

NS-DK

Version 5.00 Edition 3

Generation Manual

Information in this document is subject to change without notice as a result of changes in the product. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. It is illegal to reproduce the software on any medium unless specifically authorized within the terms of the agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the written permission of Nat Systems.

© 2007 Nat System. All rights reserved.

The Nat Systems name and logo, Adaptable Development Environment, Graphical Builder, Information Modeling, Process Modeling, Version Management and Migration Manager are registered trademarks of Nat Systems.

All other trademarks mentioned in this document are registered trademarks of their respective companies.

Contents

Conventions	v
-------------------	---

Chapter 1 Introduction

Overview	1-3
Files supplied	1-4
NSGEN.EXE	1-4
NSRESLIB.EXE	1-4
NSLIB.H	1-4
NSLIB.LIB	1-4
External Resources.....	1-6
Steps Involved in Building an Executable	1-7
Generation of source files.....	1-7
Compiling	1-8
Linking	1-8
Using the Resource Compiler.....	1-8

Chapter 2 Generating from NS-Design

Introduction.....	2-3
Setup	2-4
Configurations box	2-5
NSDKCFG.INI and NSDKLOC.INI files.....	2-18
Generating, compiling and linking.....	2-19
Build/Run project box	2-19
Generation Process	2-21
Running the Generated Executable.....	2-23

Chapter 3 Syntax for NS-Gen

Syntax	3-3
{options}	3-3
[PrimaryScreen].....	3-8

General points.....	3-9
Displaying the Syntax for NSGEN.....	3-9

Chapter 4 Syntax for NS-ResLib

Introduction.....	4-3
Syntax	4-4
Displaying the Syntax for NSRESLIB	4-6

Chapter 5 Tips for generation

Incremental Building.....	5-4
How an Application is Split Up and the Automatic Make Facility	5-4
Time Taken to Build an Application	5-4
File names and sizes	5-5
When to Regenerate	5-5
Make Facility.....	5-5
Generating an executable	5-7
Generating Source files.....	5-7
Compiling and Linking.....	5-8
Generated Files	5-9
Generating DLLs.....	5-11
Reminder	5-11
Using NS-Gen Directly	5-11
Generated Files	5-12
Using Generated DLLs.....	5-13
Applications Written in C or Pascal	5-14
Applications Written in NCL	5-14
Using DLLs Generated in C.....	5-17
Errors.....	5-20
Errors During the Source Generation Phase.....	5-20
Errors During the Compilation Phase.....	5-21
Errors During the Link Phase	5-22
Application Run-Time Errors.....	5-22

Chapter 6 Tips for running an application

Characteristics of the Main Window.....	6-3
Launching an Application.....	6-4
Files used by an application	6-6
External Resources.....	6-7

Conventions

Typographic conventions

Important term	Important terms are printed in bold .
<i>Interface component</i>	The names of windows, dialog boxes, controls, buttons, menus and options are printed in <i>italics</i> .
[F9]	Function key names appear in square brackets.
FILENAME	Filenames are printed in UPPERCASE.
<code>syntax example</code>	Syntax examples are printed in a fixed-width font.

Notational conventions

- A round bullet is used for lists
- ◆ A diamond is used for alternatives
- 1. Numbers are used to mark the steps in a procedure to be carried out in sequence

definition

A **definition** has a special presentation. It explains the term in a single paragraph. The term appears in the first column, then once in bold in the definition.

Operating conventions

Choose
XXX \ YYY

This means you need to open the *XXX* menu, then choose the *YYY* command (option) from this menu.

You can perform this action using the mouse or mnemonic characters on the keyboard.

Click the
XXX \ YYY
button

This means you need to display the tool bar named *XXX*, then click the *YYY* button in this tool bar (the name of each button is shown by its help bubble).

You can only perform this action with the mouse.

Choose the
XXX button

This means you need to choose the *XXX* button in a dialog box.

You can perform this action using the mouse or mnemonic characters on the keyboard.

Icon codes



Comment, note, etc.



Reference to another part of the documentation



Danger: precaution to be taken, irreversible action, etc.



Suggestion: helpful hints, etc.



To go a step further: level of detail or expertise greater than the average level of the document



Indicates specific information on using the software under DOS-Windows (all versions)



Indicates specific information on using the software under DOS-Windows 32 bits



Indicates specific information on using the software under DOS-Windows 32 bits



Indicates specific information on using the software under Unix systems

Chapter 1

Introduction



This chapter presents NS-Gen and some generalities to use this tool.

You will find in this chapter

- The main components of NS-DK.
- How to generate the external resources.
- How to build an executable.

Contents

Overview.....	1-3
Files supplied.....	1-4
NSGEN.EXE 1-4	
NSRESLIB.EXE 1-4	
NSLIB.H 1-4	
NSLIB.LIB 1-4	
External Resources	1-6
Steps Involved in Building an Executable	1-7
Generation of source files 1-7	
Compiling 1-8	
Linking 1-8	
Using the Resource Compiler 1-8	

Overview

NS-DK's main components -- NS-Design, NS-Test and NS-Debug -- enable you to develop, test and debug your applications.

Once you are satisfied with the progress of your development work, you can move on to the next step: generating an executable program that can run independently of these three components. This program can then be handed over to your end-users and will look and behave like any other program or package already installed on their machines.

Executables are generated by NS-DK's fourth component: **NS-Gen**. It handles all the transitional stages involved in creating an executable file (with optional external resources) from a given project's resources: generating source files, compiling and linking. Therefore, you do not need to be familiar with the language or generation tools used.

Generally speaking, programs that run in a graphical environment (Windows, OS/2 PM, etc.) can be instantly identified in the program list by the icon displayed next to their name. This association is made by a utility called the resource compiler and is totally transparent during the installation of the program.

NS-Gen and the resource compiler provide you, the developer, with the tools you need to deliver a complete program to your end-users.

Files supplied

NSGEN.EXE

NS-Gen generator

This program launches each of the tools required to generate an executable: it generates source files and an external resource file from a project's NS-DK resources (using the NCLGEN.EXE utility). It also compiles the generated source files and links them.

It can be started from NS-Design by selecting the Build command from the Build menu (see Chapter 2) or from a command line (see Chapter 3).

NSRESLIB.EXE

External resource librarian.

This program allows you to list the contents of an external resource file, extract one or more targets from another file or merge several files into a single external resource file.

See Chapter 5 for details on how to use it.

NSLIB.H

Include file required to compile the C source files generated by NS-Gen.

This file contains constant definitions and function declarations used to interface your application with the target system.

NSLIB.LIB

File required to link an application.

This file contains the information used to identify the functions in NSnnLIB.DLL, the NS-DK DLL that handles the display and behavior of graphical objects in the target system. This file also identifies most of the libraries supplied with NS-DK.

This file is an "Import Library". It allows your application to import external functions.

External Resources

External resources are data items that are usually found in an executable but are stored outside the executable to reduce its size.

NS-Gen can generate external resource files. In Version 1.5 of NS-Gen, an external resource file can contain:

- Windows (.SCR)
- Templates (.TPL)
- Bitmaps (.BMP), icons (.ICO), pointers (.PTR) and text files (.TXT) defined with the Preload option in NS-Design
- Constants (.CST).

All other NS-DK resources and the code remain in the executable.

Any generated application can have an external resource file. However, these external resources are mainly useful for applications with several targets since they enable you to have a single executable with several external resource files. One of NSResLib's functions is to extract a target from an external resource file so that end-users only receive the target they require.

The concept of multiple targets can apply to the following types of data stored in a resource file: windows (.SCR), templates (.TPL) and constants (.CST). Bitmaps and text files are target-dependent.



For more information on multi-target applications, see the chapter entitled Multiple Targets in the NS-Design User Manual.

Steps Involved in Building an Executable

Any program that runs on a computer is an executable file.

Executable files are in a binary format known as machine code, which consists of a series of commands that can be interpreted by the computer. The various steps described below generate this code from the code written in a language known as the programming language.

The various steps for generating an executable involve:

- Defining the processing required using a programming language (coding phase).
Several files, known as source files, can be created. Any information that is common to a number of files is stored in separate files known as include files.
- Transcribing the code into machine language (compilation phase).
The compiler generates binary files, known as object files, from the source files. An object file is the result of compiling a source file. Include files are used to define the types and values of the parameters used.
- Linking the object files to produce a single binary file.
This step establishes all the links between object files and functions that are external to the application (e.g. DLLs).

The generated executable file also includes a header which informs the system of the type of program to be executed and any external programs that need to be loaded into memory.

Generation of source files

With NS-DK, applications are developed using the NCL programming language (Natsys Control Language), which allows you to manipulate graphical resources easily.

The first step involves using NS-Gen to generate source files written in a programming language that is supported by the standard compilers available on the market.

The following files are generated in C:

- Source files with the ".C" extension.
- Include files with the ".H" extension (constant definitions and function declarations).
- A definition file with the ".DEF" extension. This contains the program's characteristics.
- An include file with the ".D" extension (for character strings if the /FARDATA option is used).
- External resource files with the ".RES" extension (if the /RESEXTERN option is used).

Compiling

The compilation phase can either be triggered automatically by NS-Gen or carried out by someone who is familiar with the compiler for the language in which the source files have been generated.



In C, this phase produces an object file (.OBJ) for each source file (.C).

If you compile manually, you will need to ensure that the C compiler can access the NSLIB.H file so that it can generate object files. To do this, you should either add the pathname for NSLIB.H to the INCLUDE environment parameter or pass it in the compiler's start-up parameters.

Linking

The linker phase can either be triggered automatically by NS-Gen or carried out by someone who is familiar with the linker for the language in which the source files have been generated.



In C, this phase produces an executable file (.EXE) from all the object files generated by the compiler and the .DEF file produced when the source files were generated.

If you link manually, you should tell the linker to use the NSLIB.LIB file so that it can determine which DLL contains the NS-DK display and processing functions.

Using the Resource Compiler

Use of the resource compiler is optional.

It associates an icon with the executable. This icon will be displayed next to the application name when the executable is included in a program group.



Use the resource compiler supplied with the system.

Chapter 2

Generating from NS-Design



This chapter contains all the information you need to generate source files, external resources, executables or DLLs from NS-Design's main menu.

You will find in this chapter

- How to configure the generator.
- The syntax of the NSDKCFG.INI file.
- The configuration of the dialog boxes *Configurations* and *Build/Run project*.
- How to generate, compile and edit links.
- How the generation process works.
- How to run an executable that has been generated.

Contents

Introduction.....	2-3
Setup	2-4
Configurations box 2-5	
Generator tab 2-5	
Compiler/Linker tab 2-10	
Directories tab 2-13	
DLLs tab 2-15	
NSDKCFG.INI and NSDKLOC.INI files 2-18	
Generating, compiling and linking.....	2-19
Build/Run project box 2-19	
Generation Process 2-21	
Generating source files 2-21	
Compiling and Linking 2-21	
Stopping the Generation Process 2-21	
Generator Block 2-22	
Running the Generated Executable.....	2-23

Introduction

This chapter contains all the information you need to generate source files, external resources, executables or DLLs from NS-Design's main menu.

It describes the various dialog boxes that apply to NS-Gen:

- The *Configurations* box, and the other boxes that it opens, allow you to configure each generation tool.
- The *Build/Run project* box allows you to start the generator and build an executable.
- The *Build/Run project* box allows you to test the generated executable.

If you are generating for the first time, we advise you to read this chapter carefully. By understanding it fully, you can optimize the generation process.

Next, refer to Chapters 6, 7 and 8 for tips on generating and answers to any questions you may have.

Setup

The *Configurations* box is called by the *Options/Configurations* option in NS-Design. It allows you to specify several generation configuration files for each project. A configuration file ends with the .N_C extension and contains the selected settings for each of the tools used to build an executable (generator, compiler, linker and resource compiler). These tools can be configured via this box.

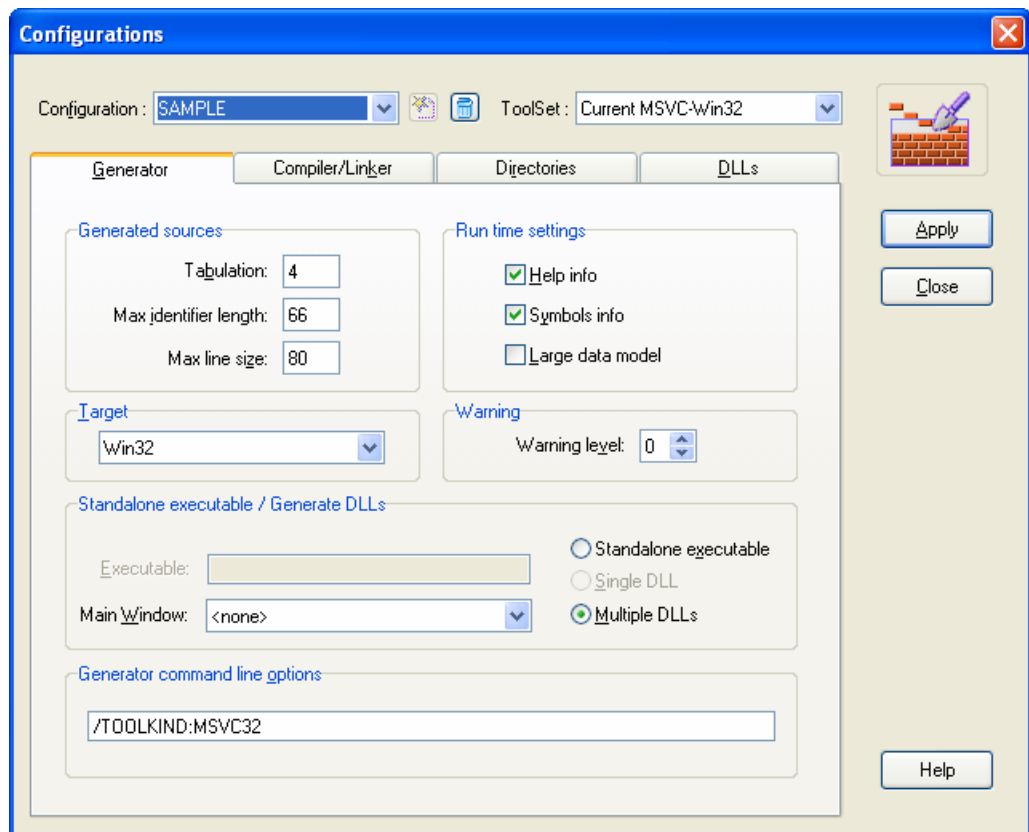
The various toolsets displayed in the *Configurations* box are listed in the NS-DK.INI file (the contents of this file are described at the end of this section).

The following tools and options are used to generate and build an executable:

- Targets
- Working directories
- Executable
- Dynamic libraries
- Generator
- Compiler
- Linker.

Each of the above options is stored in the current .N_C configuration file, i.e. the configuration file selected from the *Configuration* field in the *Configuration* box. The NSDKCFG.INI file simply lists the compilers and linkers that can be used; NS-Design accesses this file in read-only mode.

Configurations box



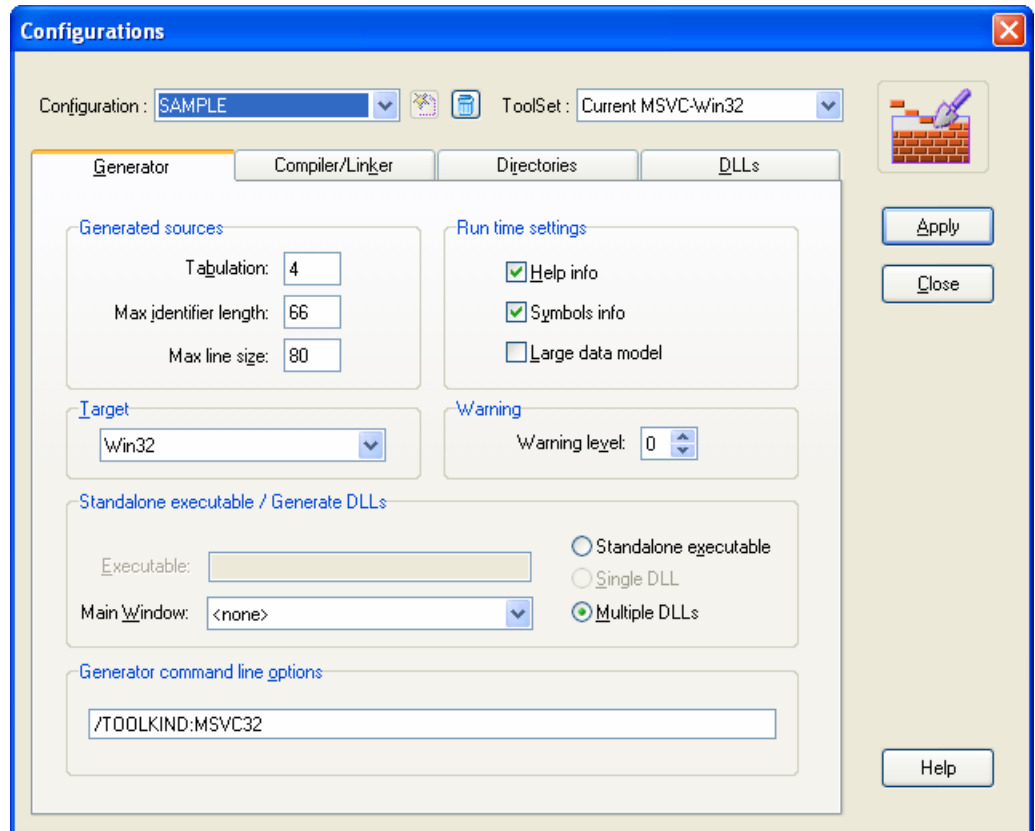
The *Configurations* box, is called by the *Options/Configurations ...* option. It allows you to create or select a configuration file and, for each file, specify the general settings for the build process (*Generator*, *Directories* and *DLLs* tabs) and configure the various tools used (*Generator* and *Compiler/Linker* tabs).

Generator tab

The *Generator* tab lets you configure, for the current configuration file, all the source files generated by NS-Gen: presentation, size, specific code,...



This tab includes the options defined, in previous versions of NS-DK, in the *Project Generation* box accessed via the *Project generation options setup* menu



This tab contains the following fields:

Configuration Displays the current configuration.

Also lets you create or delete a configuration. This field does not let you select a configuration. Selecting the configuration is carried out in the box *Build/Run Configuration*.



The button opens the *Create New Configuration* box.



The button *Delete current configuration* opens a box in which you can confirm you wish to delete the configuration.

ToolSet Displays the toolset associated to the current configuration. You can modify it. This field lists the toolsets defined in the nsdkcfg.ini file.

Generated sources

Tabulation

Number of space characters generated for each indentation of the code in source files.

Default : 4

Minimum : 2

Maximum : 32

Max identifier length

Maximum number of characters that the compiler considers when identifying variables.

When the source code is generated, NS-Gen will restrict variable and control names to this limit. As a general rule, NS-Gen adds a prefix of several characters to the names created in NS-DK. For this reason, you must distinguish the names of the identifiers that you manipulate in NCL over "Max identifier length" minus at least five characters to avoid compiler errors.

The default value displayed is the maximum commonly used by commercially available C compilers.

Default : 31

Minimum : 8

Maximum : 255

Warning level

0 The generator emits a low warning level.

1 The generator emits warnings relating to the ambiguous use of variables or INT(4) type parameters instead of POINTERS, and vice versa.

These are very useful if the user wishes to port an application to platforms on which the integers and pointers are not compatible (e.g.: 64-bit platforms, AS400, etc.).

Run time settings**Help info**

Ensures that your application's on-line help file is opened when the end-user presses F1.

Symbols info

This option informs NS-Gen that you are using the CONTROLNAME\$ and WINDOWNAME\$ functions in the NSMISC library to retrieve control and window names.

In this case, NS-Gen generates internal tables for each window so that your application can supply these names.

If you do not check this option but call the CONTROLNAME\$ and WINDOWNAME\$ functions in the NSMISC library, incorrect results will be returned when the application runs.

16 bits

Large data model

This option warns the compiler that a large number of static variables are used in the application. This results in improved memory handling to accommodate these variables.

We recommend that you check this option in two cases:

- When the overall size of your static variables exceeds 64 Kb (static variables include variables defined in NCL through the GLOBAL type and those that appear in VARIABLE resources).
- When your application is generated as DLLs and they use global variables.

One of the results of using this option is that character strings are grouped together in include files with the ".D" extension.

Target

List of targets defined for the project.

Use this list to select each target that you want to associate with the configuration file. Your selected targets will be highlighted. Any unselected targets will not be included in the generation.

One of these target names will be preceded by a symbol: this is the default target for the configuration file.

Allows you to specify the target(s) that you want the system to include when generating the executable. At least one target must be defined with the Default option.

If you are not generating external resources (see 'Generator' section), only the default target will be generated. However, if you are generating external resources, the default target is the one that will be executed if you launch the executable without specifying a target (i.e. if the /TARGET option is missing).

If you have not created any targets, the default "Generic Target" will be automatically associated with your configuration file.

On the other hand, if your current project has several targets, you can split up the various targets between several configuration files (for more information on multi-target applications, see the chapter entitled "Multiple Targets" in the NS-Design User Manual).

Standalone executable / Generate DLLs

Executable

Name of the executable to generate. This name will also be used for the main source file, which will contain the application's entry point.

The application's entry point is the first function called when the application is launched (for Windows program written in C, this function is called *WinMain*).

Main Window

Name of the first window to be displayed at the start of the application. This name must be choose among the windows of the application.

Standalone executable

Single DLL

Check to indicate there is only one DLL to be generated.

When the project is a library that provides services (functions and instructions) to several applications, this library will be generated in a single DLL.

Multiple DLLs

Check to indicate there are several DLLs to be generated.

Generator command line options

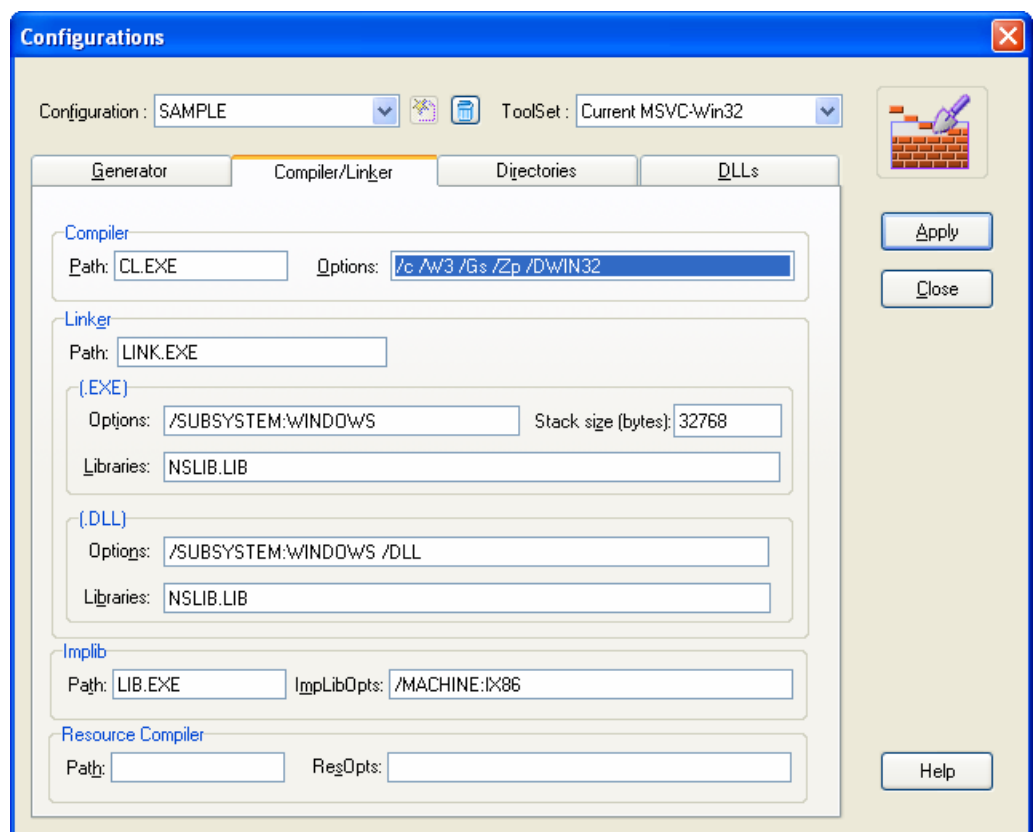
This field allows you to specify other options for NS-Gen (see chapter entitled 'Syntax for NS-Gen').

For example: /TOOLKIND (to indicate the type of tool used).



Apply	Closes the dialog box and confirms any changes made.
Cancel	Closes the dialog box and ignores any changes made.
Help	Displays on-line help for this dialog box.

Compiler/Linker tab

This *Compiler/Linker* tab allows you to set the parameters required by the linker, the ImpLib utility and the resource compiler to generate executables.



This tab contains the following fields:

Configuration	<p>Displays the current configuration.</p> <p>Also lets you create or delete a configuration. This field does not let you select a configuration. Selecting the configuration is carried out in the box <i>Build/Run Configuration</i>.</p> <div data-bbox="663 577 707 622"></div> <p>The button opens the <i>Create New Configuration</i> box.</p> <div data-bbox="663 658 707 703"></div> <p>The button <i>Delete current configuration</i> opens a box in which you can confirm you wish to delete the configuration.</p>
ToolSet	<p>Displays the toolset associated to the current configuration. You can modify it. This field lists the toolsets defined in the nsdkcfg.ini file.</p>
Compiler	<p>Build object files (with the .OBJ extension) from sources files written in C language.</p> <p>Path</p> <p>Full pathname of the resource compiler.</p> <p>Options</p> <p>Start-up parameters for the resource compiler. These depend on the compiler used.</p>
Linker	<p>Configures the linker for building DLLs or executables.</p> <p>Recovers OBJ files created by the compiler, then processes them and at the same time generates the executable and DLLs that a final user will need to run your application.</p> <p>In the <i>Linker</i> group, NS-Design automatically includes the NS-Design libraries whose functions you used in the application in question.</p> <p>Path</p> <p>Full pathname of the linker.</p> <p>(.EXE)</p> <p>Parameters required by the linker to generate an .EXE.</p>

Options

Start-up options for the linker when generating an executable. These options depend on the linker used.

Libraries

Pathnames of the libraries used by the linker.

The default list displayed must be completed using a '+' character followed by the pathname of the LIB file to be used.

As a general rule, each DLL that you use, other than those supplied with NS-DK, must have a corresponding LIB file which you should add to the *Libraries* file list.

Stack size (bytes)

Stack size to be used.

The stack is an area in memory that stores the temporary variables used by your application.

The default size proposed is sufficient in most cases. You should increase it if you have a large number of windows or if the processing performed makes numerous calls to embedded functions.

(.DLL)

Parameters required by the linker to generate a DLL.

Options

Linker start-up parameters. These depends on the linker used.

Libraries

Pathnames of the libraries used by the linker.

The default list displayed must be completed using a '+' character followed by the pathname of the LIB file to be used.

As a general rule, each DLL that you use, other than those supplied with NS-DK, must have a

corresponding LIB file which you should add to the *Libraries* file list.

Implib

This utility is only used to generate DLLs.

It generates a LIB file containing the names of the procedures in the corresponding DLL. The LIB file is used when you link an external program that uses the DLL.

Path

Pathname of IMPLIB.EXE.

ImpLibOpts

Implib start-up parameters. These depend on the ImpLib utility used.

Resource Compiler

Configuration of the resource compiler used to associate an icon with an executable that has been generated. It creates object files (with the extension .OBJ) from C source files.

Path

Full pathname of the resource compiler that you are using.

ResOpts

Start-up parameters for the resource compiler. These depend on the compiler used.

OK

Closes the dialog box and confirms any changes made.

Cancel

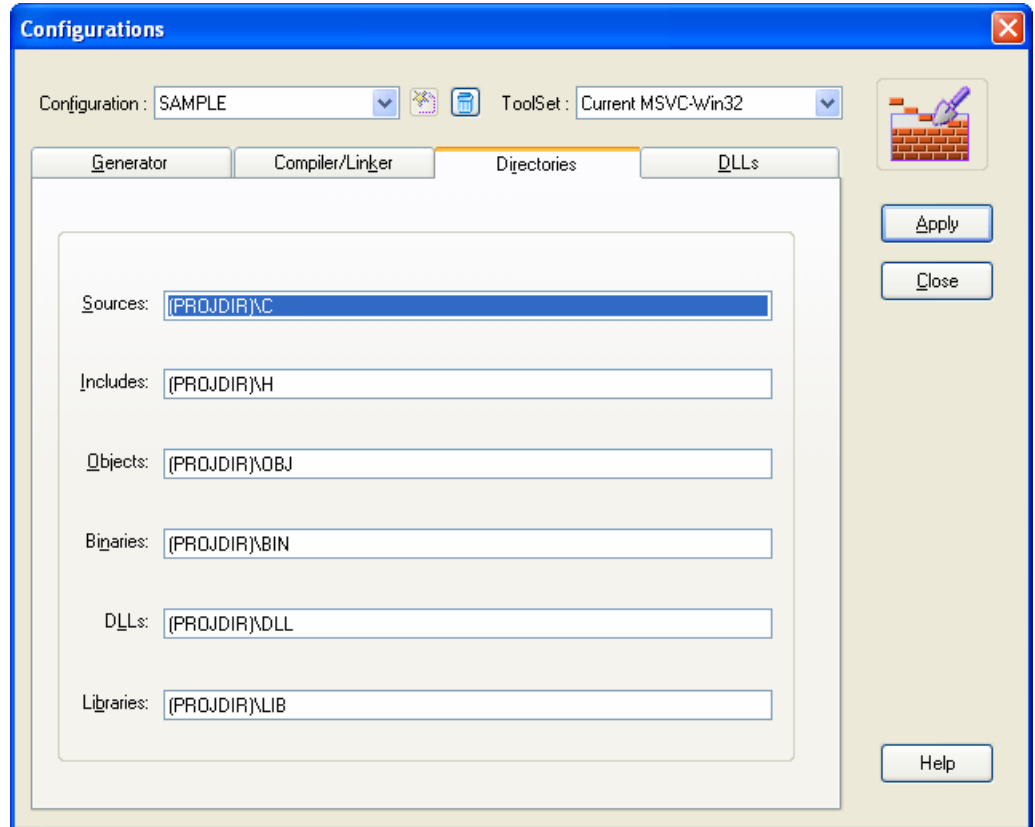
Closes the dialog box and ignores any changes made.

Help

Displays on-line help for this dialog box.

Directories tab

The *Directories* tab allows you to specify the paths of the directories used to store all the generated files.



This tab contains the following fields:

Configuration	<p>Displays the current configuration.</p> <p>Also lets you create or delete a configuration. This field does not let you select a configuration. Selecting the configuration is carried out in the box <i>Build/Run Configuration</i>.</p>
	<div data-bbox="718 1523 766 1579"></div> <div data-bbox="813 1523 1402 1579">The button opens the <i>Create New Configuration</i> box.</div>
	<div data-bbox="718 1601 766 1657"></div> <div data-bbox="813 1601 1402 1657">The button <i>Delete current configuration</i> opens a box in which you can confirm you wish to delete the configuration.</div>
ToolSet	<p>Displays the toolset associated to the current configuration. You can modify it. This field lists the toolsets defined in the nsdkcfg.ini file.</p>

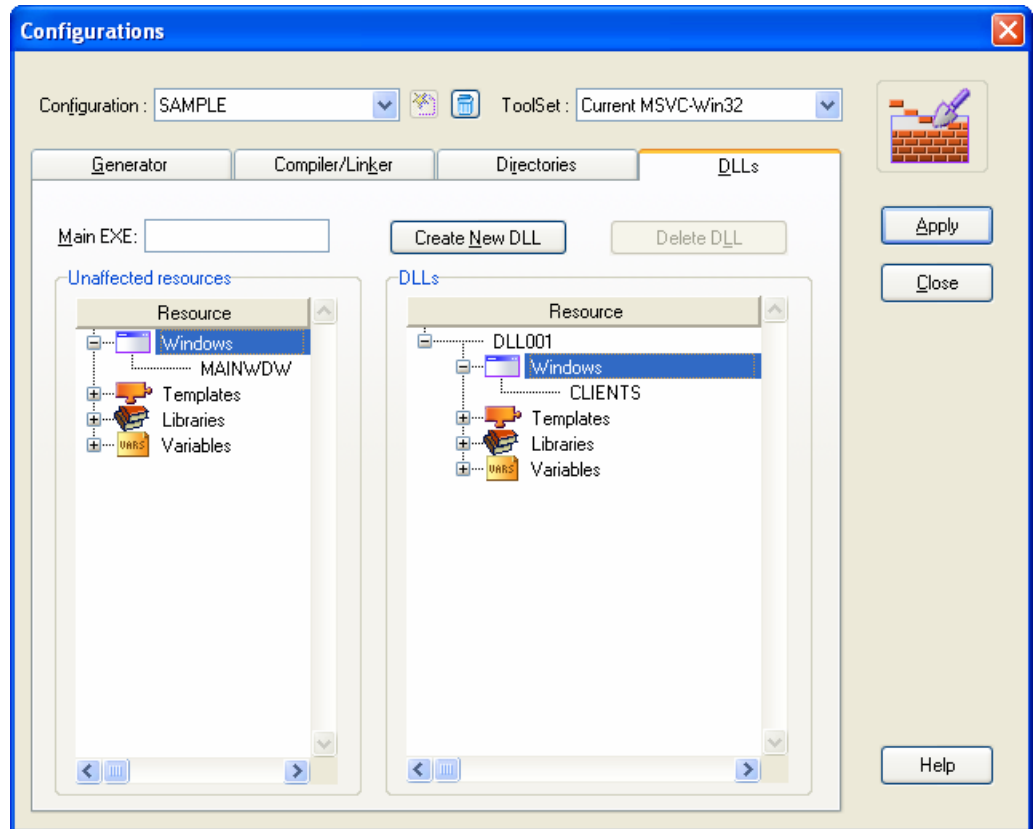
Sources	Directory used to store the source files (.C) generated by NS-Gen.
Includes	Directory used to store the include files (.H and .D) generated by NS-Gen.
Objects	Directory used to store the object files (.OBJ) generated by the compiler. It also stores two files generated by NS-Gen: the definition file (.DEF) and the .NSE file that contains the names of any exported functions.
Binaries	Directory used to store the executable file (.EXE) generated by the linker as well the external resource files (.RES) generated by NS-Gen.
DLLs	Directory used to store the DLLs generated by the linker.
Libraries	Directory used to store the LIB files generated by the ImpLib utility.
OK	Closes the dialog box and confirms any changes made.
Cancel	Closes the dialog box and ignores any changes made.
Help	Displays on-line help for this dialog box.

DLLs tab

This tab lets you to define the following information:

- the name of the DLL or DLLs to be generated and the name of the executable if you wish to generate the application in the form of an executable and several DLLs.
- the resources associated with each DLL.

This tab is only enabled if the *Multiple DLLs* field under the *Generator* tab is checked.



This tab contains the following fields:

Configuration Displays the current configuration.

Also lets you create or delete a configuration. This field does not let you select a configuration. Selecting the configuration is carried out in the box *Build/Run Configuration*.



The button opens the *Create New Configuration* box.



The button *Delete current configuration* opens a box in which you can confirm you wish to delete the configuration.

ToolSet Displays the toolset associated to the current configuration. You can modify it. This field lists the toolsets defined in the nsdkcfg.ini file.

Main EXE	<p>Name of the executable file. This file will be the entry point for the application, it will generate an executable file, the sole task of which is to start the application and to call the DLL containing the application's main window.</p> <p>You should only set this field if you are generating your application as an executable with several DLLs.</p> <p>If you specify the name of an existing DLL file, it will be rejected. This is because NS-Gen generates a source file with the name specified in Main EXE. This source file will contain the application's entry point.</p>
Create New DLL	<p>Opens the <i>Create New DLL</i> window that lets you to enter the name of a DLL to be incorporated into the list of <i>DLLs</i></p>
Delete DLL	<p>Deletes the selected DLL from the list of <i>DLLs</i>. A dialog box appears requesting confirmation of the deletion. Press <i>OK</i> to confirm, and <i>Cancel</i> to stop the deletion.</p>
Unaffected resources	<p>Name of resources not associated with a DLL. In order to associate them with a DLL, drag the selected resource into the list of <i>DLLs</i> with the right mouse button.</p> <p>The resources selected disappear from the <i>Unaffected resources</i> list and are displayed in the list of <i>DLLs</i>.</p>
DLLs	<p>Name of a source file that can be associated with NS-DK resources and will be compiled as a DLL. The DLL will have the same name as the source file.</p> <p>A valid name MUST NOT:</p> <ul style="list-style-type: none">• Begin with a digit• Exceed 8 characters• End with an extension• Be preceded by a pathname.
Apply	<p>Closes the dialog box and confirms any changes made.</p>
Close	<p>Closes the dialog box and ignores any changes made.</p>
Help	<p>Displays on-line help for this dialog box.</p>

NSDKCFG.INI and NSDKLOC.INI files



The NS-DK.INI file has been replaced with two configuration files: NSDKCFG.INI (corresponds to your project's global configuration file) and NSDKLOC.INI (corresponds to a project's local configuration file).

The file NSDKCFG.INI is provided with NS-DK. The purpose of this file is to be shared on a networked NS-DK installation, for example. The NSDKCFG.INI file is usually modified by an NS-DK installation administrator or a project manager.

The file NSDKLOC.INI is not provided with NS-DK. It's a configuration file specific to each user, who creates it and modifies it.

The purpose of the NSDKCFG.INI and NSDKLOC.INI files is to specify a range of tools and parameters that are displayed as defaults in the various generation boxes. These tools are grouped into toolsets, whose names are displayed by the *Toolset* field in the *Create new Configuration* and *Configurations* boxes. The corresponding details for the toolset selected from this box are displayed in the other generation boxes:

- The start-up parameters for NS-Gen are displayed as defaults in the *Configurations* box at the *Generator* tab.
- The compiler and its parameters are displayed as defaults in the *Configurations* box at the *Compiler/Linker* tab.
- The linker, ImpLib utility, resource compiler and their parameters are displayed as defaults in the *Configurations* box at the *Compiler/Linker* tab

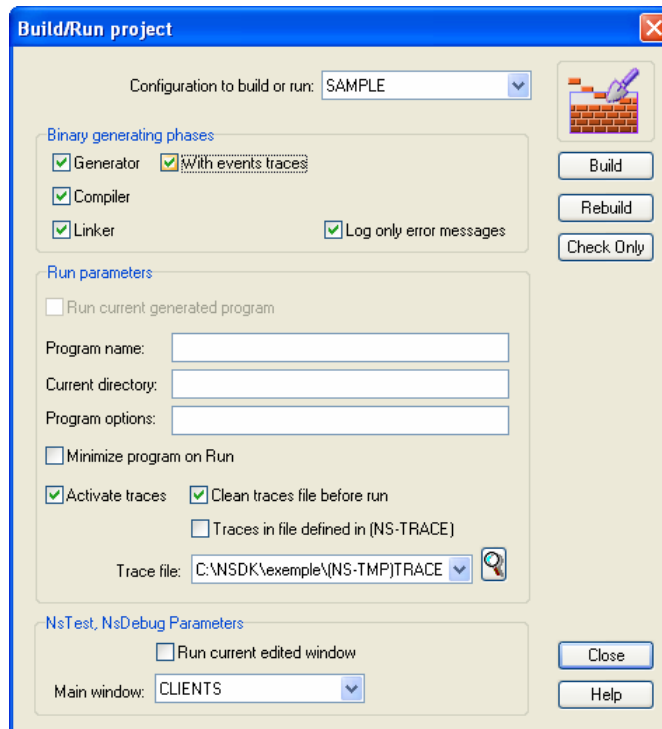
The NSDKCFG.INI file allows you to generate your applications rapidly and easily by selecting only one toolset. The corresponding tools and their start-up parameters will automatically appear in the generation boxes for each new application.



For more informations about these two files, see the ".INI Files" manual.

Generating, compiling and linking

Build/Run project box





The *Build/Run project* box can be obtained by selecting the *Build/Run configuration* option in the Build menu.


This dialog box allows you to start NS-Gen, which uses all the information you have entered in the *Configurations* box.

It is divided into two parts:

- The top part allows you to select the configuration file that you want to use.
- The second part contains the various operations that NS-Gen can perform sequentially.

This tab contains the following fields:

Configuration to build or run	Selects the configuration file to be used from the configurations specified in the <i>Configurations</i> box.
Build	<p>Triggers the various actions selected (<i>Generator</i>, <i>Compiler</i>, etc.) for the files that have been modified:</p> <ul style="list-style-type: none">• A C source file is only regenerated if the corresponding resource has changed since the last generation or if the C file does not exist.• A C source file is only recompiled if the file has changed since the last generation or if the corresponding .OBJ file does not exist.• The link is only performed if one of the .OBJ files has changed since the last generation or if the .DLL library or .EXE executable does not exist. <p> In some cases, you will need to force a regeneration of the application (by pressing the <i>Rebuild</i> button). This is usually necessary when one of the build parameters has been changed (<i>Configurations</i> box).</p> <p> The directories specified in the Configurations box at the Directories tab allow you to store the generated files. If these directories don't exist, a confirmation message appears asking you to create them.</p>
Rebuild	Triggers the various actions selected (<i>Generator</i> , <i>Compiler</i> , etc.) for all files, no matter when they were last updated. Everything is regenerated (C files) if the <i>Generator</i> option is selected; everything is recompiled (.OBJ files) if the <i>Compiler</i> option is selected; and everything is relinked (.EXE or .DLL files) if the <i>Linker</i> option is selected.
Generator	This option tells NS-Gen to generate the source files.
Compiler	<p>This option tells NS-Gen to compile the source files after generating them.</p> <p>If the <i>Generator</i> option was selected, the compiler will only run if no errors were detected while generating the source files.</p>
Linker	<p>This option tells NS-Gen to link the object files produced by the compiler.</p> <p>If the <i>Compiler</i> option was selected, the linker will only run if no errors were detected while compiling.</p>

Check only	<p>This option tells NS-Gen to only run a syntax check on the resources associated with each source file.</p> <p>No source files will be generated if this option is checked.</p> <p> By using this option you can display all the syntax errors caused by the resources associated with the selected source files; the syntax checker in the <i>Events</i> box only indicates the first error detected in the script displayed.</p>
Log only error messages	List of errors detected by NS-Gen during each stage of the generation. Warning messages will not be displayed if this option is ticked.
OK	Closes the dialog box. If a generation has been started, it will continue in the background. You can check the progress of the operations with the <i>Log</i> window.
Help	Displays on-line help for this dialog box.

Generation Process

Generating source files

Pressing the *Build* or *Rebuild* button triggers the call to NS-Gen.


NS-Gen runs as a background task named 'Generating project xxxx', where xxxx stands for the name of the current project. You can display details of the operations carried out by pressing [Ctrl]+[Esc] and selecting this task from the Task list. The details are displayed in an DOS window session which is automatically closed when the last operation is completed.

Compiling and Linking

The compiler and linker are started by NS-Gen in the same DOS window session as the one described above. By displaying this session, you can check the parameters passed to these utilities. Any errors returned are displayed in the Errors list in the *Log* window.

Stopping the Generation Process

The generation process can be stopped during any phase (source generation,

compiling, linking) by pressing the *Stop*  button in the toolbar.

Generator Block

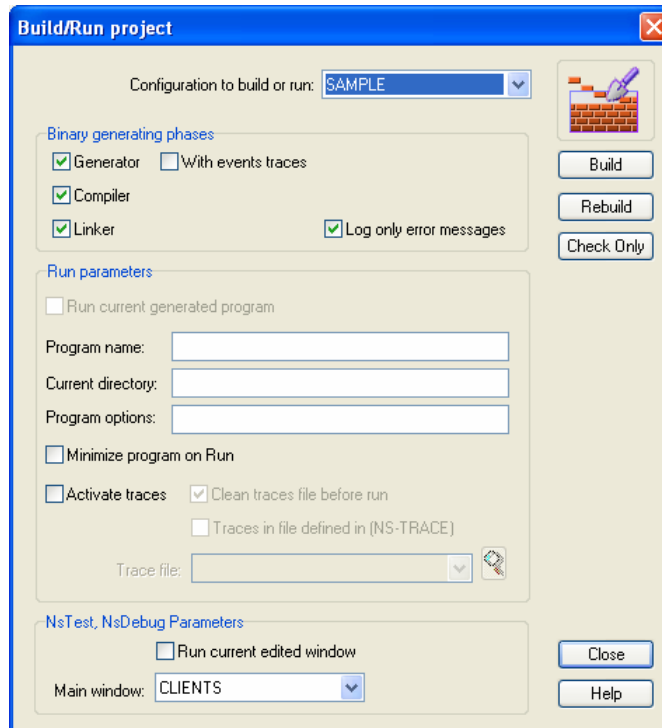
The generation program may sometimes act as though it is blocked: no more messages are displayed in the *Log* window and no more information is sent back to indicate that the generation is complete.

The last error message in the *Log* window may help put you on the right track.

A block problem occurs when the pathname of a .LIB file has not been specified in the *Configurations* box at the *Compiler/Linker* tab.

The *Log* window contains the message: "Cannot find library"

Running the Generated Executable



The *Build/Run project* box, called by selecting the *Build/Run Configuration* option in the *Build* menu, lets you to run the executable program generated by the *Build/Build* command or any other executable program that you wish to run without exiting NS-Design.

This tab contains the following fields:


Program name Pathname of the executable program that you want to run.



This program can be a Windows executable, a DOS executable, a .BAT batch file or a .PIF file.

Program options Start-up parameters for the program specified in *Program name*.

These parameters, together with the syntax required, depend on the program run. They are identical to the ones you specify after the executable name when you call it from the command line in the Windows *File/Run* option or the DOS command line.

Minimize program on Run	Minimizes NS-Design while the program specified in <i>Program name</i> runs. If this option is not checked, NS-Design stays in the background while the program runs.
Build	<p>Closes the dialog box and launches the program. If the program cannot be started up, an error message is displayed indicating the cause (e.g. program not found).</p> <p> If your NS-DK application's main window was designed without checking the <i>Quit on close</i> option in its property pane, you will need to select the <i>Stop</i> option from the <i>Build</i> menu in NS-Design to actually terminate the application and be able to start another one via <i>Start EXE</i>.</p>
Close	Closes the dialog box, ignores any changes made and does not start a program.
Help	Displays on-line help for this dialog box.

Chapter 3

Syntax for NS-Gen



This chapter presents the NS-Gen syntax.



NS-Gen can be launched from a DOS session.

***You will find in
this chapter***

- Presentation of NS-Gen syntax.
- How to obtain NS-Gen syntax.

Contents

Syntax.....	3-3
{options}	3-3
[PrimaryScreen]	3-8
General points	3-9
Displaying the Syntax for NSGEN	3-9

Syntax

The following syntax should be used:

```
NSGEN {options} [PrimaryScreen]
```

where {options} stands for the generator options and {PrimaryScreen} specifies the name of the application's main window.

{options}

stands for the generator options.

In general, several options should be specified.

Their syntax and description are as follows (in alphabetical order):

/CHECKONLY

Tells NS-Gen to only run a syntax check on the resources associated with each source file.

No source files will be generated, even if no syntax errors are detected.

Equivalent to the Check only option in the Build/Run project box

/CL

Tells NS-Gen to compile the source files after generating them.

If the /NOGEN option is not specified, the compiler will only run if no syntax errors were detected while generating the source files.

If the /FORCE option is not specified, a C source file will only be recompiled if the file has changed since the last generation or if the corresponding .OBJ object file does not exist.

Equivalent to the Compiler group in the Configurations box at the Compiler/Linker tab

/CLOPT

Tells NS-Gen the C compiler options.

/CUT[:<LineWidth>]

Indicates that each line in the source file must be split into as many lines as required so as not to exceed 80 characters.

<LineWidth> is used to specify a value other than the default.

Set this number according to the text editor or printer that you are using and depending on whether you want to display or print the generated source files.

Default: 80
Minimum: 40
Maximum: 32767

Examples

/CUT:120 allows up to 120 characters per line
/CUT allows up to 80 characters per line

/DEST:<DestinationName>

If /SETUP is not used, only one executable file is generated. It will have the same name as the window "PrimaryScreen".


/DEST:<Destination Name> allows you to specify another executable filename.

/DLL[:<ExeName>]

Tells NS-Gen to generate one or more DLLs.

- ◆ If you are generating the project as a single DLL, <ExeName> must be omitted.
- ◆ If you are splitting your application into several DLLs, with a main executable that is only used to start the application, <ExeName> should specify the name of this executable.

See Chapter 6 for more details on the additional files generated with DLLs.

 <ExeName> must not match any of the resource filenames.

Equivalent to the Generate DLLs option in the Configurations box at the Generator tab

/FORCE

Triggers the various actions selected (Generator, Compiler, etc.) for all files, no matter when they were last updated. Everything is regenerated (C files) if the /NOGEN parameter is not specified; everything is recompiled (.OBJ files) if the /CL parameter is specified; and everything is relinked (.EXE or .DLL files) if the /LINK parameter is specified.

Equivalent to the Rebuild option in the Build/Run project box

/HELP

Displays the syntax for NS-Gen.

Any other options specified will be ignored by NS-Gen.

/HELPS

Ensures that your application's on-line help file is automatically opened when the end-user presses F1.



For details on how to create a customized on-line help file, see the NSHELP chapter in the Libraries Programming Manual.

Equivalent to the *Help file* option in the *Setup box* at the *Miscellaneous* tab

/LINK

Tells NS-Gen to link the object files produced by the compiler.

If the /CL option is specified, the link will only occur if no errors were detected while compiling.

If the /FORCE option is not specified, the link will only occur if one of the .OBJ files has changed since the last generation or if the .DLL library or .EXE executable does not exist.

Equivalent to the Linker group in the Configurations box at the Compiler/Linker tab

/MAXEXPR:<MaxExprLevel>

Tells NS-Gen to split up expressions that are too complex in the generated C code.

<MaxExprLevel> specifies the maximum expression level allowed. Any expression that exceeds this level will be split into expressions that fall within the acceptable level.

/NAMES

Tells NS-Gen that you are using the CONTROLNAME\$ and WINDOWNAME\$ functions in the NSMISC library to retrieve control and window names.

As a result, NS-Gen generates internal tables for each window so that your application can supply these names.



If you do not check this option but call the CONTROLNAME\$ and WINDOWNAME\$ functions in the NSMISC library, incorrect results will be returned when the application runs.

Equivalent to the Symbols info option in the Configurations box at the Generator tab

/NOGEN

Tells NS-Gen not to generate source files.

Equivalent to unchecking the Generator option in the Build/Run project box

/PRJ:<ProjectName>

Specifies the name of the project being generated.

<ProjectName> specifies the project's full pathname. The project name must be supplied with its ".PRJ" extension.



If /PRJ:<ProjectName> is not specified, NS-Gen considers that the default project is being generated. The NS-INI environment parameter specifies the name of the default project (this parameter is defined in the AUTOEXEC.BAT file under Windows).

Exemple

`/PRJ:C:\NSDK\ESSAI.XNP`

/RESEXTERN[:M]

Tells NS-Gen to generate an external resource file named *ExeName.RES*.

If you are generating DLLs and have specified a Main EXE file, the name of the external resource file will be *MainExe.RES*. It will contain all the resources for the generated DLLs.

The M option tells NS-Gen to generate an external resource file for each DLL. Each of these files will also contain all the resources shared by the DLLs. Any application that uses this type of DLL must load the .RES file associated with the DLL when the application is initialized. If several DLLs of this kind need to be used, the corresponding .RES files should first be merged into a single .RES file. For more details, see the section on using DLLs in Chapter 6 of this manual.

/SETUP[:<ExeName>]

Specified with or without an argument, this option tells NS-Gen to use the information in the configuration file *ProjectName.N_C*.

<ExeName> specifies which of the .DLL and .EXE files defined for the project's configuration files should be generated. If no

filename is specified, all the executable files defined will be generated.

You can specify several /SETUP options to generate several executables. However, since you can only indicate one .EXE per configuration file, but several .DLLs, there is no point using this /SETUP option for a .EXE.

Using /SETUP without a filename is equivalent to using /SETUPF:*ProjectName.N_C*.



The name of the executable file should be specified without its extension.

Exemple

```
/PRJ:C:\NSDK\ESSAI.XNP /SETUP:TEST1  
/SETUP:TEST4 /VERBOSE
```

/SETUPF:<ConfigurationFile>

Tells NS-Gen to use the configuration file *ConfigurationFile.N_C*. This name must match one of the project configuration filenames listed in the *Project Generation* box.

Any additional options used on the command line (such as /TAB and /CUT) will override the corresponding option in the configuration file.

Equivalent to the Configuration to build or run field in the Build/Run project box

/SQLSTATIC

Tells NS-Gen to generate source files containing static SQL.

This option should be used for applications that use NS-SQL so as to produce a C source file with the .SQC extension that contains static links with a specific SQL engine.

Static SQL is a sub-language that can describe SQL queries independently of the operating system or programming language used. Once these types of source files have been generated by NS-Gen, you need to use a pre-compiler that converts the Static SQL according to the programming language used. You will then need to bind the code generated by the pre-compiler with the SQL engine itself. This will improve performance for SQL access.

The pre-compiler for DB2 is SQLPREP, while the bind is performed with SQLBIND.

`/TAB:<TabulationSize>`

Indicates the number of space characters generated for each indentation in the source file.

Default: 4

Minimum: 2

Maximum: 32

Equivalent to the Tabulation option in the Setup box

Example

`/TAB : 6`

`/TOOLKIND:<MSC|MSVC32|IBMCPP>`

Indicates the type of tool used so that NS-Gen can determine the various system characteristics it requires, such as the format of the .DEF definition file, any additional compiler options that must always be used (other than the ones specified in the *Setup compiler* box), the format required by the linker, etc.



MSVC32 : Microsoft Visual C++ (32 bits),

To determine the appropriate TOOLKIND, simply use the one associated with your compiler, as shown above.

`/VERBOSE`

Tells NS-Gen to display any information that can be used to track the various generation phases.

If this option is not used, NS-Gen will only display any errors detected. If no errors are detected; there will be no messages indicating which phase is currently in progress.

[PrimaryScreen]

Specifies the name of the application's main window.

This is the window that will be displayed when the generated program is started up.

Notes

The specified window name must not end with the '.SCR' extension.

The name of the main window must be specified if the /SETUP option is not used. The name of the generated executable will be the name of the main window followed by the ".EXE" extension.

General points

If the /SETUP option is not used, you cannot define the directories in which the various files will be created. They will all be created in the directory that is current when NS-Gen is started up. However, if you use the /SETUP option, the various files will be stored in their respective directories, as specified in the *Setup directories* box.

NSGEN can be placed before the compilation section in a MAKE file. In this case, the target will be a source file and the dependencies will be the various SCR, TPL, ..., and BMP files containing the NS-DK resources associated with the source file. If a resource file is modified after creating the source file, NSGEN must be started with the /SETUP option followed by the name of the source file.

See the tips on generating in Chapter 6 of this manual.

Displaying the Syntax for NSGEN

You can display the full syntax for NSGEN by typing the following on the DOS command line:

```
NSGEN /HELP
```

Dans ce cas, apparaît à l'écran :

```
Nat-Systems(R). NS-Gen version 3.00.02
```

```
Copyright Nat-Systems 1990-2005. All rights reserved.
```

```
usage : NSGEN {options} [PrimaryScreen]
```

where {options} are:

```
/CHECKONLY
```

```
/CL
```

```
/CUT[:<LineWidth>]
```

```
/DEST:<DestinationName>

/DLL[:<ExeName>]

/FARDATA

/FORCE

/HELP

/HELPS

/LINK

/MAXEXPR:<MaxExprLevel>

/NAMES

/NOGEN

/NOQUOTE

/OPTIMIZE

/PRJ:<ProjectName>

/RESEXTERN[:M]

/SETUP[:<ExeName>]

/SETUPF:<ConfigurationFile>

/SQLSTATIC

/OPENCURSOR:<init number>

/NOSQLENV

/TAB:<TabulationSize>

/TOOLKIND:<MSC | MSVC32 | IBMCPP>

/VERBOSE

/WIN16
```

Chapter 4

Syntax for NS-ResLib



***You will find in
this chapter***

- Presentation of NS-ResLib syntax.
- How to obtain NS-ResLib syntax.

Contents

Introduction.....	4-3
Syntax.....	4-4
Displaying the Syntax for NSRESLIB	4-6

Introduction

The external resource librarian, NSResLib, allows you to:

- List the contents of an external resource file.
- Extract one or more targets from an external resource file.
- Merge several external resource files.

Syntax

NSResLib can be launched from a DOS session. Its syntax is as follows:

```
NSRESLIB <Option> <InputFiles> [<OutputFile>]
```

where <Option> stands for one of three possible options:

/L <InputFile>

Displays the contents of the resource file <InputFile>, i.e. lists all the targets defined in the file, followed by all the resources with no data, and finally, all the resources with data.

Each line in the last list contains the resource name, the data class and the target name.

Example

```
NSRESLIB /L TARGET.RES
```

displays :

```
NatSys (R) External Resources Librarian. Version 1.00.  
Copyright (C) Nat Systèmes 1992-1994. All rights  
reserved.
```

```
Targets defined in this file : Generic Target  
                               Env. Anglais
```

```
Resources with no data : ABOUT,BMP  
                        T_LSTGEN,TXT  
                        T_LSTPRO,TXT  
                        T_LSTSOC,TXT
```

```
Resources with data : D_ABOUT, SCR, Env. Anglais  
                      D_ABOUT, SCR, Generic Target <D>  
                      C_MSG, CST, Env. Anglais  
                      C_MSG, CST, Generic Target <D>  
                      W_MAIN, SCR, Env. Anglais  
                      W_MAIN, SCR, Generic Target <D>
```

/M <InputFile1> ... <InputFileN> <OutputFile>

Merges the resource files <InputFile1> ... <InputFileN> into the resource file <OutputFile>.

The merge cannot occur if the external resource files being merged have at least one resource in common that contains data and is associated with the same target. In this case, NsResLib

displays a list of duplicate targets together with the names of the files in which they appear.

You can merge up to 16 files.

Example

```
NSRESLIB /M CIBLE1.RES CIBLE2.RES DEST.RES
```

```
displays :  
NatSys (R) External Resources Librarian. Version  
1.00.  
Copyright (C) COGNICASE 1992-1994. All rights  
reserved.
```

```
Merging cible1.res cible2.res into dest.res  
Merge done.
```

/E"<Target1>" ... /E"<TargetN>" <InputFile> <OutputFile>

Extracts from <InputFile> all the targets passed as parameters together with all the resources and associated data, then builds <OutputFile>.

During the extraction, the first target in the output file becomes the Default target.

You can extract up to 16 targets from a file.

Each target name must be enclosed in double-quotation marks ("). There should be no spaces between /E and the target name.

Example

```
NSRESLIB /E"Env Français" /E"Env Anglais"  
TARGET.RES TRGT_ENV.RES
```

```
displays :  
NatSys (R) External Resources Librarian. Version  
1.00.  
Copyright (C) COGNICASE 1992-1994. All rights  
reserved.
```

```
File was built : trgt_env.res
```

Note

Options can either be preceded by a slash (/) or a hyphen (-) and can be typed in lowercase or uppercase.

For example: '/L', '/l', '-L' and '-l' are equivalent.

Displaying the Syntax for NSRESLIB

To display the help panel, use the /H option. Hence, the command:

```
NSRESLIB /H
```

displays :

NatSys (R) External Resources Librarian. Version 1.00.
Copyright (C) Nat Systèmes 1992-1994. All rights reserved.

NsResLib <Option> <InputFiles> <OutputFile>

Options :

```

    /L,/l,-L,-l : List contents of file <Inputfile>
    /M,/m,-M,-m : merge up to 16 input files into one
                  output file
    M+          : ignore duplicate resources.
    /E"<Target>" ,
    -E"<Target>" ,
    /e"<Target>" ,
    -e"<Target>" : Extract <Target> out of an input
                  file into an output file
                  There can be up to 16 /E"<Target>"
                  options.
    /H,/h,-H,-h : Displays this help
    <InputFiles> : One or more (/M) valid resource
                  files
    <OutputFile> : a destination file name. Only in /M
                  or /E.
```

Chapter 5

Tips for generation



You will find in this chapter

- Practical advice on generating an application developed under NS-Design.
- How to carry out an incremental build.
- Tips for generating an executable and DLLs.
- How to handle errors.

Contents

Incremental Building	5-4
How an Application is Split Up and the Automatic Make Facility	5-4
Time Taken to Build an Application	5-4
File names and sizes	5-5
When to Regenerate	5-5
Make Facility	5-5
Generating an executable.....	5-7
Generating Source files	5-7
<i>Build/Run project</i> Dialog Box	5-7
Using NS-Gen Directly	5-7
Compiling and Linking	5-8
<i>Build/Run project</i> Dialog box	5-8
Alignment	5-8
Using NS-Gen Directly	5-9
Generated Files	5-9
Source Generation	5-9
Compiling	5-10
Linking	5-10
Generating DLLs	5-11
Reminder	5-11
Using NS-Gen Directly	5-11
Generated Files	5-12
Source Generation	5-12
Compiling	5-12
Linking	5-12
Using Generated DLLs	5-13
Applications Written in C or Pascal	5-14
Applications Written in NCL	5-14
Using DLLs that contain functions	5-14
Using DLLs that contain windows	5-14
Loading DLLs dynamically	5-15
Loading DLLs statically	5-15
Data shared by DLLs	5-16
Using DLLs with an external resource file	5-16
Using DLLs Generated in C	5-17
Example 1	5-17
Example 2	5-18
Example 3	5-18

Errors	5-20
Errors During the Source Generation Phase	5-20
Errors During the Compilation Phase	5-21
Errors and Warnings	5-21
Errors During the Link Phase	5-22
Application Run-Time Errors	5-22

Incremental Building

How an Application is Split Up and the Automatic Make Facility

NS-Gen automatically splits up each application so that it can build it incrementally. The term 'incremental building' refers to the fact that each NS-DK resource (window, template, library, etc.) is associated with a .C file and a .H file.

Based on this breakdown, NS-Gen generates an application in the same way as a Make utility:

- A .C file is only regenerated if the corresponding resource has been modified since the last generation or if the corresponding .C file or .H file does not exist.
- A .C file is only recompiled if it has been modified since it was last compiled or if the corresponding .OBJ file does not exist.
- The .OBJ files are only linked if one of them has been modified since the last link or if the application's .EXE or .DLL file does not exist.

Notes

You can always force a regeneration, recompilation or new link by using the Rebuild button in the Build/Run project box (Build/Build option), or the /FORCE option with NS-Gen.

Some resources have no corresponding .C file, in particular .NCL libraries that contain constant declarations, .CST constant files and .SEG segments.

Time Taken to Build an Application

The above feature is comparable with a Make facility. It speeds up the process of building an application.

Supposing that a line of code has been modified in resource R, then the total time taken to build the application — $T()$ — will be:

$$T() = T(\text{generate resource } R) + T(\text{compile } R.C) + T(\text{link})$$

Note

The situation described above is a simplified "best case scenario" where the only resource that needs to be regenerated is resource R. Nevertheless, although this is a simplified case, it is not rare.

File names and sizes

The generated files associated with a resource have the same name as the resource but end with the extensions .C and .H. For this reason, you should be careful when naming your resources: their name should not match the name of an external C file, such as a file containing procedures and calculations written independently of NS-DK and compiled in DLL form.

If you do not take this precaution and your .C files are generated in the same directory as your native C sources, some of your external files may be overwritten by the files generated from your resources.

The size of the generated files is minimized by the fact that the application is broken down.

When to Regenerate

If one of the setup parameters for the build process has been changed, you will need to force a regeneration of the whole application. Changes to the following options (which can be modified via the *Project generation* box) require you to regenerate:

- Targets: *Target* field
- Directories: *Directories* tab in the *Configurations* box
- Executable: *Generator* tab in the *Configurations* box
- DLLs: *DLLs* tab in the *Configurations* box
- Generator: *Generator* tab in the *Configurations* box
- Compiler: *Compiler/Linker* tab in the *Configurations* box
- Linker: *Compiler/Linker* tab in the *Configurations* box

Make Facility

Compilers and linkers usually come with a utility that automates the process of generating an executable (called MAKE or NMAKE by Microsoft and IBM).

The basic principle behind this tool involves applying an action to a file when certain other files have been updated more recently. For example, you can tell the system to compile a source file only if the include files that it uses have been updated more recently than itself. With a Make facility, files are kept up to date automatically.

Incremental building allows applications to be generated using the same principle, since only the files that have changed are regenerated, recompiled or relinked. Therefore, it is much easier to use this feature of NS-Design and NS-Gen, which is available via the *Build/Build* command.

Generating an executable

Each step involved in generating an executable can be requested from the *Build/Run Configuration* dialog box in NS-Design (*Build/Build* option) or by using NS-Gen independently of NS-Design.

Generating Source files

***Build/Run project* Dialog Box**

To generate source files, you can press either the *Build* or *Rebuild* button in the *Build/Run project* dialog box.

If, initially, you only want to run a syntax check on the NCL code, select the *Check Only* option. As a result, NS-Gen will not create a source file and will display any errors detected in the Errors list.

Whether you select the *Build* or *Rebuild* button depends on the changes made to your resources:

- ◆ If these changes only affect the application's resources, use the *Build* button. This will only regenerate the files associated with these resources, and your EXE file or DLLs will be produced very rapidly.
- ◆ If these changes affect the parameters for one or more generation tools (generator, compiler, targets, etc.), use the *Rebuild* button, which will force a generation of all the files and trigger a global compilation. This will ensure that all your files have been generated with the same parameters. You can also use this button once your application is complete so that your final checks are performed on a fully regenerated binary file.

Using NS-Gen Directly

Using NS-Gen directly is only appropriate for an application that is generated from several projects. In this case, a batch generation program can launch NS-Gen for each of the projects involved.

Compiling and Linking

Build/Run project Dialog box

The compilation and link phases are triggered automatically by NS-Gen if the *Compiler* and *Linker* options have been checked in the *Build/Run project* dialog boxes, and after generating the source files, if the *Generator* option has been selected.

As a general rule, compiling and linking are two interdependent stages. For this reason, both these options are always checked by default.

If your application contains a large number of resources, the compilation phase may be lengthy in the following cases:

- If all the C files need to be recompiled.
- If you are generating the application for the first time.
- If you have used the *Rebuild* option.

Obviously, if you have the slightest doubt as to which source files are affected by any changes made, you should regenerate the whole application (*Rebuild* button); the extra time taken to generate the application will be more than adequately compensated for by the time saved in testing an invalid executable.

You cannot start the compiler and linker until you have generated; you will always need to regenerate at least one source file after making changes.

You can also generate with a MEDIUM, SMALL or COMPACT model. However, note that only the MEDIUM or SMALL models enable you to generate a multi-instance executable, i.e. a program that can be launched several times, where each instance is independent of the others.

Alignment

Each compiler configuration that appears in the NSDKCFG.INI file includes the byte-alignment option for the fields in a structure, also referred to as "packing = 1". For Microsoft compilers, this option is `/Zp` and for IBM compilers it is `/Sp1`.

This option can reduce the performance of the code generated by the compiler. However, it is sometimes necessary since SEGMENT variables are themselves allocated with byte-alignment. Similarly, if you are testing an application that uses DLLs generated by NS-DK, the same alignment should be used by NS-Test and the DLLs. Therefore, you should always compile these DLLs with an alignment of 1.

On the other hand, if the above conditions do not apply to the application that you are developing, you can switch off this alignment option to improve performance. In other words, if your application does not use any DLLs generated by NS-DK, if you do not execute NS-Test, or if the application does not define any segments, you can compile without byte-alignment.

Using NS-Gen Directly

The case for using NS-Gen directly, described in 6.2.1., also applies for compiling and linking.

For prototypes, the /CL and /LINK options should be used together so that the compiler and linker are run consecutively and an executable is obtained immediately.

The generated executable will have the same name as the main window (*PrimaryWindow*), followed by the .EXE extension. If this is unsuitable, you can rename it, provided that you keep the .EXE extension.

For a MAKE file, only the /CL option should be used to prevent the link from starting before all the object files have been generated. Using /CL means that the default parameters will be passed to the compiler. If these parameters are unsuitable, you should compile the source files outside NS-Gen. The link will be performed in the standard way, at the end of the MAKE process.

Generated Files

Source Generation

For each resource:

- A source file with the same name and the .C extension.
- An include file with the same name and the .H extension.
- An include file with the same name and the .D extension, if the /FARDATA option has been used.

For the executable file or DLL:

- A definition file with the same name and the .DEF extension. This file is built according to the target and compiler used and is based on a .NSE file that contains the names of the exported functions (only for DLLs).
 - A .RES external resource file, if the /RESEXTERN option has been used.
-

Compiling

For each generated source file:

- An object file with the same name and the .OBJ extension.

Linking

- An executable file with the same name and the .EXE extension.
- Or a dynamic library with the same name and the .DLL extension.
- A file with the same name and the .LRF extension (Linker Response File), stored in the NS-DK temporary directory. This file contains the options required by the linker.

The definition file is required by the linker to generate the executable.



For a Windows 32-bit target, there is no .DEF file associated with the executable.

Generating DLLs

Reminder

Here is a brief reminder about DLLs. Please refer to the OS/2 or Windows technical documentation for further details.

Dynamic Linking enables programs to use and share functions or memory resources outside their own .EXE file. These are located in DLLs (Dynamic Link Libraries).

Just like an executable file, a DLL contains code and data. However, a DLL is not executed directly but is called by executable files or other DLLs.

For example, NS-Design comprises a number of DLLs that NS-Test and NS-Gen can use simultaneously. Some of these DLLs, such as NSnnLIB.DLL, are also used by the programs generated by NS-Gen.

When you generate a program that calls functions in DLLs, the linker uses the information in the .LIB file associated with each DLL to build tables in the .EXE file so that it can identify these functions and the DLLs to which they belong. The code in the .EXE file that calls these functions contains only a dummy address.

There are two main reasons for generating a program in DLL form:

- Parts of the program can be reused and shared by several programs.
- Less memory is used by each program at a given time.

Similarly, by isolating modules of code inside DLLs, they can be modified without having to regenerate the programs that use them, unless, of course, the parameters used to call the DLL's functions are modified.

Using NS-Gen Directly

Using the /DLL option tells NS-Gen to generate DLLs.

If the /SETUP option is not used, a single DLL file will be generated.

On the other hand, the /SETUP option allows you to generate DLLs from the DLL files defined in the *DLLs* tab of *Configurations* dialog box, based on associations made with the NS-DK resources which you can fully control. The name of the executable used to start up the application is the one specified in the /DLL option.

Generated Files

Source Generation

For each resource:

- A source file with the same name and the .C extension.
- An include file with the same name and the .H extension.
- An include file with the same name and the .D extension, if the /FARDATA option has been used.

For each DLL file:

- A definition file with the same name and the .DEF extension. This file is built according to the target and compiler used and based on a .NSE file that contains the names of the exported functions.
- A .RES external resource file, if the /RESEXTERN option has been used.

For the executable file (if applicable):

- A source file named *MainExe* (*Setup DLLs* box) with the .C extension.
- A definition file with the same name and the .DEF extension.
- A .RES external resource file, if the /RESEXTERN option has been used.



You will need to merge the various external resource files obtained into a single file using NSResLib (See Chapter 5 for details on its syntax) since an executable can only call one resource file.

Compiling

For each generated source file:

- An object file with the same name and the .OBJ extension.

Linking

- An executable file whose name was specified in the *Main EXE* field of the *DLLs* tab of the *Configurations* box or using the /DLL option.

For each generated DLL:

- A DLL file with the same name and the .DLL extension.
- An import file with the same name and the .LIB extension.

- A file with the same name and the .LRF extension (Linker Response File), stored in the NS DK temporary directory. This file contains the options required to link the DLL.



The .DEF file for a Windows 32-bit target:

```
LIBRARY DLL_NAME
EXPORTS
    List of "exportable" functions in the DLL
```

The .LIB filename should be specified to the linker when you generate a program that uses the DLL.

Using Generated DLLs

The way that your generated DLLs are used depends largely on their contents and the applications that call them.

Whether the application that uses your DLLs is written in C, Pascal or NCL, you need to supply the DLLs 'NSnnxxx.DLL' together with your own DLLs if your functions call the functions in the corresponding NSxxx.NCL libraries. For example, if your DLL calls the T_* functions in the NSMISC.NCL library, you need to supply the DLL 'NSnnMISC.DLL'.

You should also supply the following DLLs with your application:

For all targets: NSnnWPS.DLL, NSnnINIM.DLL and NSnnKMP.DLL; plus the following DLLs, depending on the target:



NSnnLIB.DLL, NSnnMEM.DLL, NSnnRT.DLL, NSnnTBLS.DLL and NSnnPRN.DLL,



NSnnLIB.DLL, NSnnRT.DLL, NSnnEM.DLL, NSnnTBLS.DLL and NSnnPRN.DLL.

Applications Written in C or Pascal

To be able to access a DLL written in NCL from code written in C or Pascal, you need to create a description file for your functions which gives the equivalent C or Pascal type for each NCL type. In C, this definition usually appears in an include file.

Applications Written in NCL

Using DLLs that contain functions

To be able to access your DLL from NCL code, you need to create an NCL file which includes a description of your functions and indicates the DLL to be used (EXTERNAL 'DLLname.FunctionName').

The function is defined in the same way that it is declared. For example, if you have coded the following function in a DLL named SAMPLE.DLL:

```
FUNCTION TEST% (A%,B%)  
    RETURN (A% + B%)  
ENDFUNCTION
```

The name of the function (iTEST) was retrieved from the EXPORTS list in the file SAMPLE.DEF.

So, the EXTERNAL declaration of the function in the file, which we will call SAMPLE.NCL, is as follows:

```
FUNCTION TEST% (A%,B%) EXTERNAL 'SAMPLE.ITEST'
```

Notes

The DLL and function names specified with EXTERNAL must be in uppercase.

See Chapter 3 of the NCL Programming Manual for more details on the EXTERNAL declaration and the rules for converting between NCL and C/Pascal.

Include SAMPLE.NCL in the library resources for the project that uses the functions in your DLL in the same way that you include the NSxxx.NCL libraries so that you can use their functions. Naturally, your function will also run when you test your application with NS-Test.

Using DLLs that contain windows

The STRCALL* and STROPEN* instructions allow you to open windows belonging to DLLs that are external to the current project and generated by NS-DK.

There are two techniques for loading DLLs: dynamic loading and static loading. Whether you choose one or the other depends on the strategy you have adopted for using DLLs:

- ◆ Dynamic loading is preferable if your application uses dozens or hundreds of DLLs. They will only be loaded when their resources are required.
- ◆ Static loading is preferable if you only use a few DLLs. They will be loaded at the same time as the executable. One of the advantages of this is that, if the load succeeds, all the DLLs will remain in memory and all their resources will be available to the application.

Loading DLLs dynamically

The NCL instruction, `LOADDLL`, loads a DLL dynamically. After this, the windows in the DLL can be called by using the `STROPEN*` and `STRCALL*` instructions. The DLL must be unloaded using the `UNLOADDLL` instruction once it is no longer required.

Example

```
LOADDLL "DLL_name",DLL_handle%  
  
; Call a window in the DLL "DLL_name"  
STRCALL "window_name"  
  
UNLOADDLL DLL_handle%
```

Loading DLLs statically

Using this method requires you to create an NCL library in which you declare two instructions that are generated automatically for each DLL: `REGISTER_XXX` and `DEREGISTER_XXX` (where XXX stands for the DLL name). Typically, this NCL file contains:

```
INSTRUCTION REGISTER_XXX EXTERNAL 'XXX.REGISTER_XXX'  
INSTRUCTION DEREGISTER_XXX EXTERNAL  
'XXX.DEREGISTER_XXX'
```

When this NCL file is used, the DLL is loaded at the same time as the application. The `REGISTER_XXX` instruction enables the executable to register the names of the windows that belong to the DLL and update its internal tables. After this, the DLL's windows can be called using the `STROPEN*` and `STRCALL*` instructions. The `DEREGISTER_XXX` instruction frees the memory area used by the executable to register the names of the windows, which can no longer be called. You need to code `DEREGISTER_XXX` if you want to call another DLL containing window names that are identical to the ones in the previous DLL, which will be registered by another `REGISTER_XXX` instruction.

Example

```
; In the INIT event for the application's
```

```

; main window
REGISTER_SAMPLE

; While the application is running, call a window
; in the DLL SAMPLE
STROPEN "window_name",self%

; In the main window's TERMINATE event
DEREGISTER_SAMPLE

```

Data shared by DLLs

Within an application written in NCL and generated as DLLs, variables declared as GLOBAL can be shared without difficulty. However, variables can only be shared by an application and an external DLL if you use functions designed to access the global variables in the DLL. Two applications designed with NS-DK can exchange data by using shared memory, queues or a DDE mechanism.

Important

To be able to use your DLLs, you must place them in a directory specified by the PATH environment parameter in the AUTOEXEC.BAT file under Windows.



If you regenerate after testing your application, DON'T FORGET to update your DLLs in the directory specified by PATH (Windows) if it differs from the generation directory, otherwise the system will execute your old DLLs!

Using DLLs with an external resource file

An application that uses a DLL generated with an external resource file must load the DLL's .RES file explicitly when the application is initialized (in the main window's INIT event).

The resource file is loaded by using the INITRESOURCES instruction, which requires two parameters: the name of the .RES file (which will be searched for in the current directory or in the directories specified by the PATH parameter) and the default target name.

The INITRESOURCES instruction is located in the DLL 'NSnnLIB'. It must be declared in an NCL library by adding the following line:

```

INSTRUCTION INITRESOURCES CSTRING RESNAME, \
                        CSTRING TARGNAME \
EXTERNAL 'NSnnLIB.WINITRESOURCES'

```

Warning: if your application uses several DLLs generated with .RES files, you must first merge all the files into a single .RES file which will be loaded by INITRESOURCES.

Using DLLs Generated in C

To enable an application written in NCL to use external DLLs generated in C, you need to:

1. Create an NCL file that contains the EXTERNAL declarations of the functions in these DLLs.
2. Include the NCL file in the project's "Libraries" resources.
3. Place the DLLs in a directory specified by the PATH environment parameter in the AUTOEXEC.BAT file (Windows).

Make sure the NCL types that you specify in the NCL file are consistent with the types of the calling parameters and return codes in the C or Pascal functions.

Here are three examples of C functions together with their NCL declarations:

Example 1

Passing the address of a character string to be modified by the DLL function

NCL declaration

```
INSTRUCTION MODIFSTRING CSTRING @A$ \  
EXTERNAL 'MYDLL.MODIFSTRING'
```

C function

```
void far pascal ModifString (char * buf)  
{  
    int i;  
    for (i=0; buf[i] != '\0'; i++)  
    {  
        buf[i] += 1;  
        if (buf[i] > 'z')  
            buf[i] = ' '  
    }  
}
```

NCL call to the function

```
LOCAL CSTRING A$  
  
MOVE EF_ENTRY TO A$  
MODIFSTRING A$  
MOVE A$ TO EF_ENTRY
```

Notes

The way a variable is passed to a function (by value or by address) is only ever specified in the function's NCL declaration and never in the call.

Prefixing a variable name with a '@' indicates that the variable is being passed by address. This allows the contents of the variable passed as a parameter to be modified by the function.

This applies to the A\$ string in our example which corresponds to a pointer in the C function. Note that '@' is not specified when the MODIFSTRING function is called in the NCL code.

If there is no prefix before the variable name, this means that the variable is being passed by value. The variable is copied into the stack and the copy is passed to the function. As a result, the original value cannot be modified by the function.

Example 2

Passing a string by value to a function that returns a pointer

NCL declaration

```
FUNCTION RETURNUPC% (A$) EXTERNAL 'MYDLL.RETURNUPC'
```

C function

```
char tab[30];

char * far pascal ReturnUpc (char *str)
{
    int i;

    for (i=0; i < 30;i++)
        tab[i] = '\0';
    for (i=0; str[i]; i++)
    {
        if (str[i] >= 'a' && str[i] <= 'z')
            tab[i] = str[i] + 'A' - 'a';
        else
            tab[i] = ' ';
    }
    return tab;
}
```

NCL call to the function

```
LOCAL A$
LOCAL P%

MOVE ENTRY TO A$
MOVE RETURNUPC%(A$) TO P%
MESSAGE "Result:",MYSEG(P%).UPC
```

The segment is declared as a Segment resource as follows:

```
SEGMENT MYSEG
    CSTRING UPC(30)
ENDSEGMENT
```

Example 3

Passing a string by value to a function that returns a character string

NCL declaration

```
FUNCTION INVERTSTRING$ (a$, INT LG%(2)) \
```

```
EXTERNAL 'MYDLL.INVERTSTRING'
```

C function

```
char * far pascal InvertString (char * result,char *str,int lg)
{
    int i;
    for (i=0;i< lg;i++)
    {
        *(result+i) = str[lg-i-1];
    }
    *(result+lg) = 0;
    return result;
}
```

NCL call to the function

```
MOVE INVERTSTRING$( ENTRY, LENGTH ENTRY) TO ENTRY
```

Errors

If NS-Gen is started from the *Generation* dialog box, it reports any errors encountered during the various generation phases by displaying: "Generation terminated. Error(s) found" in the Processing field and providing a description of each error in the Errors list.

If NS-Gen is started from the DOS command line, errors will be displayed as each generation phase progresses. NS-Gen will terminate and display "Error(s) found during generation". You should redirect the messages output by NS-Gen to a file since the first few messages may disappear during the generation process.

During the source file generation phase, all files are checked and any errors detected are displayed. During the compilation phase, NS-Gen stops as soon as the compiler detects a fatal error.

Errors During the Source Generation Phase

These are usually syntax errors in the resources' NCL code, which are detected when the corresponding source file is generated.

They include:

Cannot find target XXXX


Cause	Neither the requested target nor the generic target can be found in the resource.
Corrective action	Open the resource and associate at least the generic target with it.

Cannot find resource XXXX

Cause	The resource has not been included in the project.
Corrective action	Check that the resource has been saved before starting the generation.

Unknown or invalid identifier

Cause	Syntax error in an NCL statement or reference to an unknown resource in the project.
Corrective action	The error description tells you where the error occurred: Window/Control/Event/Line of code.

Edit the event for the appropriate control or window and press the  button. This will run the syntax checker and automatically position your cursor on the error. Correct the error.



A sure way to avoid these errors during this phase is to always run the syntax checker on your NCL script when you write it and leave the *Display error* option in the *Action on undef symbol* field checked in the NS-Design *Miscellaneous* tab of the *Setup* box.

Errors During the Compilation Phase

Errors and Warnings

When the compiler analyzes your code, it reports two types of errors:

- ◆ Fatal errors, which prevent an object file from being generated.
- ◆ Warnings, which warn you of a potential problem that the compiler has managed to solve (e.g. implicit conversion). A warning does not prevent an object file from being generated but may result in unexpected behavior of the executable program.

Any fatal errors reported by the compiler terminate the generation process immediately. If this type of error occurs, contact Nat System as this means that there is a fault in the generator. You will be asked to supply the source file involved as well as the resources associated with this file so that the generator fault can be identified and corrected rapidly.

There are numerous types of warnings. The two most common ones are:

Data conversion supplied

Two data items in an expression are of different types, resulting in a type conversion for one of the data items.

Unreferenced local variable

A variable defined in the call parameters for a function is not used within the function.

These two types of warning do not require correction and do not alter the behavior of the executable.

If your code is too complex, the error "*Heap overflow*" may be reported by the compiler. Reduce the size of the corresponding resource file for the C source file involved.

Errors During the Link Phase

The linker may indicate that it cannot find a library ("*Cannot find library*"). If this occurs, it will wait for the pathname of the specified library as a reply to its message.

When you generate from NS-Design, this error has the effect of disabling the *Build* and *Rebuild* all buttons in the *Build/Run project* dialog box.



You need to press [Ctrl]+[Esc] to display the current task list. Select the task "Generating project..." and, once the window for this task has been opened, press [Ctrl]+C to stop the linker. After returning to the *Generation* box, you will notice the message "*Fatal error. Terminated by user*" displayed in the *Log* window.

If the generation was started by calling NS-Gen from the command line, press [Ctrl]+C as soon as the linker informs you that it cannot find a library.

Return to the *Configurations* dialog box at the *Compiler/Linker* tab and specify a correct pathname for the library. Then, restart the generation.

To avoid this problem, add the /BATCH option (supported by the Microsoft and IBM linkers) to the *Libraries* fields in the *Compiler/Linker* tab of the *Configurations* box, which will divert the message to the *Log* window. This will halt the generation process instead of blocking it.

Application Run-Time Errors

The most serious run-time error that can occur in your application is a memory violation error ("General Protection Fault" under Windows), indicating that an attempt has been made to write to an area of memory which does not belong to your application.

There are two main causes of this error:

1. The stack size is too small (especially if you have opened too many windows)
Corrective action: increase your stack size (*Stack size* field in the *Configurations* box at the *Compiler/Linker* tab) and also check that you are not using too many local STRING variables with the default size (255 bytes) when they could be much smaller.
2. The application tried to use a variable that is no longer valid; for example, it used a file handle to close a file that has already been closed.
Corrective action: examine your code at the location where the "G.P. Fault" or "Trap D" occurred and make sure that you are not using a handle incorrectly.

If your application behaves incorrectly (e.g. windows not opening and blocking), this may mean that system cannot swap correctly because it does not have enough space in the hard disk partition reserved for the swap area. Free some space in the hard disk partition or modify the swap area.

In general, pay particular attention to your application's GUI design: each graphical object opened consumes memory and your application may well run on one machine but not on another, depending on how their memory is configured. As a general rule, avoid opening more than 3 - 4 modal windows simultaneously (which also implies 3 or 4 call levels: think of the user!).

Chapter 6

Tips for running an application



***You will find in
this chapter***

- Practical advice on the execution of an application developed under NS-Design.

Contents

Characteristics of the Main Window	6-3
Launching an Application.....	6-4
Files used by an application	6-6
External Resources	6-7

Characteristics of the Main Window

Your application's main window determines its characteristics in the graphical environment.

Closing the application's main window does not terminate the program: memory resources, possibly CPU resources, and a session will still be allocated to it. To terminate the program and free the session, you need to check the *Quit on close* option in the main window's property pane.

Launching an Application

You can launch an application generated with NS-DK by selecting the *Build/Start* option.

Outside NS-DK, an NS-DK application can be launched in two ways, like any other graphical application:



- ◆ By typing its name in the Command line field displayed by the Windows *File/Run* option.
- ◆ By including it in a program group and selecting its name from the program list.



To launch and run the application correctly, the operating system needs to access certain DLLs. Their pathnames must be specified by the PATH variable in the AUTOEXEC.BAT file (Windows).

The following DLLs are required :

For all environments :

NSnnWPS.DLL

NSnnINIM.DLL

NSnnKMP.DLL

In addition to the following DLLs for each environment:



NSnnLIB.DLL

NSnnSWPS.DLL

NSnnMEM.DLL

NSnnPRN.DLL

NSnnTBLS.DLL

NSINTRT.DLL

plus the DLLs for any libraries used by the application: NSnnMISC.DLL, NSnnDATE.DLL, etc.

You can use start-up parameters, which are specified after the name of your executable on the command line.

See the NSMISC functions, PARAMCOUNT% and PARAMSTR\$, in the Libraries Programming Manual for more details on how to receive these parameters in your program.

A word of warning: when you test your application with NS-Test, the command line will not begin with your executable name but with 'NSTEST' followed by the start-up options for the tester! Consequently, your application needs to include code that searches for your program's name in the command line so that valid parameters are always retrieved, whether you run the application in test mode or generated mode.

Files used by an application

The text and bitmap files used in an application's List and Bitmap resources can either be loaded when the application is started up or only when the application displays the object that calls them.

To ensure that your application is portable in terms of the text or bitmap files that it uses, do not specify them with their full pathname -- use the NS-LST or NS-BMP environment parameters instead.

For example, to load a file into a list, code:

```
LOAD " (NS-LST) FILE.TXT" TO LB_LIST
```

and to specify the name of a bitmap file in the Released field of a Push button's Info box, code:

```
(NS-BMP) OPENR.BMP
```

When you install your executable on a machine, place the files that it uses in a specific directory and specify this directory using the NS-LST and NS-BMP environment parameters in the AUTOEXEC.BAT file (Windows). This will enable your program to retrieve the text or bitmap files from the appropriate directories.

External Resources

Each application generated with an external resource file comprises an executable file and an external resource file with the same name as the executable but suffixed by the .RES extension. For example, the SAMPLE.EXE executable will search for the data it requires in the SAMPLE.RES file.

The parameters that you need to pass to the executable so that it can run depend on the number of targets specified in the configuration file used to generate the source files (*Configurations* box) and the "Default target" that you have defined.

If several targets have been defined, they will appear in the .RES file obtained. The parameters passed to the executable depend on the target defined for the environment in which the executable will run:

<i>Environment</i>	<i>Parameter to pass</i>
Same as the default target	none
	or
	/TARGET:" target_name"
Different from the default target	/TARGET:" target_name"

You can also extract one or more targets from a .RES file using the NSResLib utility (the syntax for NSResLib is described in Chapter 5 of this manual).

The .RES file obtained can contain one or more targets. The number of targets in a .RES file depends entirely on how you want to supply and run your multi-target executable. In all cases, you need to specify the name of the new .RES file when you run the executable since, by default, the system searches for the original .RES file (with the same name as the executable).

If you have extracted several targets from the original .RES file, the following parameters should be passed to the executable

<i>Environment</i>	<i>Parameter to pass</i>
Same as the default target	/RES: new_res_name
	or
	/RES: new_res_name /TARGET:"target_name"
Different from the default target	/RES: new_res_name /TARGET:" target_name"

If you have extracted one target from the original .RES file, i.e. if each environment has its own .RES file, the executable should be started with the following parameter:

<i>Environment</i>	<i>Parameter to pass</i>
Same as the default target	/RES: env_res_name
Different from the default target	/RES: env_res_name



If you do not specify the pathname of the external resource file, the system will initially search for it in the current directory, then in the directories specified by the PATH variable (in AUTOEXEC.BAT under Windows).