AUTHORIZED DOCUMENTATION

Introduction to Novell Script for NetWare®

Novell®
Developer Kit

June 21, 2006

www.novell.com

Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. Please refer to www.novell.com/info/exports/ for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Novell, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.novell.com/company/legal/patents/ and one or more additional patents or pending patent applications in the U.S. and in other countries.

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the online documentation for this and other Novell products, and to get updates, see www.novell.com/documentation.

Novell Trademarks

For a list of Novell Trademarks, see Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmlist.html)

Contents

Introduction to Novell Script for NetWare

Compatibility. Simplicity. Power. These are the strengths of Novell Script for NetWare (NSN). If you've ever programmed in Visual Basic or VBScript, you will soon find yourself using NSN to write Common Gateway Interface (CGI) scripts, put intelligence into your Web pages, and build server-side task automation applications for NetWare.

NSN incorporates predefined NetWare components with a VBScript syntax-compatible scripting language. It is an object-oriented development environment that supports both the highly skilled RAD programmer and the high-level Web designer.

You can now develop the Web pages using Microsoft* FrontPage and publish them on a NetWare server. See "FrontPage Compatible Components" for details.

NSN replaces the NMX architecture of NetBasic 6.0 with the Universal Component System (UCS), a unified scripting and component architecture. The UCS kernel allows you to write solutions using and combining NSN components, JavaBeans, Perl scripts, and other popular scripting and component languages.

Server Scripting

Scripting languages and visual builder tools have gained popularity in recent years because of their ease of use in delivering content to the Web. NSN is designed to emulate the server-side use of VBScript. Because NSN is VBScript syntax-compatible, it can run scripts built on other platforms. It supports a variety of scripting tools to enliven Web content while leveraging the skills a high-level developer requires in a visual development environment.

Rapid Application Development

Rapid Application Development, or RAD, provides accelerated development of applications using encapsulated objects executed with scripts. A RAD component is an encapsulated, abstracted unit of functionality that acts as a building block for a RAD visual builder or scripting language. By using predefined objects, a developer can drastically raise the quality of finished systems while reducing the time it takes to build them. RAD is now widely accepted as a standard for better software quality and security, and RAD can improve and accelerate software development. NSN provides a component framework to encapsulate basic application functionalities within the NetWare environment.

Concepts

1

This section discusses the key Novell Script for NetWare (NSN) concepts, such as:

- Compiler and how to compile NLMs.
- Console commands used in NSN
- Universal Component System (UCS) and accessing UCS from NSN.
- Differences between NSN and its predecessor, NetBasic 6.0.

1.1 Compiler

Novell Script for NetWare includes a script compiler. This can be used to compile scripts into NLMs.Compiler, required tools and the samples are installed under the directory `SYS:NSN\SDK`.Compiler can be used to compile a script into a regular NLM. Such NLMs can be loaded on the server console. Script can also be compiled to a web based NLM. Such NLMs can be loaded from the Web browser. Only those methods which are exported can invoked from the web browser.

The NSN Compiler project file (`.prj` file) is an ASCII file designed based on the conventions of a standard `INI` file. This file contains application sections and the various keys associated with these applications.This can be created and maintained using any standard text editor.

The 5 application sections of the NSN Compiler Project File are:

- “[Common] Application” on page 9
- “[Compiler] Application” on page 10
- “[Linker] Application” on page 10
- “[NLMOpts] Application” on page 13
- “[NLMLink] Application” on page 14

You can edit the sample `.prj` files provide in “Guidelines for Creating Compiler Project Files” on page 15 to suit your requirements and compile your scripts

1.1.1 [Common] Application

This application section stores the keys which are not related to the compile or link phase of NSN NLMs (NetWare Loadable Modules). The keys available in this section are:

- “Include” on page 9
- “Import” on page 10
- “ExePath” on page 10

Include

This key is reserved and not in use now.

Import

This key contains the path to the NSN Compiler import directory. Both the compiler and the linker search for import files and startup object modules in the import directory, based on the path mentioned in this key . The syntax is `Import=<Directory Path>`.

ExePath

This key contains the path to the NSN Compiler tools. The compiler uses this path to locate the linker, script compiler and pack utilities. The syntax is `ExePath=<Directory Path>`.

1.1.2 [Compiler] Application

This application section stores the keys needed by the NSN Compiler, during script compilation. These keys identify the compiler specific flags and source file to be compiled. The keys are defined as:

File1
File2
...
Filen

File1- Filen contain the full path of the NSN script for which NLM is to be generated. Additional modules can be compiled by entering corresponding file statements in this section. The syntax is `File1=<Script name>`.

1.1.3 [Linker] Application

This application section stores all the keys related to the linking of the NSN script. These keys relate directly to the type of NSN NLM being created.

Keys common to Generic and UCX NLMs

The following keys apply to both Generic NLMs and UCX Component NLMs.

- “Program” on page 10
- “NoLink” on page 11
- “NoDefFile” on page 11
- “EncryptType” on page 11
- “NLMType” on page 11

Program

This key allows you to specify the message prefix that will be used on all messages printed to the System Console by the startup code. The message prefix should have a maximum of 11 characters. The syntax is `Program=<Message>`.

NoLink

This key intimates the compiler not to invoke the NLM linker. This is normally used if the NLM is to be linked later, or if the linker definition file needs modification. This key can be activated by setting NoLink = 1. The syntax is `NoLink=<value>`.

NoDefFile

This key intimates the compiler not to create a linker definition file. This key is used only when the developer customizes a build or generates linker definition file. This key can be activated by setting NoDefFile = 1.The syntax is `NoDefFile=<value>`.

EncryptType

This key intimates the compiler the type of encryption to be used. Currently, only two types are supported: FULL or NONE. For FULL encryption, use EncryptType = 1. For NONE encryption set EncryptType = 0. The syntax is `EncryptType=<value>`.

NOTE: It is advisable to use full encryption, as the code will be stored completely in the NLM executable.

NLMType

This key intimates the compiler the type of NLM that is to be created. Set NLMType = 0, for creating a generic NLM. Set NLMType = 1, for creating an UCX component. The Syntax is `NLMType=<value>`.

Keys used by Generic NLMs

The following keys can be used only for generic NLMs.

- "ScreenName" on page 11
- "UnloadType" on page 11

ScreenName

Specifies the screen name for the NLM. This is the name that is displayed on the highlighted menu bar when the <ALT> key is pressed at the System Console, as well as in the menu list when <CTRL-ESC> is pressed. The syntax is `ScreenName=<Name of the NLM screen>`.

UnloadType

Specifies the unload type for the generic NLM.The syntax for UnloadType is `UnloadType=#`, where '#' is 0, 1, 2 or 3. It currently supports the following types listed in Table 1.

Table 1-1 *Provides a list of values for the Unload type and their descriptions.*

Value Type	Short Description	Long Description
0	Unload immediate	Uses a SIGTERM signal handler for this. The handler is part of the internal NSN startup code.

Value Type	Short Description	Long Description
1	Unload with delay	This option will raise a signal in the NSN NLM that will cause it to exit immediately upon completion of the current line. This option prints a message on the system notifying the operator that the request is being processed.
2	Unload with Signal	This option will raise a signal .NSN NLM monitors this signal and exits after detecting this signal. This option is useful for NSN NLMs that need to perform some level of cleanup before exiting. Like Unload with Delay, this option prints a message on the SystemConsole notifying the operator that the request has been received, and will be processed.
3	Never Unload	This option is used when you do not want to allow the operator to unload the NLM. This option will display a message notifying the operator that the NLM cannot be unloaded from the Console and take the appropriate action to prevent the unload from occurring.

Keys used by UCX NLMs

The following keys can be used only for UCX component NLMS.

- “NBClass” on page 12
- “NBMsg1” on page 12
- “NBMsg2” on page 13
- “NBCopyright” on page 13
- “NBExportAll” on page 13
- “NBExport#” on page 13

NBClass

Specify the UCX class name. The syntax is `NBClass=<class name>`.

NBMsg1

NBMsg1 is used to specify the banner that is displayed when the UCX library manager loads an UCX component. This banner is displayed on the UCX Library Manager screen, along with NBMsg2 and NBCopyright.

NBMsg2

NBMsg2 is similar to NBMsg1.For more details see “NBMsg1” on page 12.

NBCopyright

NBCopyright is similar to NBMsg1.For more details see “NBMsg1” on page 12.

NBExportAll

NBExportAll intimates the compiler to export all subroutines in the NSN script as callable UCX APIs. The syntax is `NBExportAll=<Value>`.

NBExport#

Individual subroutines can be exported using the NBExport# key. The syntax for this key is `NBExport#=<Subroutine name>`, where # is the next sequential number to be used.

Example

If your NSN script had three subroutines: MAIN, SUB1 and SUB2, and you only wanted to export MAIN and SUB2, you would place the following keys in the project file:

```
NBExport1=MAIN
NBExport2=SUB2
```

1.1.4 [NLMOpts] Application

The following keys are used for setting the properties of the NLMs.

- “Description” on page 13
- “Name” on page 13
- “StackSize” on page 14
- “ThreadName” on page 14
- “Version” on page 14
- “CopyRight” on page 14
- “Map” on page 14

Description

Description is used to set the NLM description displayed by the NetWare OS Loader when the NLM is loaded. The syntax is `Description=<name>`.

Name

Name is used to specify the module name of the NLM. If you are creating an UCX component NLM. The syntax is `Name=<Module name>`

StackSize

StackSize is used to set the stack size (in KB) of the main thread for the NLM. it depends on the kind of applications.For database applications it is set to 64 KB. For Generic NLMs, it should be no less than 32 KB. For UCX component NLMs the stacksize can be 16KB. The syntax is `StackSize=<Size>`

ThreadName

ThreadName is used to specify the thread naming pattern used by CLIB when a new thread is created. The syntax is `ThreadName=<name>`.

Version

Version is used to specify the version of the NLM, and is specified in three parts: major, minor, revision. The NLM version is displayed by the NetWare OS Loader when the NLM is first loaded.

The syntax is `Description=<Major number, Minor number,revision number>`

Example

To set the NLM version to 1.5c, the code will be :

```
version = 1,5,3
```

CopyRight

Copyright is used to specify the copyright string for the NLM. This string is displayed by the NetWare OS loader when the NLM is first loaded into memory. The syntax is `CopyRight=<CopyRight String>`.

Map

Map is used to specify whether you want the linker to create a `.MAP` file for the NLM. The syntax is `MAP=#`, where # = 0 for NO MAP, and # = 1 to Create a map file.

1.1.5 [NLMLink] Application

This application section stores the keys used by NLMLINKX. This allows you to pass linker directives to `NLMLINKX.EXE` for a specific project having special needs. The keys are specified using the following syntax:

The value of each key is written "as-is" to the linker definition file for the project. Invalid syntax in this section will cause errors during the link phase of the project. For a full description of all supported NLMLink keywords, refer to the chapter entitled NLMLink User’s Guide****, in this manual. For example, to include an external object module called CUSTOM.OBJ, you would add the following to your `.PRJ` file: [NLMLink] NLMOption1=INPUT CUSTOM.OBJ Be sure that the options are specified sequentially, and no gaps are left in the numbering sequence. For example, if you have added NLMOption1= and NLMOption3 = to this file, only the option specified in NLMOption1 would be included. It is recommended that you keep the options ordered sequentially, to make it easier to determine what the next sequential number should be. You can also include additional object modules in this application area.

1.1.6 Guidelines for Creating Compiler Project Files

1. Create a `.prj` file using the application keys discussed in the previous sections. For more details refer to "Sample Project Files" on page 15.

   NOTE: The `.prj` file and the script for which the NLM is to be generated can be stored under `SYS:\NSN\SDK\BAS\GENERIC` or `SYS:\NSN\SDK\BAS\WEBNLM` folders.

2. Run MS-DOS from your client and set the path to the `NSNCOMP` directory.
3. Run the command `nsncomp <prj file>` from the MS-Dos prompt. The NLM with the module name mentioned in the .prj file will be generated and will be stored in the same directory as the `.prj` file.
4. To run the NLM from the system console copy it to `SYS:\SYSTEM` folder and type the name of the NLM on the system console.

Sample Project Files

Sample for creating Generic NLM

This file generates *print.nlm* based for the script *print.bas*

```
[Common]
Include=.\;..\..\INCLUDE\
;Sets path to the NSN Compiler import directory
Import=..\..\IMP\
;Sets the path to the NSN Compiler tools
ExePath=..\..\TOOLS\
[Compiler]
;Sets the script for generating the NLM
File1=PRINT.BAS
[Linker]
;sets the message prefix to Print
Program=PRINT
;Intimates the compiler to invoke the linker
NoLink=0
;Allows the user to create linker definition file
NoDefFile=0
;Sets FULL encryption during compilation
EncryptType=1
;Creates a Generic NLM
NLMType=0
;Sets the name of the NLM screen
ScreenName=PRINT NLM Screen
; Performs immediate unload after the execution of the NLM
UnloadType=0
;Sets the UCX class name to PRINT
NBClass=PRINT
;Sets the contents of the displayed banner
NBMsg1=Test Component
;Sets the copyright details
NBCopyright=Copyright(c)1997 Novell Inc. All Rights Reserved.
;Does not export all the subroutines in the script as callable ;UCX APIs
NBExportAll=0
;Exports the subroutine Main
NBExport1=Main
[NLMOpts]
;Sets the NLM description
Description=PRINT NLM
;Sets the NLM module name
Name=PRINT
;Sets the stack size for the main thread in the NLM
StackSize=32
;Sets the thread naming convention used by CLIB
ThreadName=PRINT__P
;Sets the NLM version
Version=7,0,0
;Sets the copyright string for the NLM
Copyright=Copyright (c) 1998 Novell Inc. All Rights Reserved.
;Intimates the linker not to create a .map file
Map=0
; sets ****
Pack=0
```

Sample for Compilation of Script into UCX NLM

```
[Common]
Include=.\;..\..\INCLUDE\
Import=..\..\IMP\
ExePath=..\..\TOOLS\
[Compiler]
File1=WEBSCRPT.BAS
[Linker]
Program=WEBSCRPT
NoLink=0
NoDefFile=0
EncryptType=1NLMType=1
ScreenName=WEBSCRPT NLM Screen
UnloadType=0
NBClass=WEBSCRPT
NBMsg1=Test Component
NBCopyright=Copyright(c)1997 Novell Inc.All Rights Reserved.
NBExportAll=0
NBExport1=Main
NBExport2=Login
Version=7,0,0
Copyright=Copyright (c) 1998 Novell Inc. All Rights Reserved.
Map=0
Pack=0
[NLMOpts]
Description=WEBSCRPT NLM
Name=WEBSCRPT
StackSize=32
ThreadName=WEBSCRPT__P
```

1.2 Console Commands Used in NSN

The following are the console commands that are supported by NSN. These commands should be run from the server console.

1.2.1 Commands For UCX Components

UCXSTOP <Component name>

`UCXSTOP<Component_name>` De-registers and unload the UCX component NLM.

Example

`UCXSTOP UCX:NWDIR` will de-register and unload the NWDIR component NLM.

UCXDelay <Component name><Time>

`UCXDelay<Component_name><Time>` Changes the idle time for the component to unload. Component name is the name of the component for which the unload time has to be changed. Time is the delay time specified in minutes. By default, any UCX component will be unloaded automatically after 5 minutes.

Example

`UCXDelay UCX:DIR 20` will unload the NWDir UCX component after 20 minutes of idle time.

UCXSCREEN <ON/OFF>

`UCXSCREEN<ON/OFF>` Displays or conceals the UCX Library Manager screen. ON displays the screen; OFF will not bring up the screen. By default, the UCX Library Manager screen is not displayed.

1.2.2 Commands for Infinite Web scripts

NSN CGI interface has following commands which help in detecting and stopping the infinite scripts.

CGI2NSN STOP

`CGI2NSN STOP` displays all the web scripts.

Example

```
CGI2NSN STOP
```

CGI2NSN STOP <N>

`CGI2NSN STOP<N>` stops the Nth script.

Example

`CGI2NSN STOP 10` stops the 10th script.

CGI2NSN STOP *

`CGI2NSN STOP *` stops all the scripts.

Example

```
CGI2NSN STOP *
```

1.3 Accessing UCS From NSN

NSN uses Novell Universal Component System (UCS) technology as a universal publish-and-subscribe model for running scripts and scriptable components around the network.

The UCS layer allows you to write scripts and reuse components from different development environments to access and use NetWare services.

The UCS model is in contrast to the NMX technology used in NetBasic 6.0, which interprets a Basic-like language on NetWare.

NSN scripts can make use of UCX components, Java classes and Beans. UCX components are native components of NSN. NSN accesses Java classes and Beans using Universal Component System (UCS).

Using UCS components in a NSN script is similar to using any native UCX component. The class argument of "CreateObject function" on page 19 is used to create objects of the required type.

1.3.1 CreateObject function

Creates a scriptable object. Following are the conventions for class names:

For Accessing UCX components

To access UCX components use the prefix "UCX:"

```
CreateObject("UCX:ComponentName")
```

For Accessing Java classes and Bean

To access Java classes and Beans use the prefix "Java:"

```
CreateObject("Java:ComponentName")
```

NOTE: Any class name without a prefix defaults to Java classes or Beans.

Parameters

ComponentName

String. Represents the component that is to be used.

Example

This sample accesses UCS from NSN and creates a Hash table using Java classes.

```
' This sample shows how to use Java classes in a Novell Script
' This sample uses java.util.Hashtable class
'
Sub main()
Set x = CreateObject("java.util.Hashtable")
x.put 1, True
x.put 2, "Apples"
x.put 3, 3.142

For k = 1 To 3
  Value = x.get(k)
  Print "The value for key " & k; " is: " & Value
  Next
End Sub
```

See Also

For more details about UCS architecture, naming conventions, samples and limitations, see *UCS documentation* (http://developer.novell.com/ndk/doc/ucs/index.html?ucs__enu/data/hgtu7g1y.html).

1.4 NSN vs. NetBasic 6.0

NSN uses an interpreted language that is compatible and consistent with VBScript and ANSI Basic standards. It provides a common scripting solution to build dynamic Web pages and develop distributed server-side applications using NetWare services. Because NSN is both Basic and VBScript syntax-compatible, developers can leverage their Basic, Visual Basic and VBScript programming skills while utilizing the power of NetWare services.

For more information, see the Microsoft Scripting Web page (http://msdn.microsoft.com/scripting).

1.4.1 NSN Example

```
'Example to list all the files and subdirectories
'of the current working directory.
'
Set nwf = createobject("UCX:NWFileMgr")
Print "Following are the files and subdirectories in " &nwf.currentDir.name
For each entry In nwf.currentDir.getChildren
      Print entry.Name
Next
```

1.4.2 NetBasic 6.0 Example

```
Name="MyFile"
Ext="txt"
Obj=DOS:Dir:First("*.*",0)
Do While(Obj.Error=0)
     if ((Name=Obj.Name)&(Ext=Obj.Extension))
          Print("Found:",obj.Name);Newline
          DOS:Dir:End(Obj)
          Exit
     Endif
     Obj=DOS:Dir:Next(Obj)
EndDo
```

Tasks

2

This section describes Novell Script for NetWare (NSN) set up procedures and implementation tasks.

These tasks are not solution-oriented. They are general procedures that you should follow to best optimize your implementation of NSN. Most of these tasks consist of suggested uses for NSN and are intended to be viewed as launching points for your NSN projects rather than the solutions to your system needs.

The following tasks are discussed here:

2.1 Installing and Running NSN

Novell Script for NetWare (NSN) features a Win32 client-side installation, set up by downloading and executing a self-extracting.EXE file. Follow these steps to install and run NSN on your system.

1. Go to the NSN Details page and click the *Download* icon to download the self-extracting executable file (`nscript.exe`). *Note: Make sure you have Admin rights on the target server with a mapped drive.*
2. Extract the file and run `setup.exe`, choosing the appropriate drive.
3. At the NetWare console command line, enter `Load NSNINIT.NLM`.
4. To run the NSN shell, at the NetWare console command line, enter `NSNSHELL`.
5. To run an NSN script from a NetWare console command prompt, enter `NSN <script>`.Or, to run a script from a Web browser, enter `http://myserver/nsn/<script>`.

NOTE: While upgrading from NW65 SP3 to later, `sys apache2/conf/mod_nsn.conf` file is backed up to `sys apache2/conf/mod_nsn.sp`

2.2 Modifying NSNSYS.INI File

The NSNSYS.INI file contains NSN configuration information. The first example below shows the NSNSYS.INI file as it ships with this NDK. The second example shows how you can modify the NSNSYS.INI file to fit your needs.

Note: A semicolon at the beginning of a line represents a comment.

2.2.1 Shipping Version Of NSNSYS.INI

```
[NSN]
StackSize=32768

;[WEB]
;path=SYS:nsweb;
; Environment Settings Section For NSN Shell

[ENVIRONMENT]
; Path. Semicolon can be used as separator.
; Default : SERVER\SYS:NSN\USER;SERVER\SYS:NSN\INCLUDE;SERVER\SYS:UTIL

PATH=
; Shell Prompt. Similar to setting DOS prompt.
; Default : $l$f$g$_$v$p$g

PROMPT=
; Location for creating temporary files.
; Default : SERVER\SYS:NSN\TEMP

TEMP_PATH=
; Drive mappings
; S drive.
; Default: SERVER\SYS:

DRIVE_S=
; Z drive.
; Default SERVER\SYS:PUBLIC

DRIVE_Z=
```

2.2.2 Modified NSNSYS.INI

```
[NSN]

; This is the maximum stack size a NSN script can use for a thread.
;
StackSize=32768
; [WEB]
; Search Path for NSN Web called scripts
;path=SYS:nsweb;

[ENVIRONMENT]
; Environment Settings Section For NSN Shell. This is similar to the
; DOS environment. Values here are created with the SET command.
; Using the SET command alone will display the current environment.
; Search Path for NSN Shell. Semicolon can be used as separator.
; Default : SERVER\SYS:NSN\USER;SERVER\SYS:NSN\INCLUDE;SERVER\SYS:UTIL

PATH=
; Shell Prompt. Similar to setting DOS prompt.
; Default : $l$f$g$_$v$p$g
; $p = Current drive/directory
; $g >
; $l <
; $b |
; $$ $
; $t Time
; $d Date
; $v Volume
; $f Server
; $_ Go to next line
; $n n is the character you want to display.

PROMPT=
; Location for creating temporary files.
; Default : SERVER\SYS:NSN\TEMP

TEMP_PATH=
; Drive mappings
; S drive.
; Default: SERVER\SYS:

DRIVE_S=
; Z drive.
; Default SERVER\SYS:PUBLIC

DRIVE_Z=
```

2.3 Executing Sample Scripts

Sample scripts provided for each property, method and event can be executed using the following steps:

1. Open any text editor such as NOTEPAD.
2. Copy the sample script and paste it in the NOTEPAD.
3. Save the file with .bas or .asp extension, appropriately.

4 Execute the .bas script using the steps specified in Section 2.5, “Executing NSN scripts,” on page 24 and Section 2.7, “Executing NSP scripts,” on page 26.

NOTE: Make sure that .asp files are in `sys:/apache2/htdocs` folder and .bas files are in the `sys:/nsn/web` folder

2.4 Creating NSN Scripts

NSN scripts have the file extension .bas. NSN does not provide an IDE for developing scripting applications; they can be created using any text editor such as NOTEPAD or WORDPAD. A basic IDE such as Visual Basic can also be used. Try creating your first script as follows:

1 Map a drive from the client to the SYS volume of the server.

2 Open NOTEPAD and type the following lines

```
'The first NSN script
Print "Hello world"
```

3 Save this file as hello.bas under `SYS:NSN\USER`.

4 Execute the script:

“From Server Console” on page 24
“From NSN Shell” on page 24
“From Browser” on page 25

2.5 Executing NSN scripts

NSN scripts can be executed in three modes:

- “From Server Console” on page 24.
- “From NSN Shell” on page 24.
- “From Browser” on page 25.

2.5.1 From Server Console

1 Start the NSN environment for executing the .bas scripts, by executing the following commands at the server console:

- `NSNINIT`

  This command loads the `NSNINIT.NLM` and should be typed only once.

- `NSN <scriptname>`

  Example: NSN *hello.bas*. This example executes the script created in Section 2.4, “Creating NSN Scripts,” on page 24.

2.5.2 From NSN Shell

The NSN shell is similar to the DOS environment. NSN shell includes several commands (for example, COPY, CD, MD, RD and DIR) that are the same as DOS commands. These commands are also NSN scripts that can be used for reference and customized as needed. They are located in the `SYS:\NSN\UTIL` directory.

The general convention is that all the utility scripts should be stored under the UTIL directory and all the user-created scripts should be stored under the USER directory.

1 Start the NSN environment by typing the following commands at the server console:

- `NSNINIT`

  This command loads the `NSNINIT.NLM` and should be typed only once.

- `NSNSHELL`

  This command activates the NSN shell with SYS:\> prompt. Any number of shells can be opened by typing this command in the server console.

- Test the configuration by typing dir at the NSN shell. This generates the directory listing of the root directory of the volume SYS.

2 Type script name inside the shell.

Syntax: *scriptname*

Example: *Hello*. This examples executes the script hello.bas created in Section 2.4, "Creating NSN Scripts," on page 24

NOTE: NSN\UTIL and NSN\USER directories are included in the search path. If the file is saved under a different directory then the working directory should be changed or the absolute file path should be used to execute the script.

Example: If the script is stored under `SYS:\NSN\SCRIPT\SAMPLES` the working directory or the absolute file path should be set to that path before executing the script by typing CD `\NSN\SCRIPT\SAMPLES` in the shell. Then the shell will be displayed as `SYS:\NSN\SCRIPT\SAMPLES\`.

2.5.3 From Browser

1 To execute a script from the browser the script or the subdirectory containing the script should be in the `SYS:\NSN\WEB` directory.

2 Type the URL, of the form *http://<Servername>/nsn/<Destinationfolder>/<scriptname>*

Example: *http://testserver/nsn/hello.bas*

3 The URL should contain the full path of the script, when the script is present in a subdirectory under the folder WEB.

Example: *http://testserver/nsn/test/hello.bas* where test is a subdirectory created by the user under `SYS:\NSN\WEB`.

2.6 Creating NSP Scripts

NSP helps in generation of dynamic content by embedding scripts in HTML. NSP is a clone of Active Server Pages (ASP).

The following information should be considered when creating a NSP.

- All scripts should be embedded within the tags "<%. %>".
- NSP script should always be saved under the directory `SYS:\NOVONYX\SUITESPOT\DOCS` or a sub directory under it.
- Always use RESPONSE.WRITE instead of PRINT to display results using a NSP.

Try creating your first NSP using these steps:

1. Create an HTML file with the following code using any standard authoring tool or text editor such as NOTEPAD.

```
'Create an HTML page
<html>
  <head>
      <title>      helloworld    </title>
  </head>
  <body>
    System time is
  </body>
</html>

'Embed the script in the HTML page
<html>
  <head>
      <title>      helloworld    </title>
  </head>
  <body>
'Embed the code for getting the current time
    System time is   <% = cstr(now) %>
</html>
```

2. Save the script as helloworld.asp under `NOVONYX\SUITESPOT\DOCS`.

2.7 Executing NSP scripts

The NSP script can be executed only from the browser. NSP scripts can be executed only when they have EXECUTE permission. The server administrator should provide EXECUTE rights for the file or the folder containing the scripts. If the EXECUTE rights are not provided, server returns the "Insufficient rights to access the page" error. For more details refer to Section 2.8, “Providing Execute Rights For Folder or File,” on page 26.

To execute an NSP script:

- If the script is saved in the DOCS directory, type the URL in the form http://severname/scriptname in the browser.

  Example: http://testserver/helloworld.asp

  This example executes the helloworld.asp saved under the `SYS:\NOVONYX\SUITESPOT\DOCS` directory in the server testserver.

- If the script is saved in a subdirectory under the DOCS directory, type the URL in the form http://severname/destination folder/script name in the browser.

  Example: http://testserver/sample/helloworld.asp.

  This example executes the helloworld.asp saved under the `SYS:\NOVONYX\SUITESPOT\DOCS\SAMPLE` directory in the server testserver.

2.8 Providing Execute Rights For Folder or File

1. Type the URL in the form https://<SERVERNAME:Port number>
2. Login to the server as the administrator.

3 Go to NetWare Enterprise Web Server by clicking the appropriate button.

4 Click Restrict Access option provided in the left navigation frame.

5 Click `/NOVONYX/SUITESPOT/DOCS`.

6 Click Contents to view the contents of the folder.

7 Click the folder containing the ASP script if the folder is available under the /DOCS directory.

8 If the folder is not present in the DOCS directory, repeat the steps 6 and 7 to navigate to the folder containing the NSP script.

9 Click Access Control List.

10 Check the option EXECUTE and click SAVE to commit the changes.

11 Click Server Administration.

12 Click NetWare Enterprise Web server.

13 Click Apply.

14 Click Apply Changes. The system provides Execute rights for that specific folder and a suitable message is displayed.

2.9 Configure Apache Web Server

Proceed with the following steps to configure Apache Web server to execute NSN scripts and ASP pages.

2.9.1 Configure Apache 1.3 Web server

1 Open the file `sys:apache\conf\httpd.conf`.

2 Enter the following configuration details in `httpd.conf` file.

```
LoadModule lcgi_module modules/mod_lcgi.nlm
<IfModule mod_lcgi.c>
   AddHandler lcgi-script nlm .pl .asp .nsp .bas
   LCGIModuleMap sys:\nsn\lcgi\scrptpgs.nlm .asp .nsp /sp
   LCGIModuleMap sys:\nsn\lcgi\cgi2ucs.nlm .bas /nsn
   ScriptAlias /nsn sys:/nsn/web
</IfModule>
```

NOTE: If the NSN scripts are kept in a different directory enter the corresponding path for ScripAlias.

---

3 Restart the Web server and test the setup by typing the URL similar to

`http://<servername>/nsn/env.bas`

2.9.2 Configure Apache 2.0 Web server

The AMP pattern installation makes the necessary changes to Apache configuration to enable the execution of NSN scripts and Novell Script Pages from Apache 2.0 Web server.You can test the Web server configuration using a browser. To do this, enter a URL using the following format:

`http://`*`server_name`*`:`*`port_number`*`/nsn/env.bas`

This URL displays the Web server environment variables.

If AMP pattern is not selected during server installation, then to enable the execution of NSN scripts and ASP pages from Apache Web server on NetWare, include the following lines in the `httpd.conf` file.

```
include sys:apache2/conf/mod_nsn.conf
```

You can test the Web server configuration using a browser. To do this, enter a URL using the following format:

`http://server_name:port_number/nsn/env.bas`

This URL displays the Web server environment variables.

2.10 Configure Pervasive Database

NSN scripts can access pervasive SQL.2000 database through ODBC. Proceed with the following steps to configure Pervasive.SQL 2000 database.

1. Load the Pervasive database server by typing mgrstart at the server console.
2. Install the Pervasive client on a Windows machine by running `SYS:\PVSW\CLIENTS\WIN\SETUP.EXE` available on NetWare server.
3. From the client start Pervasive control center by clicking *Start* > *Programs* > *Pervasive* > *Pervasive Control Center*.
4. Goto *center* > *click Connect to Server* . Add Server dialogue box is displayed. Add Server dilogue box can also be displayed by clicking the *Connect* icon or right-clicking *Pervasive Servers* > *Connect To Server*.
5. Enter the IP address of the server that is to be connected > click *OK*. The server will be listed under Pervasive servers.
6. Click the *Specified server* > *login* as the administrator of the server.
7. Right-click *database* > *add database*.
8. Enter the database name (DSN name of the new database).
9. Enter the path to datafiles. The path should be entered as <*servername*>\<*volumename*>\<*destination directory*> > *click OK*.
10. To access the tables available in the new DSN, double-click the *new DSN* > *login as administrator*.

Reference Guide

3

NSN provides server components that you can script together to build network applications and provide server-side task automation. These components are executed using the VBScript syntax-compatible language in NSN. They are intended to be used by NetWare programmers and can be consumed by languages such as Perl, C/C++, and Java, as well as NSN.

As a matter of terminology, it is important not to confuse the two acronyms UCS and UCX.

UCS stands for Universal Component System, the conversion layer between RAD languages and output components, while UCX means Universal Component Extension, which is an implementation of the UCS. For clarity, this document uses "UCS" as a standalone term to refer to the overall component system concept, and "UCX component" to describe a specific implementation of the UCS, as in "the Directory UCX component." Where the term "UCX" is dropped for brevity (as in "the Directory component"), the component's denotation does not change.

For a complete list of Microsoft* FrontPage compatible components, see “FrontPage Compatible Components”. You can now develop Web pages using FrontPage and publish them on a NetWare server.

The new “NWDir ” component provides easy access to Novell Directory Services (NDS) from scripting languages. It allows complete access to the management, maintenance, and administrative components associated with NDS programming in the lower level SDK environment.

3.1 Components

3.1.1 Bindery

The Bindery UCX component (Bindery) is a multi-object component that runs on a NetWare server as a NetWare Loadable Module (NLM). It is used to manage the Bindery database on a NetWare database.

For more Details see “Bindery”.

3.1.2 Btrieve

The Btrieve UCX component (Btrieve.Database) is a single-object component used to perform various operations on a Btrieve database including creating, adding and deleting records, and opening files. Btrieve runs on a NetWare server as an NLM.

For more Details see “Btrieve”.

3.1.3 Client Socket

The Client Socket UCX component (NWCliSkt) is a single-object component that is used to connect to a server. It provides an easy mechanism of socket programming and client-server programming for scripting languages on NetWare. This component supports connection-oriented protocols such as Transmission Control Protocol (TCP) and Sequenced Packet Exchange (SPX) in both synchronous and asynchronous modes.

For more details see "Client Socket".

3.1.4 Com Port

The COM Port UCX component (COMPort) is a single-object component that is used to read from and write to serial ports (communications ports).

For more details see "COM Port".

3.1.5 Document

The Document UCX component (Document) is a multi-object component that is used to create HTML documents.

For more details see, "Document".

3.1.6 Dos File Manager

The DOS File Manager UCX component (DOSFileMgr) is a multi-object component that runs on a NetWare server as a NetWare Loadable Module (NLM). It is used to manage DOS file systems by letting the user perform various operations, such as the following:

- Create a file.
- Rename a file.
- Delete a file.
- List information about files and directories.

For more details see, "DOS File Manager".

3.1.7 Environment

The Environment UCX component helps to preserve and modify the NSN shell environment. Environment is a section of memory shared by the NSN interactive shell and any other NSN application initiated by the shell.

For more details see, "Environment".

3.1.8 Error

The Error UCX component (Error) is a single-object component that displays the error generated by any NSN UCX component during the last performed operation. It also resets (Clear) and sets (Raise) the error properties.

For more details see,“Error”.

3.1.9 Fax

The Fax UCX component (Cheyenne.Fax) is a single-object component that provides an easy way to access and use the Cheyenne fax services from any basic script. This component consists of one powerful method, Send, which supports parameters for fax features such as the sender (From), the recipient (To), and a cover page file (CoverText).

For more details see, “Fax”.

3.1.10 File I/O

The File I/O UCX component (FIO) is a multi-object component consisting of the file I/O manager and the file object. The file manager has a method that opens a file and returns the File object. The File object, in turn, has methods and properties to perform file I/O-related operations.

For more details see, “File I/O”.

3.1.11 FileSystemObject

When writing Common Gateway Interface (CGI) scripts or scripts for Active Server Pages (ASP) to build server-side task automation applications, it is important to add, move, change, create, or delete directories and files on the Web server. The FileSystem object (FSO) allows you to process folders and files using object model. This object is intendant to work only on the server side.

For more details see, “File System Object”.

3.1.12 Frontpage Compatible Components

The FrontPage Compatible UCX components are equivalent to ActiveX server side (ASP installable) components, and are compatible with Microsoft FrontPage. These components in Web pages generated using Microsoft FrontPage.

For more details see, “FrontPage Compatible Components”.

3.1.13 FTP File Manager

The File Transfer Protocol File Manager UCX component (FTPFileMgr) is a multi-object component that exposes FTP-related operations. It supports all FTP operations, including

- Transferring a file.
- Renaming a file.
- Getting the working directory.
- Changing the working directory.
- Logging in.
- Logging out.

For more details see, “FTP File Manager”.

3.1.14 Inews

The Inews UCX component allows a developer to write scripts for accessing NNTP (Network News Transfer Protocol) compliant News servers. The components will query a news server for:

- News groups.
- Articles within a specific group.
- Posting articles.
- Replying to a specific group.

For more Details see "Inews".

3.1.15 LPT Port

The LPT Port UCX component (LPTPort) is a single-object component that writes data to the LPT port of the printer.

NOTE: This component is protected by NWSEC security. To use them, disable the security for the respective scripting language by editing `sys:/system/nwsec.ini` file

---

For more details see, "LPT Port".

3.1.16 Netware Data object

The NetWare Data Object UCX component (NDO) provides access to B2Systems SQLIntegrator and Oracle8 database engines. It provides data access from NSN, as well as all other NetWare-supported scripting languages. NDO is implemented as a multi-object component, similar to Microsoft ActiveX Data Object (ADO).

For more details see, "NetWare Data Object".

3.1.17 Netware File System Manager

The NetWare File System Manager UCX component (NWFileMgr) is a multi-object component that performs NetWare file system operations and can repeat these operation over files and directories. It replaces the NetBasic 6.0 DIR NMX component.

For more details see,"NetWare File System Manager".

3.1.18 Novell ScriptPages

The Novell ScriptPages UCX component (NSP) provides functionality similar to Microsoft Active Server Pages (ASP). NSP is a multi-object component that supports the Session, Request, and Response objects.

For more details see, "Novell ScriptPages".

3.1.19 NWDir

The NWDir UCX component runs on a NetWare server as an NLM. This component provides an easy access to Novell Directory Services® (NDS®) from scripting languages.

It provides a set of methods and properties that allow complete access to the management, maintenance, and administrative components associated with NDS programming in the lower level SDK environment.

For more details see, “NWDir ”.

3.1.20 Power Supply

The PowerSupply UCX component (PowerSupply) is a single-object component that is used to read the values of UPS parameters and change their values.

For more details see, “Power Supply”.

3.1.21 Profile

The Profile UCX component (Profile) is a single-object component that is used for configuration setting-related operations (.INI files). It contains two methods: Read and Write.

For more details see, “Profile”.

3.1.22 Queue

The Queue UCX component (Queue) is a multi-object component that provides generic queuing mechanisms. A queue is simply an object that uses a directory on the NetWare server where jobs (such as print jobs) can be stored and manipulated. A job is actually a file in this directory. Though the primary use of queue services is processing print jobs, any file can be queued for any kind of processing.

For more details see, “Queue”.

3.1.23 Semaphore

The Semaphore UCX component (Semaphore) is a single-object component that runs on a NetWare server as an NLM. It provides access to local and global semaphores.

For more details see, “Semaphore”.

3.1.24 Server

The Server UCX component (Server) is a multi-object component that provides information about a NetWare server, including:

- Server statistics information such as network statistics, file system statistics, and disk statistics.
- Client connection information.
- Network board information.
- Ability to log in and log out from server.

- Information about the screens on the file server console.
- Information about the screens on the file server console.

For more details see, “Server”.

3.1.25 Server Socket

The Server Socket UCX Component (NWSrvSkt) is a multi-object component that lets you implement server functionality using either TCP or SPX protocols. The server socket component listens for connections from the client and manages the collection of such client connections. When a connection request is received, it creates a connection object and fires the Connect event.

For more details see, “Server Socket”.

3.1.26 SNMP

The SNMP UCX component (SNMP) is a single-object component that reads the SNMP object and sends an SNMP alert. It runs on NetWare as an NLM.

For more details see, “SNMP”.

3.1.27 Text Window

The Text Window UCX component (TextWindow) is a multi-object component that lets you create and manage windows, menus, and menu items on a NetWare server screen.

For more details see, “Text Window”.

3.1.28 Volume Manager

The Volume Manager UCX component (VolumeMgr) is a multi-object component that lets you perform such functions as

- Retrieving volume information.
- Enumerating volumes.
- Mounting and dismounting volumes.
- Modifying user space restrictions on volumes.

For more details see, “Volume Manager”.

3.1.29 Zip File

The Zip File UCX component (ZipFile) is a single-object component that is used to compress and decompress a file.

For more details see, “Zip File”.

Secure Scripting

A

This section covers an introduction to the security features, the support for remote file systems, configuration details, the changes required in scripts, and the limitations.

A.1 Introduction

Secure scripting environment is provided for NetWare with the implementation of various security features in Novell Script for NetWare, UCX components, and Perl 5.6 for NetWare.

The security features are implemented in such a way that it preserves the power of script execution from the server console, and it ensures that the Web-based scripts (server-side) can access only resources for which the user has appropriate rights.

The security feature uses Novell eDirectory™. The Web-based scripts should now authenticate to eDirectory with appropriate rights for various operations.

A.2 Script Operations and Security

The following table lists the various operations and the role of security.

Operations	Security Feature
Local file system access - Directory listing, opening files for reading/writing, creation, deletion, moving of directories and files, etc.	Operations are allowed based on the rights of the user. If the script does not log in, these operations will return an error.
Local file system - Above file system activities under web document directory, such as `SYS:NOVONYX\SUITESPOT\DOCS`	These operations are allowed based on the rights of the user. If the script does not log in (public script), read-only access is allowed to the Web documents.
Remote file system access - Directory listing, opening files for reading/writing, creation, deletion, moving of directories and files, etc.	These operations are allowed based on the rights of the user.
Server management - Access to server configuration parameters, loading modules and NCF files, bringing down the server, access to server hardware, and peripherals	Allowed to admin or console operator.
Import functions - Calling functions exported by NLMs	These function calls are performed under the context of the user. Based on the user rights, the call either succeeds or fails.

A.3 Remote File System Support

Security feature is provided for the remote NetWare file system. Existing components are modified to support it. Scripts can access the file systems on other servers, by logging into eDirectory™ using UCX:NWDir component and the FileSystem object.

The resources on the remote NetWare file system are identified using the Universal Naming Convention (UNC). The strings in the UNC format have to be used as arguments for the method/ property call, to access the remote file/directory.

Following is the syntax and examples.

Syntax 1 : \\ServerName\Volume:Path\name
Ex: \\My-Test-Server\DATA:User\mydata\account.txt

Syntax 2: \\IP Address\Volume:Path\name
Ex: \\1.2.3.4\DATA:User\mydata\account.txt

Following example adds a folder to a remote NetWare volume and displays the contents of the directory.

```
' Script to create a directory on remote NetWare volume
'
Set nwdir=CreateObject("UCX:NWDIR")
nwdir.fullname="nds:\\remote-tree\remote-context"
If(nwdir.login ("username","password")) Then
   Print "Login successful"
Else
   Print "Login failed" &err.description
End If

Set FSO = CreateObject("Scripting.FileSystemObject")
Set myFolder=fso.getfolder("\\remote-server\sys:\temp")

Print "Remote folder list: Prior to add"
Set subFolders=myfolder.subfolders
For Each subfolder In subfolders
     Print subfolder.name
Next
Print "<BR>"

Set newfld=subFolders.add ("Test")
Print "Remote folder list: After adding test"
Set subFolders=myfolder.subfolders
For Each subfolder In subfolders
     Print subfolder.name
Next
Print "<BR>"

nwdir.logout
```

A.4 Configuration

By default, the security features are enabled on the system. However, it is an optional feature and you can choose to enable or disable it. Edit the `SYS:SYSTEM\NWSEC.INI` file to enable or disable the security features.

For example, to disable the security features for NSN, modify the `SYS:SYSTEM\NWSEC.INI`, to include the following line under the section [Languages].

```
NSN=OFF
```

NOTE: If the security features are turned off, public access is granted to all the system resources. The decision to disable this feature has to be taken only after carefully considering the consequences.

A.5 Changes Required in Scripting Applications

The changes required in the scripts with the introduction of the security features are minimal.

The following table lists the changes for the scripts.

Script Type	Changes
Console-based scripts - Scripts that are executed from the server console.	No changes. Include script code to login to eDirectory™ using UCX:NWDIR object if access to remote file system is required.
Web-based scripts - Scripts that are executed as an LCGI program, or scripts embedded in HTML pages such as NSP or ASP.	Security features ensure that the users are allowed to perform operations for which they have the relevant rights. For example, a script may not be able to list the content of the SYS volume, until the user is authenticated to eDirectory with appropriate rights. Include script code to login to eDirectory using UCX:NWDIR object. Otherwise scripts have restricted access to resources.

Following is a simple NSN example to demonstrate the need for the script to authenticate with appropriate rights.

An existing Web script that is executed without any rights returns an error.

```
' Script to list the contents of SYS:NSN directory
'
Set FSO = CreateObject("Scripting.FileSystemObject")
Set myFolder=fso.getfolder("sys:nsn")
Print "Directory listing of: " &myFolder.name
Set subFolders=myfolder.subfolders
For Each subfolder In subfolders
     Print subfolder.name
Next
```

The modified Web script does not throw an error as it logs into eDirectory™ with appropriate rights.

```
' Script to list the contents of SYS:NSN directory
'
Set nwdir=CreateObject ("UCX:NWDIR")
nwdir.login("user", "password")

Set FSO = CreateObject("Scripting.FileSystemObject")
Set myFolder=fso.getfolder("sys:nsn")
Print "Directory listing of: " &myFolder.name
Set subFolders=myfolder.subfolders
For Each subfolder In subfolders
     Print subfolder.name
Next

nwdir.logout
```

The following components are updated with implementation details of the security features:

- "Bindery"
- "Btrieve"
- "Client Socket"
- "COM Port"
- "Document"
- "Environment"
- "Error"
- "Fax"
- "File I/O"
- "File System Object"
- "FrontPage Compatible Components"
- "FTP File Manager"
- "Inews"
- "LPT Port"
- "NetWare Data Object"
- "NetWare File System Manager"
- "NWDir "
- "Power Supply"
- "Profile"
- "Queue"
- "Semaphore"
- "Server"
- "Server Socket"
- "SNMP"
- "Text Window"
- "Volume Manager"
- "Zip File"

- "Novell ScriptPages"
- "Legacy BASIC Statements" on page 41

A.6 Limitation

Multiple tree support is not implemented for remote file system access. Once the script logs into an eDirectory™ tree, the script can access the files on any server within this tree. Simultaneous authentication to multiple trees, and accessing resources on different servers are not supported. If there is a need to access different servers, it has to be done in a serial fashion. For example, log in to the tree A, access resources on server A1, and logout; again log in to tree B, access resources on server B1, log out, and so on.

Legacy BASIC Statements

B

NSN uses several non-VBScript convenience functions that are derived from legacy BASIC. The legacy BASIC statements supportedby NSN are:

CLOSE Statement (BASIC)

Closes one or more open files or devices.

Syntax

```
CLOSE [[#]filenumber%[,[#]filenumber%]...]
```

Parameters

filenumber%

The number of an open file or device.

Remarks

CLOSE with no arguments closes all open files and devices.

Example

```
CLS
INPUT "Enter filename: ", n$
OPEN n$ FOR OUTPUT AS #1
PRINT #1, "This is saved to the file."
CLOSE
OPEN n$ FOR INPUT AS #1
INPUT #1, a$
PRINT "Read from file: "; a$
CLOSE
```

END Statement (BASIC)

Ends a program, procedure, block, or user-defined data type.

Syntax

```
END [{DEF | FUNCTION | IF | SELECT | SUB | TYPE}]
```

Parameters

DEF

Ends a multiline DEF FN function definition.

FUNCTION

Ends a FUNCTION procedure definition.

IF

Ends a block IF-THEN-ELSE statement.

SELECT

Ends a SELECT CASE block.

SUB

Ends a SUB procedure.

TYPE

Ends a user-defined data type definition.

Remarks

If no argument is supplied, END ends the program and closes all files.

Example

```
PRINT "Game over."
END
```

INKEY Statement (BASIC)

Reads a character from the keyboard.

Syntax

```
INKEY
```

Parameters

None.

Remarks

INKEY returns a NULL string if there is no character to return. For standard keys, INKEY returns a 1-byte string containing the character read. For extended keys, INKEY returns a 2-byte string made up of the NULL character (ASCII 0) and the keyboard scan code.

Example

```
PRINT "Press Esc to exit..."
DO
LOOP UNTIL INKEY = CHR(27)          '27 is the ASCII code for Esc.
```

INP Statement (BASIC)

Returns a byte read from a hardware I/O port.

Syntax

```
INP(port%)
```

Parameters

port%

A number in the range 0 through 65,535 that identifies the port.

Remarks

INP returns a byte read from a hardware I/O port.

OUT sends a byte to a hardware I/O port.

Example

```
x% = INP(&H3FC)           'Read COM1 Modem Control Register.
OUT &H3FC, (x% XOR 1)      'Change Date Terminal Ready bit.
```

INPUT Statement (BASIC)

Reads input from the keyboard or a file.

Syntax

```
INPUT [;] ["prompt"{; | ,}] variablelist
or
INPUT #filenumber%, variablelist
```

Parameters

prompt

An optional literal string that is displayed before the user enters data. A semicolon after prompt appends a question mark to the prompt string.

variablelist

One or more variables, separated by commas, in which data entered from the keyboard or read from a file is stored. Variable names can consist of up to 40 characters and must begin with a letter. Valid characters are A-Z, 0-9, and period (.).

variable$

Holds a line of characters entered from the keyboard or read from a file.

filenumber%

The number of an open file.

Remarks

INPUT uses a comma as a separator between entries. For keyboard input, a semicolon immediately after INPUT keeps the cursor on the same line after the user presses the ENTER key.

Example

```
CLS
OPEN "LIST" FOR OUTPUT AS #1
DO
     INPUT " NAME: ", Name$ ’Read entries from the keyboard.
     INPUT " AGE: ", Age$
     WRITE #1, Name$, Age$
     INPUT "Add another entry"; R$
LOOP WHILE UCASE$(R$) = "Y"
CLOSE #1
’Echo the file back.
OPEN "LIST" FOR INPUT AS #1
CLS
PRINT "Entries in file:": PRINT
DO WHILE NOT EOF(1)
     LINE INPUT #1, REC$           ’Read entries from the file.
     PRINT REC$                     ’Print the entries on the screen.
LOOP
CLOSE #1
KILL "LIST"
```

OPEN Statement (BASIC)

Opens a file or device.

Syntax

```
OPEN file$ [FOR mode] [ACCESS access] [lock] AS [#]filenumber% [LEN=reclen%]
```

Parameters

file$

The name of the file or device. The file name may include a drive and path.

mode

One of the following file modes: APPEND, BINARY, INPUT, OUTPUT, or RANDOM.

access

In network environments, specifies whether the file is opened for READ, WRITE, or READ WRITE access.

lock

Specifies the file locking in network environments: SHARED, LOCK READ, LOCK WRITE, LOCK READ WRITE.

filenumber%

A number in the range 1 through 255 that identifies the file while it is open.

reclen%

For random-access files, the record length (default is 128 bytes). For sequential files, the number of characters buffered (default is 512 bytes).

Remarks

If the Web-based script is not authenticated to eDirectory with appropriate rights, an error is returned.

Example

```
INPUT "Enter Filename: "; n$
OPEN n$ FOR OUTPUT AS #1
PRINT #1, "This is saved to the file."
CLOSE
OPEN n$ FOR INPUT AS #1
INPUT #1, a$
PRINT "Read from file: "; a$
CLOSE
```

OUT Statement (BASIC)

Sends a byte to a hardware I/O port.

Syntax

```
OUT port%, data%
```

Parameters

port%

A number in the range 0 through 65,535 that identifies the port.

data%

A numeric expression in the range 0 through 255 to send to the port.

Remarks

INP returns a byte read from a hardware I/O port.

OUT sends a byte to a hardware I/O port.

Example

```
x% = INP(&H3FC)           'Read COM1 Modem Control Register.
OUT &H3FC, (x% XOR 1)      'Change Date Terminal Ready bit.
```

PRINT Statement (BASIC)

Writes data to the screen or to a file.

Syntax

```
PRINT [#filenumber%,] [expressionlist] [{; | ,}]
```

Parameters

filenumber%

The number of an open file. If you do not specify a file number, PRINT writes to the screen.

expressionlist

A list of one or more numeric or string expressions to print.

{; | ,}

Determines where the next output begins. Print zones are 14 characters wide.; (semicolon) Print immediately after the last value.

, (comma) Print at the start of the next print zone.

Example

```
OPEN "TEST.DAT" FOR OUTPUT AS #1
PRINT #1, USING "##.### "; 12.12345
CLOSE
OPEN "TEST.DAT" FOR INPUT AS #1
INPUT #1, a$
PRINT a$
LPRINT "This is a line"; 1
LPRINT "This is a line",
LPRINT 2
```

SOUND Statement (BASIC)

Generates a sound through your computer's speaker.

Syntax

```
SOUND frequency, duration
```

Parameters

frequency

The frequency of the sound in hertz; a value in the range 37 through 32,767.

duration

The number of system clock ticks the sound lasts; a value in the range 0 through 65,535. There are 18.2 clock ticks per second.

Example

```
FOR i% = 440 TP 1000 STEP 5
SOUND i%, i% / 1000
NEXT i%
```

WAIT Statement (BASIC)

Suspends program execution.

Syntax

```
WAIT [seconds&]
```

Parameters

seconds&

Number of seconds to suspend the program. If seconds& is 0 (zero) or is omitted, the program is suspended until a key is pressed or a trapped event occurs.

Example

```
PRINT "Taking a 10-second nap..."
WAIT 10
PRINT "Wake up!"
```

UCX Component Documentation Structure

C

Each of the 29 components listed under Section 3.1, “Components,” on page 29 under Chapter 3, “Reference Guide,” on page 29 is accompanied by an object diagram that shows the relationship between that component's objects. Below each object diagram is a navigation table that lists all the properties, methods, and events.

The chapter for each component defines the top-level object first, followed by all sub-objects. Object definitions are displayed in order as close as possible to their sequential structure. In other words, collections are displayed before the objects that belong to them, and objects closest to the top-level object are described first.

All the properties, methods and events related to a particular object are described under a single chapter that gives all the details about this object.The order of description is:

1. Properties
2. Methods
3. Events

C.1 Properties

This provides the following details about each property that are explained in this documentation:

C.1.1 Syntax

The syntax of the command used for that property. [=] indicates that the property has READ-WRITE

Example:

ItemBackColor[=ColorConstant As String]

C.1.2 Type

The datatype of the property.

C.1.3 Attributes

The attributes associated with the property. A property can have the READ attribute or the WRITE attribute or the READ-WRITE attribute.

C.1.4 Remarks

Additional information related to the property.

NOTE: If this section is not available for a property or a method then there are not additional information related with the same.

C.1.5 Example

A sample script using the described property that can be executed by copying and pasting in a text editor. For more details see Section 2.3, “Executing Sample Scripts,” on page 23.

C.1.6 See Also

Links to other properties, methods and events that are related to the discussed property.

C.2 Methods and Events

This documentation provides the following details about each method and event:

C.2.1 Syntax

The syntax of the command used for that method or event.

All method or event names are terminated with a pair of parentheses which contain any valid parameters.

In syntax descriptions, square brackets are used to denote optional parameters, as in *dir.timeout [Timeperiod as Integer]*. In this case, if the time out period is not mentioned, the default time out period is used during script execution.

C.2.2 Parameters

Describes the parameters required for the method or event. If there are no parameters required, it is stated as void.

C.2.3 Return Values

The datatype of the return value of this method or event. If there is no return value, it is stated as void.

C.2.4 Remarks

Additional information related to the method/event.

C.2.5 Example

A sample script using the described method or event that can be executed by copying and pasting in a text editor. For more details, see Section 2.3, “Executing Sample Scripts,” on page 23.

C.2.6 See Also

Links to other properties, methods and events that are related to the discussed property.

About NSN Documentation

D

The NSN documentation comprises of 4 books.

1. The first book contains the following chapters.
   - "Concepts" on page 9 of NSN
   - "Tasks" on page 21 - provides details about the tasks to be performed for installing NSN, and executing NSN and NSP scripts
   - "Reference Guide" on page 29 - provides details about the UCX components supported by NSN.
   - "Legacy BASIC Statements" on page 41 - provides details about the legacy BASIC supported by NSN.
   - "UCX Component Navigator" on page 59 - includes a navigation table that gives the list of UCX components and the related objects.
2. The second book NDK: NSN Components - Part One (A-L) gives the details about the UCX components (A-L) and the related objects.
3. The third book NDK: NSN Components - Part Two (N-Z) gives the details about the UCX components (M-Z) and the related objects.
4. The fourth book NDK: UCX SDK Components gives the details about the UCX SDK API that are supported by NSN.

UCX Component Navigator

E

Refer the following table for the list of all UCX components supported by NSN and the related objects.

Component Name	Related Objects
“Bindery”	“Bindery Object”
	“Entries Collection”
	“Entry Object”
	“Fields Collection”
	“Field Object”
	“NetAddress Object”
“Btrieve”	“Btrieve Object”
“Client Socket”	“NWCliSkt Object”
“COM Port”	“COMPort Object”
“Document”	“Document Object”
	“CGIObjs”
	“CGIObj ”
	“EnvObjs”
	“User Object”
“DOS File Manager”	“DOSFileMgr Object”
	“Entry Object”
	“Entries Collection”
“Environment”	“Environment Object”
“Error”	“Error Object”
“Fax”	“Send method”
“File I/O”	“FIO Object”
	“File Object”

Component Name	Related Objects
“File System Object”	“File System Object”
	“TextStream”
	“File”
	“Folder”
	“FilesCollection”
	“FoldersCollection”
“FrontPage Compatible Components”	This list gives a set of Microsoft Frontpage compatible components.
	“Ad (Advertisement) Rotator”
	“Content Rotator”
	“Counter”
	“MyInfo”
	“NextLink”
	“PageCounter”
	“PermissionChecker Component”
	“Tools”
“FTP File Manager”	“FTPFileMgr Object”
	“Entry Object”
	“Entries Collection”
“Inews”	“News Object”
	“Groups Object”
	“Group Object”
	“Articles Object”
	“Article Object”
“LPT Port”	“LPTPort Object”
“NetWare Data Object”	“Connection Object”
	“Command Object”
	“Parameters Collection”
	“Parameter Object”
	“Recordset Object”
	“Fields Collection”
	“Field Object”

Component Name	Related Objects
"NetWare File System Manager"	"NWFileMgr Object"
	"Entry Object"
	"Entries Collection"
	"ParsedName Object"
	"Trustees Collection"
	"Trustee Object"
	"TrusteePaths Collection"
	"TrusteePath Object"
"Novell ScriptPages"	"Session Object"
	"Application Object"
	"Request Object"
	"Response Object"
	"Cookies Collection"
	"Cookie Object"
	"Form Collection"
	"QueryString Collection"
	"ServerVariables Collection"
"NWDir "	"NWDir Object"
	"NWEntries Collection"
	"NWEntry Object"
	"NWFieldDescriptions Collection"
	"NWFieldDescription Object"
	"NWFilter Object"
	"NWLayoutDescriptions Collection"
	"NWFieldDescription Object"
	"NWFieldTypes Collection"
	"NWFieldType Object"
"Power Supply"	"Read method"
	"Write method"
"Profile"	"Read method"
	"Write method"

Component Name	Related Objects
“Queue”	“Queue Object”
	“Jobs Collection”
	“Job Object”
“Semaphore”	“Semaphore Object”
“Server”	“Server Object”
	“Connections Collection”
	“Connection Object”
	“DiskStat Object”
	“FATStat Object”
	“FileStat Object”
	“LanBoards Collection”
	“LanBoard Object”
	“Modules Collection”
	“Module Object”
	“NetAddress Object”
	“NetStat Object”
	“Params Collection”
	“ParamObjects Collection”
	“ParamObject Object”
	“Screens Collection”
	“Screen Object”
	“ServerInfo Object”
“Server Socket”	“NWSrvSkt Object”
	“Connections Collection”
	“Connection Object”
“SNMP”	“ReadObject method”
	“Trap method”
“Text Window”	“TextWindow Object”
	“Cursor Object”
	“Menu Object”
	“MenuItem Object”

Component Name	Related Objects
“Volume Manager”	“VolumeMgr Object”
	“Volumes Collection”
	“Volume Object”
	“VolumeRestrictions Collection”
	“VolumeRestriction Object”
	“ExtendedVolInfo Object”
“Zip File”	“Compress method”
	“Decompress method”

Revision History

F

This section outlines all the changes that have been made to the documentation (in reverse chronological order).

F.1 June 2006

- Updated the Trademarks list to comply with revised Novell documentation standards

F.2 March 2006

- Page design reformatted to comply with revised Novell documentation standards.

F.3 October 2005

- Transitioned to new template

F.4 February 2002

- Added Appendix A, “Secure Scripting,” on page 35
- Updated Section 1.2, “Console Commands Used in NSN,” on page 17

F.5 October 2001

- This book was existing as a part of the book Novell Script for NetWare (NSN) and has been split and re-organized
- Improved the contents for enhanced doc accessibility features