: GetHandleList Method" }method should be then be called to retrieve the handle for each of the items in the group.
3. Set the **ItemIndex** { XE "ItemIndex Property" }{ XE "Properties: ItemIndex" }{ XE "ItemIndex Property:

Querying for the Object Count " }{ XE "Querying for the Object Count: ItemIndex Property" }to any value between 0 and 1 minus the **ObjectCount** property.

4. Query the **ObjectHandle** { XE "ObjectHandle Property" }{ XE "Properties: ObjectHandle" }{ XE "ObjectHandle Property: Querying for the Object Count " }{ XE "Querying for the Object Count: ObjectHandle Property" } property to retrieve the object's handle.

Querying for the Object Class

Before an querying for an object's class, the group must first be retrieved by utilizing the **RetrieveGroup** { XE "RetrieveGroup Method" }{ XE "Methods: RetrieveGroup" }{ XE "RetrieveGroup Method: Querying for the Object Class " }{ XE "Querying for the Object Class: RetrieveGroup Method" }method. Once the group has been successfully retrieved, you can query for the class of an object within that group by completing the following steps:

1. Query the **ObjectCount** { XE "ObjectCount Property" }{ XE "Property: ObjectCount" }{ XE "ObjectCount Property: Querying for the Object Class " }{ XE "Querying for the Object Class: ObjectCount" }property to obtain the number of items existing in that group.
2. The **GetHandleList** { XE "GetHandleList Method" }{ XE "Methods: GetHandleList" }{ XE "GetHandleList Method: Querying for the Object Class " }{ XE "Querying for the Object Class: GetHandleList Method" }method should be then be called to retrieve the handle for each of the items in the group.
3. Set the **ItemIndex** { XE "ItemIndex Property" }{ XE "Properties: ItemIndex" }{ XE "ItemIndex Property: Querying for the Object Class " }{ XE "Querying for the Object Class: ItemIndex Property" }to any value between 0 and 1 minus the **ObjectCount** property.
4. Query the **ObjectClass** { XE "ObjectClass Property" }{ XE "Properties: ObjectClass" }{ XE "ObjectClass Property: Querying for the Object Class " }{ XE "Querying for the Object Class: ObjectClass Property" }property to retrieve the object's class.

Querying for the Object Encryption and Compression

Before an querying for an object's class, the group must first be retrieved by utilizing the **RetrieveGroup** { XE "RetrieveGroup Method" }{ XE "Methods:

RetrieveGroup" }} XE "RetrieveGroup Method: Querying for the Object Encryption and Compression" }} XE "Querying for the Object Encryption and Compression: RetrieveGroup Method" }method. Once the group has been successfully retrieved, you can query for the encryption or compression of an object within that group by completing the following steps:

1. Query the **ObjectCount** { XE "ObjectCount Property" }} XE "Properties: ObjectCount" }} XE "ObjectCount Property: Querying for the Object Encryption and Compression" }} XE "Querying for the Object Encryption and Compression: ObjectCount Property" }property to obtain the number of items existing in that group.
2. The **GetHandleList** { XE "GetHandleList Method" }} XE "Methods: GetHandleList" }} XE "GetHandleList Method: Querying for the Object Encryption and Compression" }} XE "Querying for the Object Encryption and Compression: GetHandleList Method" }method should be then be called to retrieve the handle for each of the items in the group.
3. Set the **ItemIndex** { XE "ItemIndex Property" }} XE "Properties: ItemIndex" }} XE "ItemIndex Property: Querying for the Object Encryption and Compression" }} XE "Querying for the Object Encryption and Compression: ItemIndex Property" }to any value between 0 and 1 minus the **ObjectCount** property.
4. To retrieve an object's encryption, query the **ObjectEncryption** { XE "ObjectEncryption Property" }} XE "Properties: ObjectEncryption" }} XE "ObjectEncryption: Querying for the Object Encryption and Compression" }} XE "Querying for the Object Encryption and Compression: ObjectEncryption Property" }property.

--or--

To retrieve an object's compression, query the **ObjectCompression** { XE "ObjectCompression Property" }} XE "Properties: ObjectCompression" }} XE "ObjectCompression: Querying for the Object Encryption and Compression" }} XE "Querying for the Object Encryption and Compression: ObjectCompression Property" }property.

Chapter 5: NIC Login Control Reference

NICLgn Reference

Active Property{ XE "Active Property" }{ XE "Properties: Active" }

Definition: If set to True at design time, the control will fully initialize and verify licensing immediately upon initialization of the runtime application.

If set to False at design time, full initialization of the control will be delayed at initialization of the runtime application. In this case, this property must be explicitly set to True at runtime before the control is used.

Data Type: Boolean

Design Access: Read/Write

Runtime Access: Read/Write (see limits below)

Comments: If this property is set to True (the default) at design time, the control is fully initialized and licensing is verified immediately upon initialization of the application at runtime. The related technology libraries are loaded and the control is ready to be used.

If this property is set to False at design time, the control will only partially initialize when the application loads at runtime. By delaying these two actions, the application should be able to load more quickly:

- 1) The related technology libraries for the control will not be loaded.
- 2) The licensing server will not verify an available token for the control.

If the control initializes with Active set to False, this property must be explicitly set to True by the application. Until Active is set to True, the control will ignore all instructions to it.

If the control fails to find a license token, the Active property will be automatically set to False. The application can check this value on Form Load to determine if each control is licensed and can be used.

Connected Event{ XE "Connected Event: NICLgn Control" }{ XE "NICLgn Control Events: Connected Event" }

Definition: Is triggered after a successful login by the user.

Disconnected Event{ XE "Disconnected Event: NICLgn Control" }{ XE "NICLgn Control Events: Disconnected Event" }

Definition: Is triggered after the user is logged out of the server.

Error Event{ XE "Error Event: NICLgn Control" }{ XE "NICLgn Control Events: Error Event" }

Definition: This event is triggered whenever an error occurs in the application. The last parameter is passed by reference allowing the user to specify the action after the completion of the event. The parameter is long and can have two possible values “0-Continue” which specifies that no dialog box is to be displayed, and “1-Show Display” which specifies that a message dialog box will be display.

Parameter: Number—Native error code.
Description—Text string describing the error.
SCode—Composite number indicating error severity, facility and original native error code.
HelpFile—Name of the help file associated with the error.
HelpContext—Number of the context ID in the help file.
CancelDisplay—Value specifying the action to take upon returning from the error event.

Login Method{ XE "Login Method: NICLgn Control" }{ XE "NICLgn Control: Login Method" }

Definition: Triggers the login of the user into the NIC server.

Parameters: None

Return Values: None

Data Type: Long

Syntax: NicSes1.Login

See Also: Connect Event, LoginDlgFlag Property, Logout Method, Password Property, UserName Property

Comments: Before this method can be called the UserName and Password properties must hold a value. If the LoginDlgFlag property is set to True, a dialog box will be displayed for the user. The dialog box contains textboxes which allow the user to fill in the username and password.

However, if a customized dialog box is to be used, the LoginDlgFlag property must be set to False.

LoginDlgFlag Property{ XE "LoginDlgFlag Property: NICLgn Control" }{ XE "NICLgn Control: LoginDlgFlag Property" }

Description:	When true, triggers a built-in dialog box for the login session.
Data Type:	Boolean
D/T Access:	Read/Write
R/T Access:	Read/Write
Possible Values:	True False
Default Value:	True
See Also:	Login Method, Password Property, UserName
Description:	The LoginDlgFlag property triggers a built-in login dialog box for the user. It contains textboxes for login information such as UserName and Password. During runtime, the NICSession control will appear as a login button. When the user clicks this button, the built-in dialog box will be triggered. If a login session is to be customized, the LoginDlgFlag property should be set to False. When LoginDlgFlag is set to true, the built-in dialog box will override a customized one.

Logout Method{ XE "Logout Method: NICLgn Control" }{ XE "NICLgn Control: Logout Method" }

Definition:	Disconnects the user from the NIC server.
Parameters:	None
Return Values:	None
Data Type:	Long
Syntax:	NicSes1.Logout
See Also:	Disconnected Event

Password Property{ XE "Password Property: NICLgn Control" }{ XE "NICLgn Control: Password Property" }

Description:	Specifies the password for the user for the login session.
Data Type:	String
D/T Access:	Read/Write
R/T Access:	Read/Write

Possible Values: Any valid password. Must be registered in the server.

Default Value: Null

See Also: Login Method, LoginDlgFlag { XE "LoginDlgFlag Property" }{ XE "Properties: LoginDlgFlag" }{ XE "LoginDlgFlag Property used with the Password Property" }property

Description: The Password property must be set to a valid string before calling the Login method. If the LoginDlgFlag property is set to true, a built-in dialog box will be triggered which provides a textbox for the user to fill in the Password property.

ServerName Property { XE "ServerName Property: NICLgn Control" }{ XE "NICLgn Control: ServerName Property" }

Description: Specifies the name of the server the user needs to log into.

Data Type: String

D/T Access: Read/Write

R/T Access: Read/Write

Possible Values: Any valid server name

Default Value: Read from the WMCOMMON.INI

See Also: Login Method

Description: The ServerName property must be set to a valid name before calling the Login Method. If there is only one server, the WMCOMMON.INI on the client machine will supply the default server name.

UserName Property{ XE "UserName Property: NICLgn Control" }{ XE "NICLgn Control: UserName Property" }

Description: Specifies the name of the user for the login session.

Data Type: String

D/T Access: Read/Write

R/T Access: Read/Write

Possible Values: Any valid user name. Must be registered in the server.

Default Value: Null

See Also: Login Method, LoginDlgFlag property{ XE "LoginDlgFlag Property" }{ XE "Properties: LoginDlgFlag" }{ XE "LoginDlgFlag Property used with the UserName Property" }

Description: The UserName property must be set to a valid name before calling the Login method. If the LoginDlgFlag property is

set to true, a built-in dialog box will be triggered which provides a textbox for the user to fill in the UserName property.

Chapter 6: NIC Simple Object Reference

NICObj Reference

Active Property{ XE "Active Property" }{ XE "Properties: Active" }

Definition: If set to True at design time, the control will fully initialize and verify licensing immediately upon initialization of the runtime application.

If set to False at design time, full initialization of the control will be delayed at initialization of the runtime application. In this case, this property must be explicitly set to True at runtime before the control is used.

Data Type: Boolean

Design Access: Read/Write

Runtime Access: Read/Write (see limits below)

Comments: If this property is set to True (the default) at design time, the control is fully initialized and licensing is verified immediately upon initialization of the application at runtime. The related technology libraries are loaded and the control is ready to be used.

If this property is set to False at design time, the control will only partially initialize when the application loads at runtime. By delaying these two actions, the application should be able to load more quickly:

- 1) The related technology libraries for the control will not be loaded.
- 2) The licensing server will not verify an available token for the control.

If the control initializes with Active set to False, this property must be explicitly set to True by the application. Until Active is set to True, the control will ignore all instructions to it.

If the control fails to find a license token, the Active property will be automatically set to False. The application can check this value on Form Load to determine if each control is licensed and can be used.

DateTime Property{ XE "DateTime Property: NICObj Control" }{ XE "NICObj Control: DateTime Property" }

Definition: Reports the date and time an object is created on the server.
Data Type: String
D/T Access: None
R/T Access: Read-only
Possible Values: Returned by the server API.

Error Event{ XE "Error Event: NICObj Control" }{ XE "NICObj Control Events: Error Event" }

Definition: This event is triggered whenever an error occurs in the application. The last parameter is passed by reference allowing the user to specify the action after the completion of the event. The parameter is long and can have two possible values “0-Continue” which specifies that no dialog box is to be displayed, and “1-Show Display” which specifies that a message dialog box will be display.

Parameter: Number—Native error code.
Description—Text string describing the error.
SCode—Composite number indicating error severity, facility and original native error code.
HelpFile—Name of the help file associated with the error.
HelpContext—Number of the context ID in the help file.
CancelDisplay—Value specifying the action to take upon returning from the error event.

FileName Property{ XE "FileName Property: NICObj Control" }{ XE "NICObj Control: FileName Property" }

Definition: Specifies the fully qualified path of the file on the client machine.

Data Type: String

D/T Access: None

R/T Access: Read/Write

Description: When storing files to the database, the FileName property specifies the (FQP) of the file residing on the client machine to store to the database. When retrieving files, the FileName property indicates (FQP) of where to store that file on the client machine.

ObjectCreated Event{ XE "ObjectCreated Event: NICObj Control" }{ XE "NICObj Control: ObjectCreated Event" }

Description: This event is triggered as soon as an object is successfully retrieved from the database.

ObjectClass Property{ XE "ObjectClass Property: NICObj Control" }{ XE "NICObj Control: ObjectClass Property" }

Definition: Specifies a user defined class that is assigned to the object stored on the server.

Data Type: Long

D/T Access: None

R/T Access: Read/Write

Description: This property is strictly optional when storing files to the server. This property helps users categorize their files.

ObjectCompression Property{ XE "ObjectCompression Property: NICObj Control" }{ XE "NICObj Control: ObjectCompression Property" }

Definition: The type of compression used for the object.

Data Type: Long

D/T Access: Read/Write

R/T Access: Read/Write

ObjectEncryption Property{ XE "ObjectEncryption Property: NICObj Control" }{ XE "NICObj Control: ObjectEncryption Property" }

Definition: The type of encryption used for the object.

Data Type: Long

D/T Access: Read/Write

R/T Access: Read/Write

ObjectHandle Property{ XE "ObjectHandle Property: NICObj Control" }{ XE "NICObj Control: ObjectHandle Property" }

Definition: Identifies a file on the server. The server assigns this number when an object is saved to it.

Data Type: Long

D/T Access: Read/Write

R/T Access: Read/Write

Description: This property must be set when retrieving or deleting a file from the server. When a file is saved, the server will give it a unique ObjectHandle which identifies the file and its location.

ObjectName { XE "ObjectName Property: NICObj Control" }{ XE "NICObj Control: ObjectName Property" }Property

Definition: User defined name for a simple object saved to the NIC server.

Data Type: String

D/T Access: None

R/T Access: Read/Write

Possible Values: Any string

Description: When saving a new object to the database, the user should specify a name in the ObjectName property. Once the object is created on the server, the object is assign an ID which is reported in the ObjectHandle property. The user can not use the ObjectName property to retrieve an object. The ObjectName should be kept in a database with its corresponding ObjectHandle.

ObjectType Property{ XE "ObjectType Property: NICObj Control" }{ XE "NICObj Control: ObjectType Property" }

Definition: Specifies the type of object to be saved to the server.

Data Type: Long

D/T Access: None

R/T Access: Read/Write

Description: Objects can be saved to the database either as a simple object or a group object. When using the simple object component, ObjectType property must be set to OBJECT_SIMPLE.

ObjectRetrieved Event{ XE "ObjectRetrieved Event: NICObj Control" }{ XE "NICObj Control: ObjectRetrieved Event" }

Definition: This event is triggered after an object is successfully retrieved from the server.

See Also: ObjectHandle { XE "ObjectHandle Property" }{ XE "Properties: ObjectHandle" }{ XE "ObjectHandle Property used with the ObjRetrieved Event" }property, RetrieveObject Method

RetrieveObject Method{ XE "RetrieveObject Method: NICObj Control" }{ XE "NICObj Control: RetrieveObject Method" }

Definition:	Retrieves a simple object from the NIC server.
Parameters:	None
Return Values:	None
Data Type:	Long
Syntax:	NICObj1.RetrieveObject
See Also:	FileName Property, ObjectHandle Property, ObjectRetrieved Event, VolumeName Property
Description:	Before the RetrieveObject method can be called, the VolumeName property should be set to the appropriate volume that the object is residing on if it is not in the default volume. Also the FileName property must specify where on the client machine to place the file once it is retrieved from the server. The ObjectHandle property must specify the object to retrieve. Then the RetrieveObject method can be called. Once the item is retrieved, the ObjectRetrieved event will be triggered.

StoreObject Method{ XE "StoreObject Method: NICObj Control" }{ XE "NICObj Control: StoreObject Method" }

Definition:	Stores an object on the NIC server.
Parameters:	None
Return Values:	None
Data Type:	Long
Syntax:	NicObj1.StoreObject
See Also:	DateTime Property, FileName Property, ObjectClass Property, ObjectCompression Property, ObjectCreated Event, ObjectEncryption Property, ObjectHandle Property, ObjectName Property, VolumeName Property
Comments:	Before this method can be called, there a several parameter properties that must be set. These include the VolumeName property, which specifies which volume to store the object onto. The ObjectName which is a string name assigned to the object. The FileName property which specifies on the client machine the fully qualified path of the object to store to the server. There are also several optional properties which can be set, such as ObjectClass, ObjectCompression, ObjectEncryption. Once these properties contain valid values, the StoreObject method can be called. This triggers the ObjectCreated event, and the

server assigns a handle to the object which can be found by querying the ObjectHandle property.

VolumeName Property{ XE "Volume Property: NIObj Control" }{ XE "NIObj Control: VolumeName Property" }

Definition:	Specifies the server volume.{ XE "Volume" \r "D2HBVolume10" }
Data Type:	String
D/T Access:	Read/Write
R/T Access:	Read/Write
Description:	Whenever the user needs to store, retrieve or delete a file from the server, the VolumeName property should be set. However, if there is only one volume residing on the server, the NIObj will automatically default to it.

Chapter 7: NIC Group Control Reference

NICGrp Reference

Active Property{ XE "Active Property" }{ XE "Properties: Active" }

Definition: If set to True at design time, the control will fully initialize and verify licensing immediately upon initialization of the runtime application.

If set to False at design time, full initialization of the control will be delayed at initialization of the runtime application. In this case, this property must be explicitly set to True at runtime before the control is used.

Data Type: Boolean

Design Access: Read/Write

Runtime Access: Read/Write (see limits below)

Comments: If this property is set to True (the default) at design time, the control is fully initialized and licensing is verified immediately upon initialization of the application at runtime. The related technology libraries are loaded and the control is ready to be used.

If this property is set to False at design time, the control will only partially initialize when the application loads at runtime. By delaying these two actions, the application should be able to load more quickly:

- 1) The related technology libraries for the control will not be loaded.
- 2) The licensing server will not verify an available token for the control.

If the control initializes with Active set to False, this property must be explicitly set to True by the application. Until Active is set to True, the control will ignore all instructions to it.

If the control fails to find a license token, the Active property will be automatically set to False. The application can check this value on Form Load to determine if each control is licensed and can be used.

AppendToGroup Method{ XE "AppendToGroup Method: NICGrp Control" }{ XE "NICGrp Control: AppendToGroup Method" }

Definition: Appends the specified file to a group from the NIC server.

Parameters: None

Return Values: None

Data Type: Long

Syntax: NICGrp1.AppendToGroup

See Also: GroupHandle Property, ObjectHandle Property

Description: The GroupHandle property should be set to the group that the is to be appended. The ObjectHandle must then specify the object that is to be appended to the group. After these properties are set the AppendToGroup property can be called.

DateTime Property{ XE "DateTime Property: NICGrp Control" }{ XE "NICGrp Control: DateTime Property" }

Definition: Reports the date and time for the created group object.

Data Type: String

D/T Access: None

R/T Access: Read-only

Possible Values: String returned from the API.

Default Value: Null

See Also: FileName Property, ObjectCreated Event, ObjectHandle Property, ObjectName Property, StoreObject Method, VolumeName Property,

DeleteGroup Method{ XE "DeleteGroup Method: NICGrp Control" }{ XE "NICGrp Control: DeleteGroup Method" }

Definition: Removes the specified group from the NIC server.

Parameters: None

Return Values: None

Data Type: Long

Syntax: NICGrp1.DeleteGroup

See Also: GroupHandle Property

Description: The group to be deleted must first be specified in the GroupHandle property. The DeleteGroup method can then be called to remove the group from the server.

Error Event{ XE "Error Event: NICGrp Control" }{ XE "NICGrp Control Events: Error Event" }

Definition:	This event is triggered whenever an error occurs in the application. The last parameter is passed by reference allowing the user to specify the action after the completion of the event. The parameter is long and can have two possible values "0-Continue" which specifies that no dialog box is to be displayed, and "1-Show Display" which specifies that a message dialog box will be display.
Parameter:	Number—Native error code. Description—Text string describing the error. SCode—Composite number indicating error severity, facility and original native error code. HelpFile—Name of the help file associated with the error. HelpContext—Number of the context ID in the help file. CancelDisplay—Value specifying the action to take upon returning from the error event.

FileName Property{ XE "FileName Property: NICGrp Control" }{ XE "NICGrp Control: FileName Property" }

Definition:	Specifies the file name and location on the client machine where the object is retrieved from or stored on the server.
Data Type:	String
D/T Access:	None
R/T Access:	Read/Write
Possible Values:	Any valid fully qualified path
Default Value:	Null
See Also:	DateTime Property, ObjectCreated Event, ObjectHandle Property, ObjectName Property, VolumeName Property, StoreObject Method
Description:	When retrieving objects from the server, the FileName property must contain the fully qualified name on the client machine. This path is where the file, once retrieved will be placed and named on the client machine. When appending objects to a group, the FileName property specifies the fully qualified name of the file, which resides on the client machine, to stored on the server.

GetObjectHandleList Method{ XE "GetObjectHandleList Method: NICGrp Control" }{ XE "NICGrp Control: GetObjectHandleList Method" }

Definition:	Retrieves the object handle list of objects found in a specified group in the NIC server.
Parameters:	None
Return Values:	None
Data Type:	Long
Syntax:	NICGrp1.GetObjectHandleList
See Also:	ItemIndex Property, ObjectClass Property, ObjectCompression Property, ObjectCount Property, ObjectEncryption Property, ObjectHandle Property, ObjectName Property, RetrieveGroup Method
Description:	Used mostly for retrieving object attributes, this method should be called after a group has been retrieved from the server. The ObjectCount and ItemIndex properties can then be used to specify the a particular object within the group. Various properties, such as ObjectName, ObjectHandle, ObjectClass, ObjectCompression and ObjectEncryption can then be queried.

GroupCreated Event{ XE "GroupCreated Event: NICGrp Control" }{ XE "NICGrp Control: GroupCreated Event" }

Definition:	This event is triggered after a successful creation of a new object.
--------------------	--

GroupHandle Property{ XE "GroupHandle Property: NICGrp Control" }{ XE "NICGrp Control: GroupHandle Property" }

Definition:	Specifies the object handle of the group created and stored on the server.
Data Type:	Long
D/T Access:	None
R/T Access:	Read/Write
Possible Values:	Any valid handle value.
Default Value:	0
See Also:	ObjectName Property, StoreGroup Method, VolumeName Property
Description:	When a group object is created, the server assigns it a GroupHandle. The GroupHandle property identifies the object. This property needs to be set whenever appending

to groups or removing files from a group, or deleting a group from the server.

GroupName Property{ XE "GroupName Property: NICGrp Control" }{ XE "NICGrp Control: GroupName Property" }

Definition: Specifies the name of the group on the server.

Data Type: String

D/T Access: Read/Write

R/T Access: Read/Write

See Also: GroupHandle Property, StoreGroup Property

Description: This property is assigned by the user to give the group a more meaningful identifier rather than the GroupHandle which is assigned by the server when the group is created. A database should be created which corresponds this property with the GroupHandle property.

GroupRetrieved Event{ XE "GroupRetrieved Event: NICGrp Control" }{ XE "NICGrp Control: GroupRetrieved Event" }

Definition: This event is triggered after a successful retrieval of a simple or group object.

ObjectClass Property{ XE "ObjectClass Property: NICGrp Control" }{ XE "NICGrp Control: ObjectClass Property" }

Definition: Specifies a user defined class that is assigned to the object stored on the server.

Data Type: Long

D/T Access: None

R/T Access: Read/Write

Description: This property is strictly optional when storing files to the server. This property helps users categorize their files.

ObjectCompression Property{ XE "ObjectCompression Property: NICGrp Control" }{ XE "NICGrp Control: ObjectCompression Property" }

Definition: The type of compression used for the object.

Data Type: Long

D/T Access: Read/Write

R/T Access: Read/Write

ObjectCount Property{ XE "ObjectCount Property: NICGrp Control" }{ XE "NICGrp Control: ObjectCount Property" }

Definition: Reports the number of objects present in a group object.

Data Type: Long

D/T Access: None

R/T Access: Read/Write

Possible Values: Long integer

See Also: GetHandleList Method

Description: The ObjectCount property displays a listing of all the simple object{ XE "Simple Object" \r "D2HBSimple_Objec17" } handles available under a given group. This property is populated after the GetHandleList method is called.

ObjectEncryption Property{ XE "ObjectEncryption Property: NICGrp Control" }{ XE "NICGrp Control: ObjectEncryption Property" }

Definition: The type of encryption used for the object.

Data Type: Long

D/T Access: Read/Write

R/T Access: Read/Write

ObjectHandle Property{ XE "ObjectHandle Property: NICGrp Control" }{ XE "NICGrp Control: ObjectHandle Property" }

Definition: Specifies the object handle ID that was created by the server when the simple object was created on the server.

Data Type: Long

D/T Access: None

R/T Access: Read/Write

Possible Values: Any valid handle value.

See Also: GetObjectHandleList Method

Description: The ObjectHandle property is necessary when appending or removing simple objects from groups. The ObjectHandle is created when the simple object is created on the server using the Simple Object ActiveX. A listing of the Object Handles can be obtained by performing a refresh by call the GetObjectHandleList method and then querying the ObjectHandle property.

ObjectIndex Property{ XE "ObjectIndex Property: NICGrp Control" }{ XE "NICGrp Control: ObjectIndex Property" }

Definition: Specifies the index of the currently selected object within a group. This property automatically updates with the property list for the corresponding object.

Data Type: Long

D/T Access: None

R/T Access: Read/Write

Possible Values: Any valid long value

Default Value: 0

ObjectIndexChanged Event{ XE "ObjectIndexChanged Event: NICGrp Control" }{ XE "NICGrp Control: ObjectIndexChanged Event" }

Definition: This event is triggered after the object index has been successfully updated.

ObjectName Property{ XE "ObjectName Property: NICGrp Control" }{ XE "NICGrp Control: ObjectName Property" }

Definition: String name assigned to a newly created group object. This property should be kept in a database which corresponds it to the ObjectHandle property.

Data Type: String

D/T Access: None

R/T Access: Read/Write

Possible Values: Any string

Default Value: Null

See Also: GroupHandle Property, StoreGroup Method, VolumeName Property

Description: The ObjectName property must be set before creating a group object{ XE "Creating a Group Object" } on the server. This is usually a string name that is assigned by the user.

RemoveFromGroup Method{ XE "RemoveFromGroup Method: NICGrp Control" }{ XE "NICGrp Control: RemoveFromGroup Method" }

Definition: Removes the file specified in the ObjectHandle property from the a group that is in the NIC server.

Parameters: None

Return Values: None

Data Type: Long

Syntax: NICGrp1.RemoveFromGroup

See Also: GroupHandle Property, ObjectHandle Property, VolumeName Property

Description: Before an object can be removed from a group, it must first be specified through the GroupHandle property. The ObjectHandle property must then specify which object to remove and the RemoveFromGroup method can be called.

RetrieveGroup Method{ XE "RetrieveGroup Method: NICGrp Control" }{ XE "NICGrp Control: RetrieveGroup Method" }

Definition: Retrieves the specified group from the NIC server.

Parameters: None

Return Values: None

Data Type: Long

Syntax: NICGrp1.RetrieveGroup

See Also: FileName Property, GroupHandle Property, VolumeName Property

Description: Before a group can be retrieved, the client must specify where on his/her machine the group is to be placed. This is specified in the FileName property. The VolumeName property must specify the correct volume and the GroupHandle property must specify the group to retrieve. Once all of these parameters are met, the RetrieveGroup method should be called.

StoreGroup Method{ XE "StoreGroup Method: NICGrp Control" }{ XE "NICGrp Control: StoreGroup Method" }

Definition: Stores the specified group onto the NIC server.

Parameters: None

Return Values: None

Data Type: Long

Syntax: NicGrp1.StoreGroup

See Also: GroupHandle Property, ObjectName Property, VolumeName Property

Description: Before the StoreGroup method can be called, the VolumeName property must specify the appropriate volume to store the group. The ObjectName property must specify a name for the group to be stored. The StoreGroup method

can then be called. The server will return a number which it assigns to the newly created group. This number can be queried through the GroupHandle property.

VolumeName Property{ XE "VolumeName Property: NICGrp Control" }{ XE "NICGrp Control: VolumeName Property" }

Definition:	Specifies the Volume on which to create, delete, or append group objects
Data Type:	String
D/T Access:	Read/Write
R/T Access:	Read/Write
Default Value:	Read from the WMCOMMON.INI file (Volume1)
See Also:	GroupHandle Property, ObjectName Property, StoreGroup Method
Description:	The VolumeName property specifies the largest store unit for the NIC 1:View server. The default for this property is read from the WMCOMMON.INI file. If only one volume exists on the server, setting of the property may not be necessary. This property also defaults to the last volume specified. This property should be set if there is more than one volume existing on the server. Use this property to specifies actions such as creating group objects, removing group objects, appending simple objects to groups, or removing simple objects from groups.

Appendix A: Error Codes

Possible Error Codes

1	Network error
2	User has already logged out of the server
3	User is not logged on to the server
4	Bad server name
5	Bad user name
6	Bad password
7	User has already logged into the server
8	Error in login dialog
9	Canceled login
10	Login failed
11	Time-out
12	Error in the object services library
13	Cannot create the group object
14	Cannot create the simple object{ XE "Simple Object" \r "D2HBSimple_Objec21" }
15	Failed to delete a simple object
16	Failed to delete a group object
17	Failed to append to group object
18	Cannot check-in the object
19	Cannot checkout the object
20	Failed to get the object description
21	Error in memory allocation of object handle list

22	Failed to create the object handle list
23	Invalid object index
24	Failed to update all object properties
25	Failed to remove object from group
26	Failed to retrieve a simple object
27	Failed to retrieve a group
28	Invalid simple object handle
29	Invalid group object handle
30	FileName not specified or invalid
31	Unable to get date stamp

Index

A

- Active Property 17, 21, 27
- Active Property 2
- Appending a Group Object 13
 - AppendToGroup Method 13
 - GroupHandle Property 13
 - ObjectHandle Property 13
- AppendToGroup Method 13
 - Appending a Group Object 13
 - NICGrp Control 28

C

- Calling the StoreGroup Method 12
- Connected Event 6
 - NICLgn Control 17
- Creating a Group Object 33
 - GroupHandle Property 11
 - ObjectName Property 11
 - StoreGroup Method 11
 - VolumeName Property 11
- Creating a Simple Object
 - DateTime Property 7
 - FileName Property 7
 - ObjectClass Property 8
 - ObjectCompression Property 8
 - ObjectCreated Event 8
 - ObjectEncryption Property 8
 - ObjectHandle Property 7
 - ObjectName Property 7
 - StoreObject Method 7
 - VolumeName Property 7, 8
- Customizing a Login Dialog Box 5
 - UserName Property 5
 - UserPassword Property 5
 - Volume Property 5

D

- DateTime Property 7
 - Creating a Simple Object 7
 - NICGrp Control 28
 - NICObj Control 22
- DeleteGroup Method
 - NICGrp Control 28
- DeleteObject Method 9

- Deleting a Simple Object 9
- Deleting a Group Object 12
 - GroupHandle Property 12
- Deleting a Simple Object 9
 - DeleteObject Method 9
 - ObjectHandle Property 9
- Design Time Licensing Verification 2
- Disconnected Event 6
 - Logging out using the Logout Method 6
- NICLgn Control 18

E

- Error Event
 - NICGrp Control 28
 - NICLgn Control 18
 - NICObj Control 22
- Events
 - Connected 6
 - Disconnected 6
 - ObjectCreated 8

F

- FileName Property 7, 8, 12
 - Creating a Simple Object 7
 - NICGrp Control 29
 - NICObj Control 22
 - Retrieving a Group Object 12
 - Retrieving a Simple Object 8

G

- GetHandleList Method 14, 15
 - Querying for the Object Class 15
 - Querying for the Object Count 14
 - Querying for the Object
 - Encryption and Compression 15
 - Querying for the Object Name 14
- GetObjectHandleList Method
 - NICGrp Control 29
- Group Objects 3, 11–13
- GroupCreated Event
 - NICGrp Control 30
- GroupHandle Property 11, 12, 13
 - Appending a Group Object 13
 - Creating a Group Object 11
 - Deleting a Group Object 12
 - NICGrp Control 30

Removing an Object from a Group
13

Retrieving a Group Object 12

GroupName Property
NICGrp Control 30
GroupRetrieved Event
NICGrp Control 31

I

Introduction to the NIC ActiveX's 3
ItemIndex Property 14, 15
 Querying for the Object Class 15
 Querying for the Object Count 14
 Querying for the Object
 Encryption and Compression
 15
 Querying for the Object Name 14

L

Licensing Configuration 1
Logging on to the Server 5
 Connect Event 6
 Connected Event 6
 Customizing a Login Dialog Box
 5
 Using the Built-in Login 5
Logging out of the Server 6
Login Method
 NICLgn Control 18
LoginDlgFlag Property 5, 19, 20
 NICLgn Control 19
 Using the Built-in Login 5
LoginDlgFlag Property used with the
 Password Property 19
LoginDlgFlag Property used with the
 UserName Property 20
Logout Method
 NICLgn Control 19

M

Methods
 AppendToGroup 13
 DeleteObject 9
 GetHandleList 14, 15
 RemoveFromGroup 13
 RetrieveGroup 12, 14, 15
 RetrieveObject 9
 StoreGroup 11, 12

StoreObject 7

N

NICGrp Control
 AppendToGroup Method 28
 DateTime Property 28
 DeleteGroup Method 28
 FileName Property 29
 GetObjectHandleList Method 29
 GroupCreated Event 30
 GroupHandle Property 30
 GroupName Property 30
 GroupRetrieved Event 31
 ObjectClass Property 31
 ObjectCompression Property 31
 ObjectCount Property 31
 ObjectEncryption Property 31
 ObjectHandle Property 32
 ObjectIndex Property 32
 ObjectIndexChanged Event 32
 ObjectName Property 32
 RemoveFromGroup Method 33
 RetrieveGroup Method 33
 StoreGroup Method 33
 VolumeName Property 34
NICGrp Control Events
 Error Event 28
NICLgn Control
 Login Method 18
 LoginDlgFlag Property 19
 Logout Method 19
 Password Property 19
 ServerName Property 20
 UserName Property 20
NICLgn Control Events
 Connected Event 17
 Disconnected Event 18
 Error Event 18
NICObj Control
 DateTime Property 22
 FileName Property 22
 ObjectClass Property 23
 ObjectCompression Property 23
 ObjectCreated Event 22
 ObjectEncryption Property 23
 ObjectHandle Property 23
 ObjectName Property 23
 ObjectRetrieved Event 24
 ObjectType Property 24

- RetrieveObject Method 24
- StoreObject Method 25
- VolumeName Property 25
- NICObj Control Events
 - Error Event 22
- O
 - ObjectClass Property 8, 12, 15
 - Creating a Simple Object 8
 - NICGrp Control 31
 - NICObj Control 23
 - Querying for the Object Class 15
 - ObjectCompression
 - Querying for the Object
 - Encryption and Compression 15
 - ObjectCompression Property 8, 12, 15
 - Creating a Simple Object 8
 - NICGrp Control 31
 - NICObj Control 23
 - ObjectCount Property 14, 15
 - NICGrp Control 31
 - Querying for the Object Class 15
 - Querying for the Object
 - Encryption and Compression 15
 - Querying for the Object Name 14
 - Querying for the ObjectHandle 14
 - ObjectCreated Event 8
 - Creating a Simple Object 8
 - NICObj Control 22
 - ObjectEncryption
 - Querying for the Object
 - Encryption and Compression 15
 - ObjectEncryption Property 8, 12, 15
 - Creating a Simple Object 8
 - NICGrp Control 31
 - NICObj Control 23
 - ObjectHandle Property 7, 8, 9, 13, 14, 24
 - Appending a Group Object 13
 - Creating a Simple Object 7
 - Deleting a Simple Object 9
 - NICGrp Control 32
 - NICObj Control 23
 - Querying for the Object Count 14
 - Removing an Object from a Group 13

- Retrieving a Simple Object 8
- ObjectHandle Property used with the
 - ObjRetrieved Event 24
- ObjectIndex Property
 - NICGrp Control 32
- ObjectIndexChanged Event
 - NICGrp Control 32
- ObjectName Property 7, 11, 14
 - Creating a Group Object 11
 - Creating a Simple Object 7
 - NICGrp Control 32
 - NICObj Control 23
 - Querying for the Object Name 14
- ObjectRetrieved Event
 - NICObj Control 24
- ObjectType Property
 - NICObj Control 24

P

- Password Property
 - NICLgn Control 19
- Properties
 - Active 2, 17, 21, 27
 - DateTime 7, 8
 - FileName 7, 8
 - FileName 12
 - GroupHandle 12, 13
 - GroupHandle 11, 12
 - ItemIndex 14, 15
 - LoginDlgFlag 5, 19, 20
 - ObjectClass 8, 12, 15
 - ObjectCompression 8, 12, 15
 - ObjectCount 14, 15
 - ObjectEncryption 12, 15
 - ObjectHandle 7, 8, 13, 14, 24
 - ObjectHandle 9, 13
 - ObjectName 7, 11, 14
 - ObjectName 11
 - UserName 5
 - UserPassword 5
 - Volume 5
 - VolumeName 7, 8, 13
 - VolumeName 11, 12
- Property
 - ObjectCount 15

Q

- Querying for the Object Class

- GetHandleList Method 15
- ItemIndex Property 15
- ObjectClass Property 15
- ObjectCount 15
- RetrieveGroup Method 14
- Querying for the Object Count
 - GetHandleList Method 14
 - ItemIndex Property 14
 - ObjectHandle Property 14
- Querying for the Object Encryption and Compression
 - GetHandleList Method 15
 - ItemIndex Property 15
 - ObjectCompression Property 15
 - ObjectCount Property 15
 - ObjectEncryption Property 15
 - RetrieveGroup Method 15
- Querying for the Object Name
 - GetHandleList Method 14
 - ItemIndex 14
 - ObjectCount Property 14
 - ObjectName Property 14
 - RetrieveGroup Method 14
- Querying for the ObjectHandle
 - ObjectCount Property 14
 - RetrieveGroup Method 14

R

- RemoveFromGroup Method 13
 - NICGrp Control 33
 - Removing an Object from a Group 13
- Removing an Object from a Group 13
 - GroupHandle Property 13
 - ObjectHandle Property 13
 - RemoveFromGroup Method 13
 - VolumeName Property 13
- RetrieveGroup Method 12, 14, 15
 - NICGrp Control 33
 - Querying for the Object Class 14
 - Querying for the Object Encryption and Compression 15
 - Querying for the Object Name 14
 - Querying for the ObjectHandle 14
 - Retrieving a Group Object 12
- RetrieveObject Method 9
 - NICObj Control 24
 - Retrieving a Simple Object 9

- Retrieving a Group Object 12
 - FileName Property 12
 - GroupHandle Property 12
 - RetrieveGroup Method 12
 - VolumeName Property 12
- Retrieving a Simple Object 8
 - FileName Property 8
 - ObjectHandle Property 8
 - RetrieveObject Method 9
 - VolumeName Property 8
- Runtime Licensing Verification
 - Active Property 2
- Runtime Licensing Verification 2

S

- ServerName Property
 - NICLgn Control 20
- Setting Class, Compression and Encryption 12
- Simple Object 3, 6–9, 13–27, 32–33, 35–36
- Specifying the Object Name 11
- Specifying the Volume 11
- StoreGroup Method 11, 12
 - NICGrp Control 33
 - Storing a Group Object 11
- StoreObject Method 7
 - Creating a Simple Object 7
 - NICObj Control 25
- Storing a Group Object 11
- Storing and Managing Information with 1
 - View Object Manager 3

U

- Units of Storage 3
- UserName Property 5
 - Customizing a Login Dialog Box 5
 - NICLgn Control 20
- UserPassword Property 5
 - Customizing a Login Dialog Box 5
- Using the Built-in Login 5
 - LoginDlgFlag Property 5
- Using the Group Object Component 11
- Using the Simple Object Component 7

V

Verification

- Design Time Licensing 2

- Runtime Licensing 2

Volume 3–5, 7–8, 13–27, 10–12

Volume Property 5

- Customizing a Login Dialog Box
5

- NICObj Control 25

VolumeName Property 7, 8, 11, 12, 13

- Creating a Group Object 11

- Creating a Simple Object 7, 8

- NICGrp Control 34

- Removing an Object from a Group
13

- Retrieving a Group Object 12

- Retrieving a Simple Object 8

W

Working with the ObjectHandle

- Property 9