NS-DK

Version 5.00 Edition 1

NS-Debug User Manual

Ref. n° DBG500UTP00033

Table of Contents

Chapter 1 Introduction

Chapter 2 Launching NS-Debug

Chapter 3 NS-Debug components

Chapter 4 Using NS-Debug

Chapter 5 Debugging applications

Chapter 6 Example of the use of NS-Debug

Appendix A Error messages

Organization's manual

This manual contains six chapters and one appendix :

Chapter 1 **Introduction**

This chapter presents NS-Debug.

Chapter 2 **Launching NS-Debug**

This chapter presents how to launch NS-Debug.

Chapter 3 **NS-Debug components**

This chapter presents the components of NS-Debug.

Chapter 4 **Using NS-Debug**

This chapter presents how to use NS-Debug.

Chapter 5 **Debugging applications**

The aim of this chapter is to provide you with some rules for using the debugger.

They concern both the behaviour of particular event-driven applications and the debugger’s functionalities that can be used as required for fine tuning an application.

Chapter 6 **Example of the use of NS-Debug**

This chapter shows, with many illustrations, a use of NS-Debug through an application.

Appendix A **Error messages**

This section presents the error messages listed.

Conventions

Typographic Conventions

Important term — Important terms are printed in **bold**.

Interface component — The names of windows, dialog boxes, controls, buttons, menus and options are printed in *italics*.

[F9] — Function key names appear in square brackets.

FILENAME — Filenames are printed in UPPERCASE.

`syntax example` — 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.

Operating Conventions

Activate the *XXX* \ *YYY* ... command — This means that you need to open menu *XXX*, then select the *YYY* command (option) on this menu.
You can perform this action using the mouse or mnemonic characters on the keyboard.

Activate the *XXX* \ *YYY* ... button — This means that 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.

Activate the *XXX* ... button — This means that you need to select button *XXX* 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

Chapter 1 Introduction

This chapter presents NS-Debug

You will find in this chapter

- How to configure NS-Debug

Contents

NS-Debug

NS-Debug is a tool designed for debugging and analyzing the behaviour of applications developed using NS-DK.

It enables you to execute an application step by step and locate the source of synctatic or logical errors (erroneous operators, uninitialized variables, etc.).

It also allows you to control the way in which the application is executed, to interrupt it at any point, to display the contents of variables and controls, to modify their contents and to add code dynamically to the script as it runs.

NS-Debug complements NS-Test during application development.

NS-Test is initially used to display the interface developed with NS-Design and to check the appearance and basic behaviour of the controls within the various windows.

NS-Test is then used in a subsequent phase to check that windows are linked together coherently and that the processing performed is correct. It accurately simulates the behaviour of the executable file that will be generated later.

NS-Debug should be used between these two phases to determine the cause of errors in behaviour or processing and ensure that the sequences of operations is correct.

NSDEBUG.EXE

NS-Debug consists only of a single executable file: NSDEBUG.EXE..

This executable uses a number of NS-DK's DLLs:

NSxxLIB.DLL, NSxxLIB1.DLL , NSxxRT.DLL and NSxxPRN.DLL,

where xx is the DLL version number.

It is assumed that NS-DK has been installed on your machine so that these DLLs are available and that the PATH environment parameter.

NSDEBUG.EXE can be run from NSDESIGN.EXE or independently.

It can be started up from the Run dialog box, displayed by selecting *the File/Run ..* menu option in Program Manager or from a Program group.

☞ You can install NSDEBUG.EXE in a directory of your choice. To avoid having to type this directory each time you run NSDEBUG, remember to specify it in the PATH environment parameter if this has not already been done.

☞ Where you start up NSDEBUG.EXE, the system may display an error message.

This may be due to a number of reasons:

- NSDEBUG.EXE is not in the specified directory
- NSDEBUG.EXE's pathname is not specified in PATH
- The DLL's shown above are not installed on the machine or their pathname is not specified by PATH.

Chapter 2 Launching NS-Debug

This chapter presents how to launch NS-Debug

You will find in this chapter

- How to launch NS-Debug

Contents

NS-Debug Syntax

The syntax you should use when launching NS-Debug is the following:

NSDEBUG [/PRJ:<ProjectName>] [MainName]

where

- **/PRJ<ProjectName>**

is the project for which a debugging session needs to be carried out.

ProjectName is the full path of the project. The name of the latter should be given the extension ".XNP".

If /PRJ:<ProjectName> is not specified, NS-Debug opens the last project used by NS-Design and saved in the NSDK.INI file when NS-Design was stopped.

```
NSDEBUG  /PRJ:C:\NSDK\SAMPLE.XNP
```

- **MainName**

Indicates the name of the window in which the debugging session should start.

If MainName is not specified, NS-Debug uses the name of the main window defined for the project. If no main window has been defined, its name should be specified via the Run/Restart menu once NS-Debug has been launched.

```
NSDEBUG  /PRJ:C:\NSDK\SAMPLE.XNP  CALC
```

Launching NS-Debug

NS-Debug can be launched from NS-Design.

All you have to do is to click on the option *Build/NS-Debug* or the start-up icon .

Launching NS-Debug is carried out automatically by clicking on this icon.

NS-Debug can also be launched separately from NS-Design:

from the *File/Run* menu in the Program Manager or from a program group.

Where the *Run* option from the *File* menu in the Program Manager is used, enter the syntax expected by NS-Debug in the command line of the *Run* dialog box:

```
NSDEBUG  /PRJ:project name   window name
```

Where it is launched from a program group, NS-Debug will by default use the last project saved by NS-Design and saved in the NSDK.INI file.

Debugging session

The term *debugging session* means the fine tuning phase of an application.

Once NS-Debug is launched, a debugging session starts by:

1. choosing a start-up window by selecting *Run/Restart* if this window is not specified when launching NS-Debug,

2. choosing the session start-up mode by selecting one of the following options from the *Run* menu: *Go, Trace, Step*.

A current session is stopped by normal termination of the application or by selecting from the *Run/Stop menu*. A new session can then be started without stopping NS-Debug, using another start-up window if required, by selecting it from the *Run/Restart* menu.

Chapter 3 NS-Debug components

This chapter presents the components of NS-Debug.

You will find in this chapter

- The components of NS-Debug

Contents

Main window

After a few seconds loading, the main NS-Debug window is displayed with a copyright warning message displayed in the foreground:

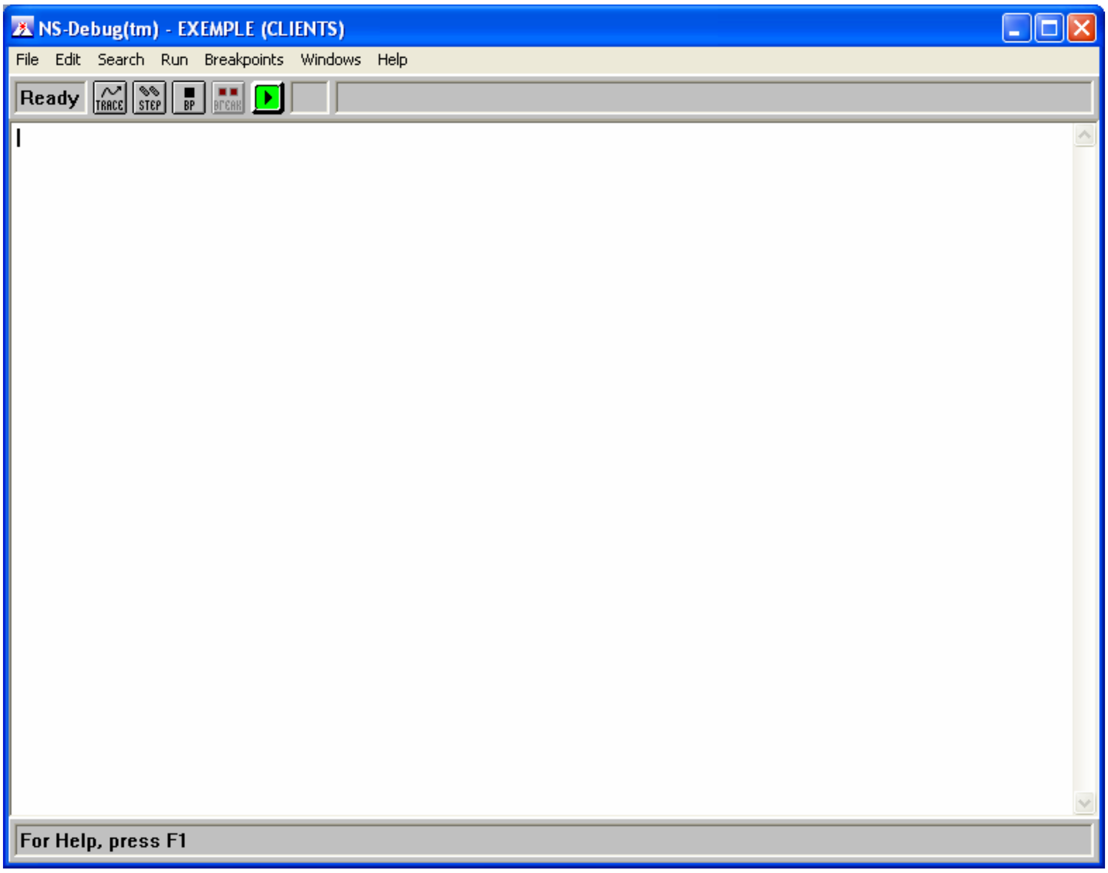

NS-Debug(tm) - EXEMPLE (CLIENTS)
File Edit Search Run Breakpoints Windows Help
Ready
TRACE
STEP
BP
BREAK
For Help, press F1

Figure - *NS-Debug at start-up*

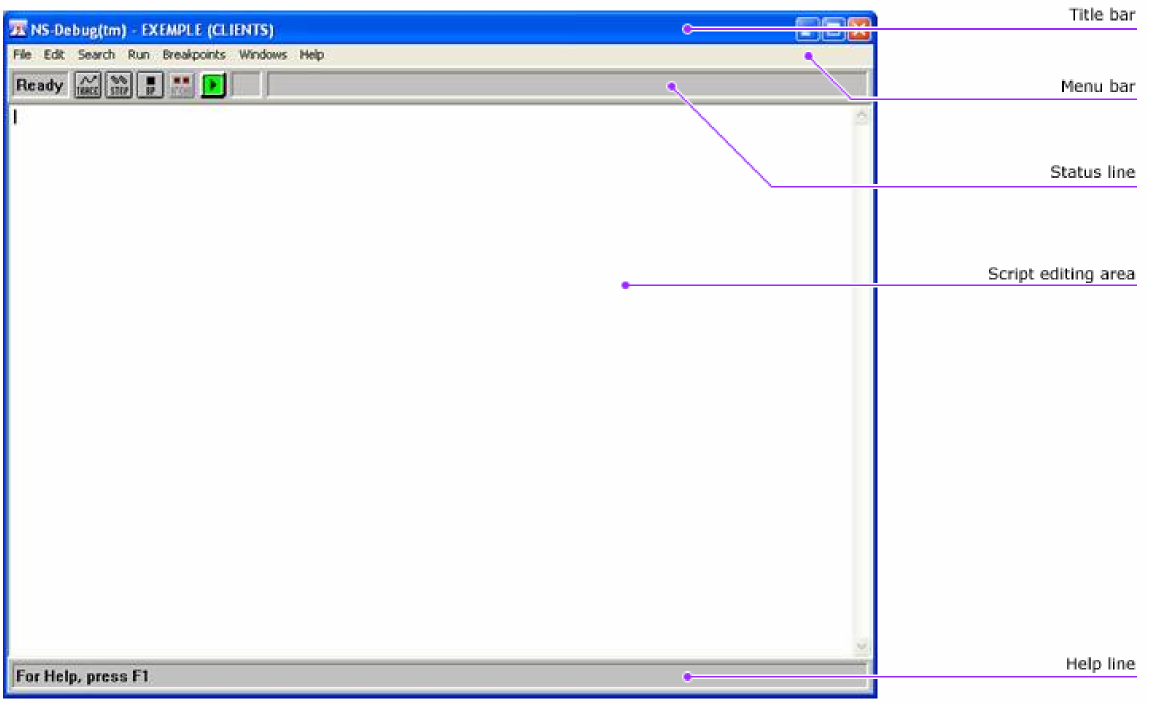


Figure - NS-Debug work space

The NS-Debug work space is composed of 5 parts:

1. The title bar, at the top.
2. The menu bar, under the title bar.
3. The status line, under the menu bar.
4. The script editing area.
5. The help line, at the bottom of the editing area.

The title bar

In this area **NS-Debug(tm)** is displayed, followed by the name of the project recognized by the debugger.

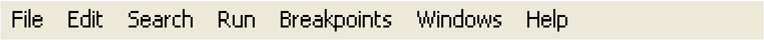
The name of the project is the one specified at start-up of the debugger or, by default, the one indicated in the NSDK.INI file.

The name of the window in which the debugging session should start appears in brackets to the right of the project name.

The start-up window is the one specified on launching NS-Debug or the one chosen from the *Run/Restart* menu.

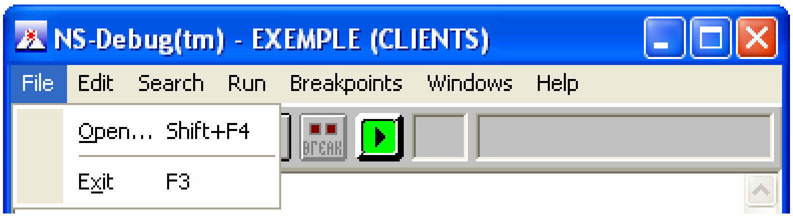
The menu bar

This bar, located below the title bar, contains all the functionalities offered by NS-Debug and which are presented in the form of menus.



The File menu

The *File* menu lets you edit event and library scripts and quit NS-Debug.



Open...

Lets you choose the script to display in the editing area.

Keyboard shortcut: [Shift] + [F4]

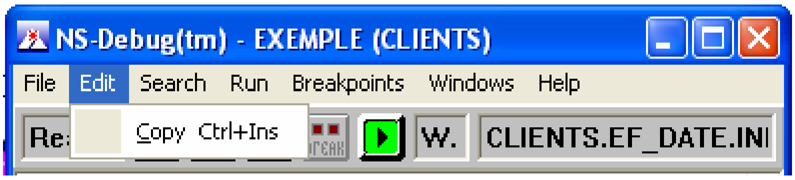
Exit

Stops NS-Debug.

Keyboard shortcut: [F3]

The Edit menu

The *Edit* menu lets you copy script extracts.



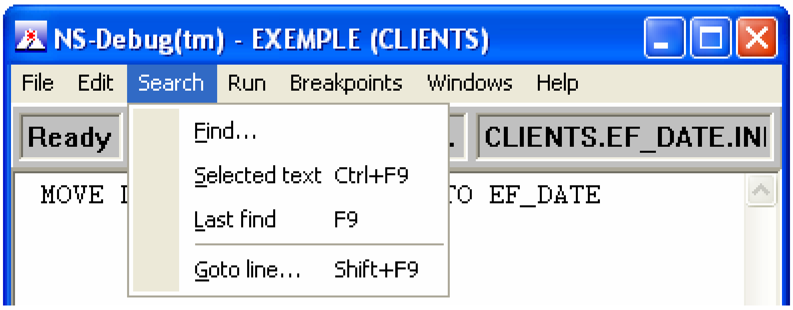
Copy

Copies the selection made in the script displayed in the editing area to the clipboard.

Keyboard shortcut: [Ctrl] + [Ins]

The Search menu

The *Search* menu lets you carry out searches for character strings in the script displayed on the screen and to position the cursor in this script.



Find...

Finds a character string.

Selected text

Finds another occurrence of the character string previously selected in the script.

Keyboard shortcut: [Ctrl] + [F9]

Last find

Repeats the previous search.

Keyboard shortcut: [F9]

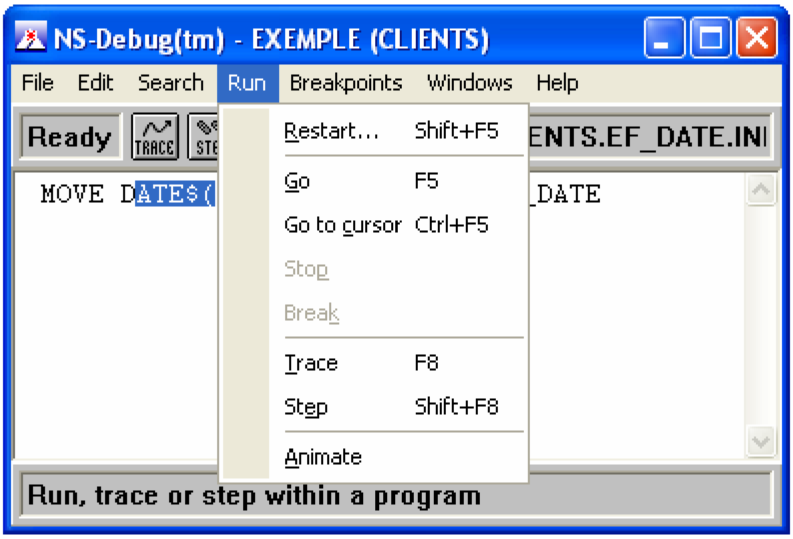
Goto line...

Positions the cursor to a given line.

Keyboard shortcut: [Shift] + [F9]

The Run menu

The *Run* menu is composed of the debugging session start-up options. These options let you launch a session in different modes, to interrupt the session or to follow its progress throughout the script.



```
MOVE DATE$(          _DATE
```

Run, trace or step within a program

Restart...

Lets you select the window in which the debugging session should start.

Keyboard shortcut: [Shift] + [F5]

Go

Launches execution of an application in Test mode.

Keyboard shortcut: [F5]

Go to cursor

Continues execution up to the line in the script where the cursor is positioned.

Keyboard shortcut: [Ctrl] + [F5]

Stop

Stops the debugging session by stopping execution of the current program.

Break

Requests the interruption of the program in course of execution.

Trace

Switches the debugger into Trace mode.

Keyboard shortcut: [F8]

Step

Switches the debugger into Step mode.

Keyboard shortcut: [Shift] + [F8]

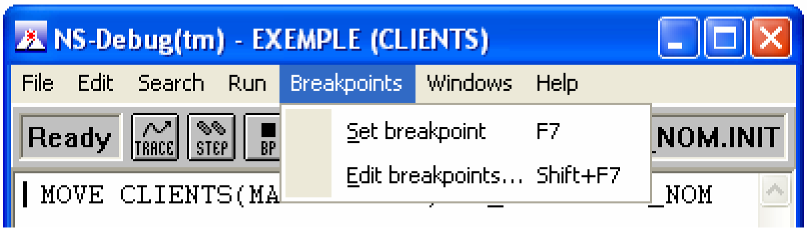
Animate

Specifies the mode of execution for the *Go* option.

If this option is ticked, the code of each script is displayed in the editing area in parallel with its execution.

The Breakpoints menu

The *Breakpoints* menu lets you set breakpoints and find out their properties.


NS-Debug(tm) - EXEMPLE (CLIENTS)
File Edit Search Run Breakpoints Windows Help
Ready TRACE STEP BP
Set breakpoint F7
Edit breakpoints... Shift+F7
NOM.INIT
MOVE CLIENTS(MA
_NOM

Set breakpoint

Set or delete a breakpoint on the current script line.

Keyboard shortcut: [F7]

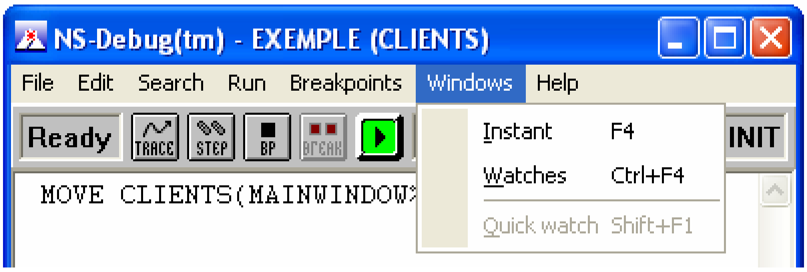
Edit breakpoints...

Displays the list of breakpoints.

Keyboard shortcut: [Shift] + [F7]

The Windows menu

The *Windows* menu comprises three options, each displaying a window. These windows are the NS-Debug components that let you display and change the application’s variables and controls and dynamically complete the script that is currently being executed.


NS-Debug(tm) - EXEMPLE (CLIENTS)
File Edit Search Run Breakpoints Windows Help
Ready TRACE STEP BP BREAK
Instant F4
Watches Ctrl+F4
Quick watch Shift+F1
INIT
MOVE CLIENTS(MAINWINDOW

Instant

Displays the *Instant window* letting you complete the script currently executing with one or more lines of NCL code.

Keyboard shortcut: [F4]

Watches

Displays the *Watches window* letting you display the content of variables, controls and the result of expressions.

Keyboard shortcut: [Ctrl] + [F4]

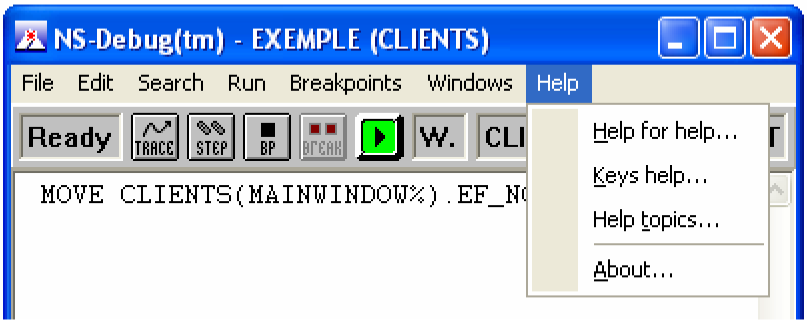
Quick watch

Displays a message box letting you quickly display the content of a variable, control or the result of an expression.

Keyboard shortcut: [Shift] + [F1]

The Help menu

This last menu lets you obtain on-line help on using the product.



```
MOVE CLIENTS(MAINWINDOW%).EF_N
```

Help for help...

Displays help on the operation of the on-line help.

Keys help

Displays a summary of the NS-Debug keyboard shortcuts.

Help index

Displays the on-line help index.

About...

Displays the NS-Debug version and the Nat System copyright.

The status bar

This line, located under the menu bar, is composed of eight items, including five icons.

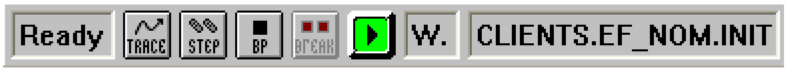


Figure - Status bar

1. The first field Ready shows the status of the debugger. It is either *ready to be used*(Ready), or *in the course of executing code* (Exec).

2. The *Trace* icon TRACE lets you execute the current line of script in Trace mode. Clicking on Trace takes you to the *Run/Trace* menu or press [F8].

3. The *Step* icon STEP lets you execute the current line of script in Step mode. Clicking on *Step* takes you to the *Run/Step* menu or press [Shift]+[F8].

4. The *BP icon* (BreakPoint) BP lets you set or delete a breakpoint in the current line of script. Clicking on *BP* takes you to the *Breakpoints/Set breakpoint* menu or press [F7].

5. The break *Break* icon BREAK lets you request an interruption to the current execution. Clicking on *Break* takes you to the *Run/Break*menu.

6. This button, subsequently called the *Run* button, lets you start execution of the application from the window, the name of which appears in the title bar.

   It then takes the following form:

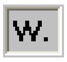
   ▶

   Clicking on this button takes you to the *Run/Go* menu or press [F5].

   Once execution is launched, this button changes aspect from green to red:

   ■

   Pressing it again takes you to the *Run/Stop* menu and stops current execution.

7. This field  W. indicates the type of resource, the script of which is displayed on the screen. The type is specified by a letter:

   W for Window.

   T for Template.

   L for Library.

8. This field gives the name of the script displayed in the editing area. This name can be:

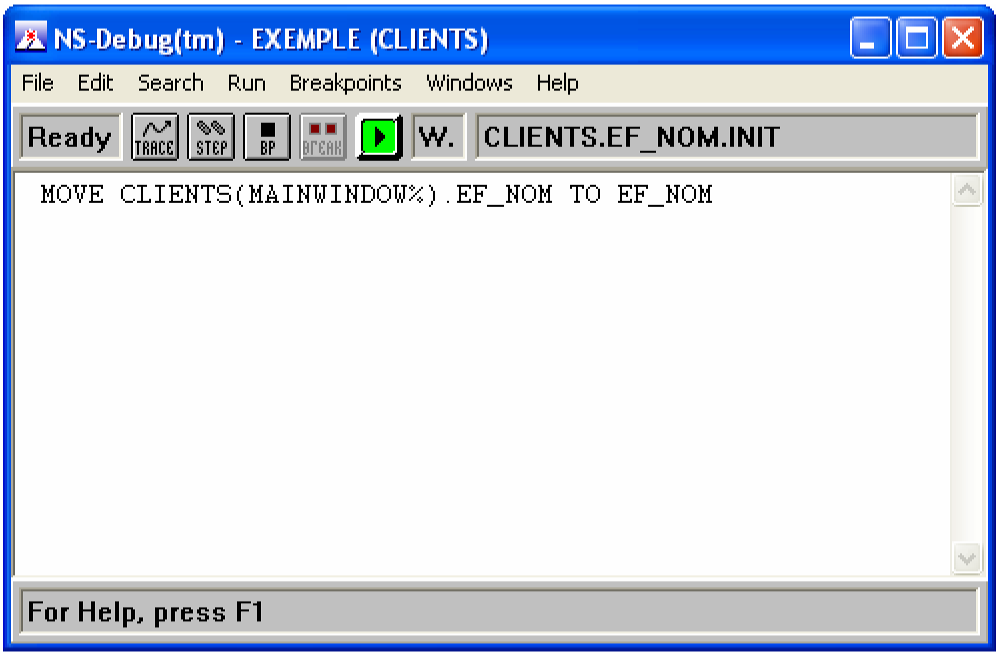
   - the name of a window event, in this case it is preceded by the name of the window.

   - the name of a control event, in this case it is preceded by the name of the window and the name of the control.

   - the name of a library, in this case it corresponds to the name of the library.

The script editing area

This zone, located below the status bar, lets you display the script of an event or of a library.


NS-Debug(tm) - EXEMPLE (CLIENTS)
File Edit Search Run Breakpoints Windows Help
Ready TRACE STEP BP break W. CLIENTS.EF_NOM.INIT

```
MOVE CLIENTS(MAINWINDOW%).EF_NOM TO EF_NOM
```

Figure - Script editing area

The initial size of this area can be reduced or increased by resizing the main NS-Debug window as normal using the mouse or the main window system menu.

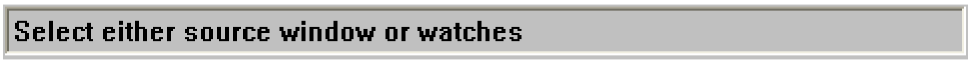
The script displayed is only accessible in read mode.

The various actions that can be carried out on the script displayed, called the current script in the rest of this document, are:

- displaying the whole script using the scroll bars.
- copying a part of the script to the clipboard.
- finding a character string.
- positioning the cursor to a given line.
- setting and deleting breakpoints.

The help line

This line, located below the editing area, contains a quick description of the currently selected menu option.



Select either source window or watches

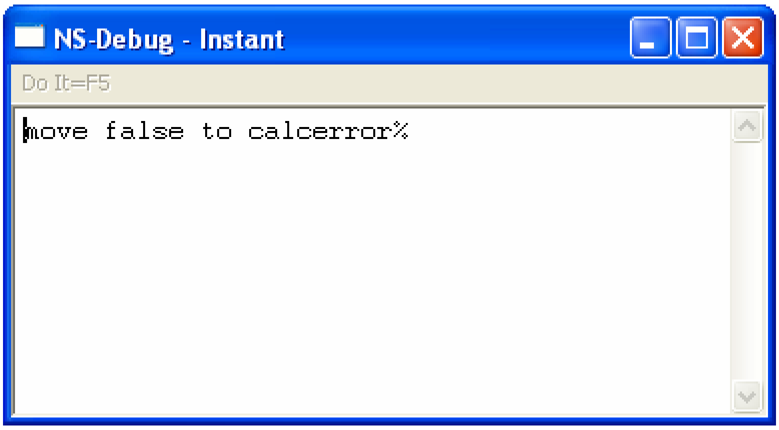
Figure - *Help line*

Instant Window

The *Instant window* is displayed by selecting the *Windows/Instant* menu or by pressing [F4].

It enables you to complete the current script once the debugging session has started.

This window is composed of a title bar, a menu bar and a script editing area:



```
move false to calcerror%
```

Figure - Instant Window

Do It

Executes the script displayed in the editing area.

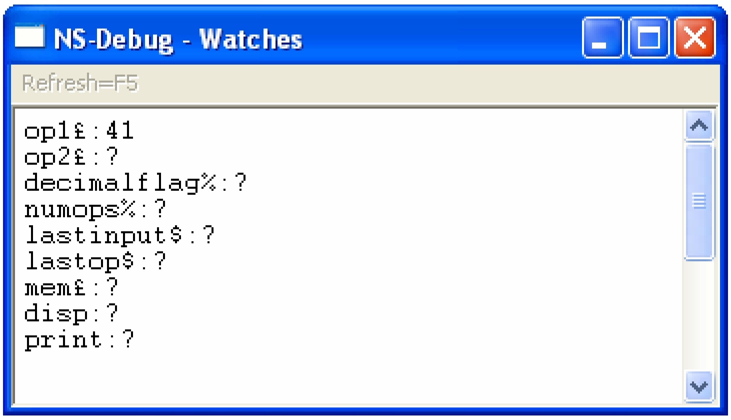
Keyboard shortcut: [F5]

Watches Window

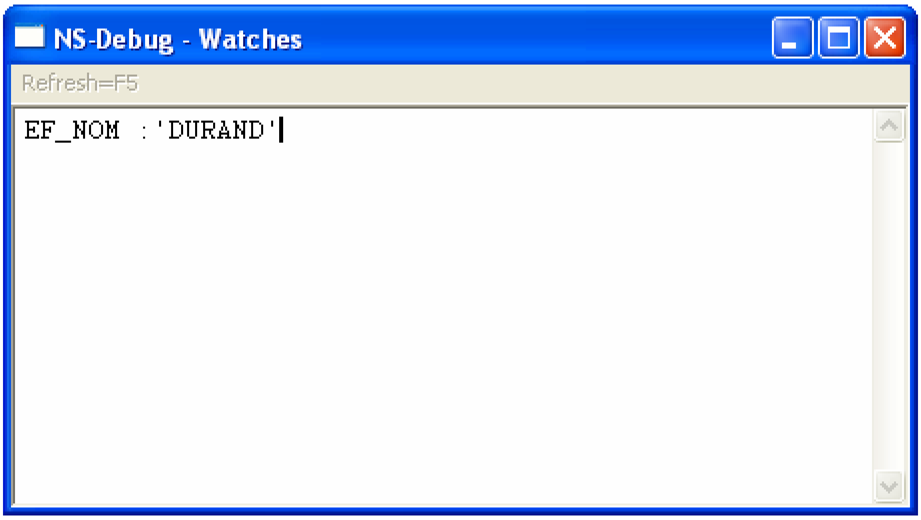
The *Watches window* is displayed by selecting the *Windows/Watches* menu or by pressing [Ctrl]+[F4].

It lets you display the content of variables, controls or the result of expressions for the application in the course of debugging.

It is composed of a title bar, a menu bar and a script editing area:



```
op1£:41
op2£:?
decimalflag%:?
numops%:?
lastinput$:?
lastop$:?
mem£:?
disp:?
print:?
```



```
EF_NOM :'DURAND'
```

Figure - *Watches Window*

Refresh

Updates the content of variables, controls and the result of expressions edited in the editing area.

Keyboard shortcut: [F5]

💡

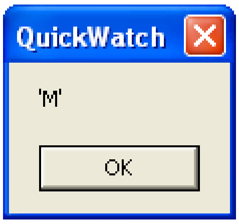
The reserved variable called __SP can be displayed in this window. This variable returns the current value of the stack (number of 16-bit integers remaining available in the stack). This value is reduced as variables are stored in the stack. This stack is the NS-Test __SP stack and therefore only gives an indication as to a situation that may cause a stack overflow (succession of numerous modal calls or numerous interleaved functions for example).

Quick watch message box

The *Quick watch* message box is displayed by selecting it from the *Windows/Quick watch* menu or by pressing [Shift]+[F1].

It lets you immediately obtain, temporarily, the content of a variable, control or the result of an expression, the title of which has been previously selected in the current script.

The *Quick watch* message box only contains the result of the request:



'M'

Figure - Example of the contents of the Quick watch box

Chapter 4 Using NS-Debug

This chapter presents how to use NS-Debug

You will find in this chapter

- How to start NS-Debug
- How to edit scripts
- How to check the execution of a program

Contents

Starting a debugging session

NS-Debug needs two pieces of information in order to start a debugging session:

- the name of the project to be taken into account.

- the window in which the debugging session should start.

The name of the project can only be specified when starting NS-Debug. NS-Debug has to be stopped, therefore, and then re-launched in order to start a debugging session on another project.

The name of the window may or may not be given as a parameter when launching NS-Debug. If it is not given, but only a main window was specified for the project, the name of this window is then taken into account.

Once the NS-Debug window displays, you can indicate a new start-up window using the *Run/Restart menu....* .

The *Run/Restart*... menu must be used when:

- NS-Debug was launched without giving the name of a window and no main window is specified for the project.

- a new debugging session must be launched using a new window.

Restarting a debugging session

In the same window that you started with

This type of restart requires the current session to be stopped first:

- by normal termination of the application during debugging.
- by the application being deliberately stopped (*Run/Stop* menu or clicking on the *Run* button).

The session then needs to be relaunched by:

- the *Run/Go* menu or by clicking on the *Run* button.
- the *Run/Trace* or *Run/Step* menus or by clicking on the corresponding icons.

In a new window

This restart may or may not be preceded by the current session being stopped, (*Run/Stop*) and is carried out using the *Run/Restart* menu.

Selecting *Run/Restart...* displays the *Set main window* dialog box:

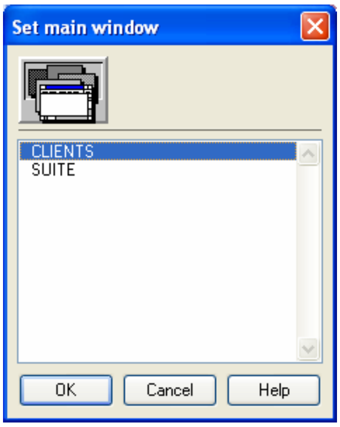


Figure - *Set main window dialog box*

The list in this dialog box contains all the windows for the current project.

The choice of start-up window is made by selecting it from the list then clicking on *OK*, or by double-clicking on its name.

The current selection is not changed if the dialog box is closed by clicking on *Cancel*.

The debugging session must then be relaunched by:

- the *Run/Go* menu or by clicking on the *Run button.*

- the *Run/Trace* or *Run/Step* menus or by clicking on the corresponding icons.

Editing scripts

Displaying scripts

The various scripts of the application to be debugged (event scripts, library scripts) can be displayed at any time.

Displaying them lets you:

- find out about processing before executing it.
- set or remove breakpoints.
- copy extracts (variable name, expression) to the clipboard.

Displaying a script is carried out using the *File/Open* menu... .

Selecting *File/Open...* displays the *Open…* dialog box:

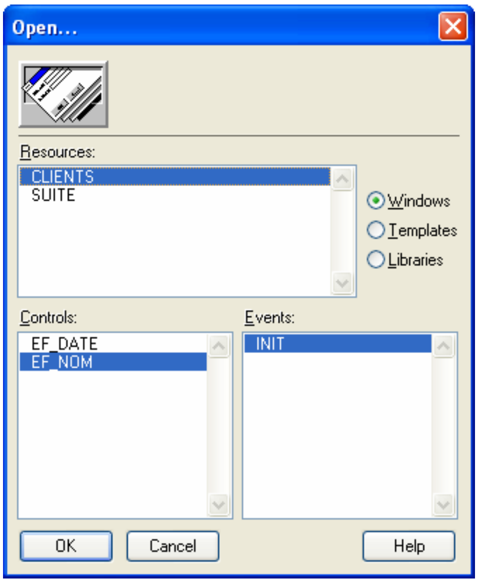


Figure - *Open… dialog box*

The *Resources* list lets you display the various project resources able to contain NCL script: *Windows*, *Templates* or *Libraries*. These are displayed by type, each type being selected by a radio button.

Each *Windows* and *Templates* resources script corresponds to a control or window event, also you should successively:

- select a resource from the *Resources* list.
- select a control or <*Client*> for the window from the *Controls* list displaying the controls for the previously selected resource.
- select an event from the Events list displaying the events for the previously selected control.

The library script is, on the other hand, selected directly from the *Resources* list.

The *OK* button closes the dialog box, accepting the choice of script. This is then displayed in the NS-Debug window editing area.

The *Cancel* button closes the dialog box without recognizing the selection.

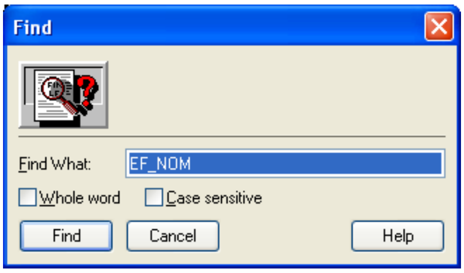
The *Help* button displays the on-line help relating to this dialog box.

Searching in a script

The *Find* menu lets you carry out searches for character strings in the script displayed in the editing area.

Search/Find...

Selecting *Find...* displays the *Find…* dialog box:



Find What: EF_NOM

□ Whole word □ Case sensitive

Figure - *Find dialog box*

This dialog box lets you specify the properties of the string to be found.

Find What

String to be found.

The previous search string or the current script selection is automatically offered when the box opens.

Input strings cannot exceed 31 characters.

Whole word

Option indicating that only whole strings, that is those preceded and terminated with a space character or tab, should be highlighted.

If the option is not ticked, any string identical to the string being sought is highlighted.

Case sensitive

Option indicating that the search only concerns strings that are an exact match for the string being sought, including upper and lower case characters.

Find

Closes the dialog box and launches the search.

This button is inactive while the *Find What* field is empty.

Cancel

Closes the dialog box without carrying out a search.

Help

Displays the on-line help relating to this dialog box.

Search/Selected text

Selected text (or [Ctrl]+[F9]) uses a string selected from the script as the search string. Selecting *Selected Text* immediately triggers the search for another occurrence of the string.

If no string was previously selected in the script, *Selected text* causes the *Find* dialog box to appear so that you can enter the string to be found.

Search/Last find

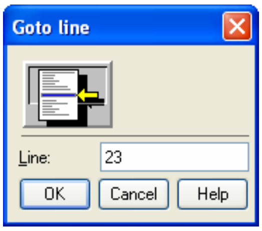
Last find (or [F9]) lets you repeat the previous search.

Pressing the [F9] key several times in succession lets you position to all the strings found in the script.

Positioning in a script

Search/Goto line... (or [Shift]+[F9]) lets you position the cursor to a given line in the script.

The number of the line must be given in the dialog box displayed after selecting *Goto line…*:



Line: 23

Figure - Goto line dialog box

Line

Line number, counted from 1, where the cursor should be positioned.

OK

Closes the dialog box and positions the cursor to the beginning of the line, the number of which is specified in *Line.*

If the number is higher than the number of lines in the script displayed, an error message tells you the limits to be observed.

This button is inactive while the field *Line* is empty.

Cancel

Closes the dialog box without positioning the cursor.

Help

Displays the on-line help relating to this dialog box.

Program execution control

Debugging modes

Debugging a program can be carried out in three modes:

- Test mode, where the program is executed until a “breakpoint” is reached or it is temporarily interrupted.

- Trace and Step mode, where the program is interrupted after each line of code is executed.

Test mode lets you execute parts of the code quickly that you know do not have bugs.

The aim of debugging in Test mode is to concentrate your investigations on parts of the code not yet tested or containing errors.

Trace mode and Step let you execute the program code line by line.

In Trace mode, any NCL function or instruction encountered in a line of code is also executed line by line. It’s script is displayed in the editing area.

In Step mode, any NCL function or instruction encountered in a line of code is executed globally without displaying its code. The current script remains displayed in the editing area.

The aim of debugging in Trace or Step mode is to find out the exact behavior of the application as execution proceeds. It lets you find out which lines of code are actually executed and to know the values of the variables and controls manipulated at any given moment.

In general these debugging modes are used alternately during a debugging session, Test mode being used when a part of the code that has already been evaluated is executed, Trace or Step mode being activated as soon as detailed information on the program needs to be obtained.

Enabling debugging modes

Mode enabled at start-up

Once the start-up window has been specified, selecting an option from the *Run* menu or clicking on one of the icons on the status bar determines the debugging mode that is enabled.

Mode enabled	**Menu**	**Keyboard shortcut**	**Icon**
Test mode	Run/Go	[F5]	Run
Trace mode	Run/Trace	[F8]	Trace
Step mode	Run/Step	[Shift] + [F8]	Step

Changing debugging mode

During a debugging session, you can change debugging mode at any time.

In all cases, changing mode cannot take place until execution of the program is interrupted. Interruption can be achieved in several different ways:

- after a line of code has been executed if the mode enabled is Trace or Step.
- when a previously set breakpoint is reached.
- through an explicit request.

An explicit request for interruption is made by selecting *Run/Break*, or by clicking on the *Break* icon, when Test mode is enabled. In this case the program is interrupted at the next line of code executed and the script concerned is displayed in the editing area.

Once execution of the program has been interrupted, the mode in which execution should continue is determined by the menu options or icons described in the previous paragraph.

Debugging in Test mode

Test mode only lets you interrupt the program at the start of parts of code that may contain errors or to find out which event is enabled for an action concerning the current application window.

NS-Debug functionalities to be used in this mode are therefore:

- setting breakpoints.
- request for interruption of execution at the next line of code executed (Break).

As soon as the program is interrupted, you also have access to other NS-Debug functionalities:

- execution of the script up to a given line of code (*Go to cursor*).
- displaying the content of variables, controls or expressions (*Watches* and *Quick watch*)
- completing the current script (*Instant*).
- editing a script other than the current script in order to set a breakpoint (*File/Open*).
- relaunching execution in Test, Trace or Step mode.

Debugging in Trace or Step mode

Debugging is carried out in Trace or Step mode when you want to find out the lines of code actually executed and the values of the variables or controls after a line of code has been executed.

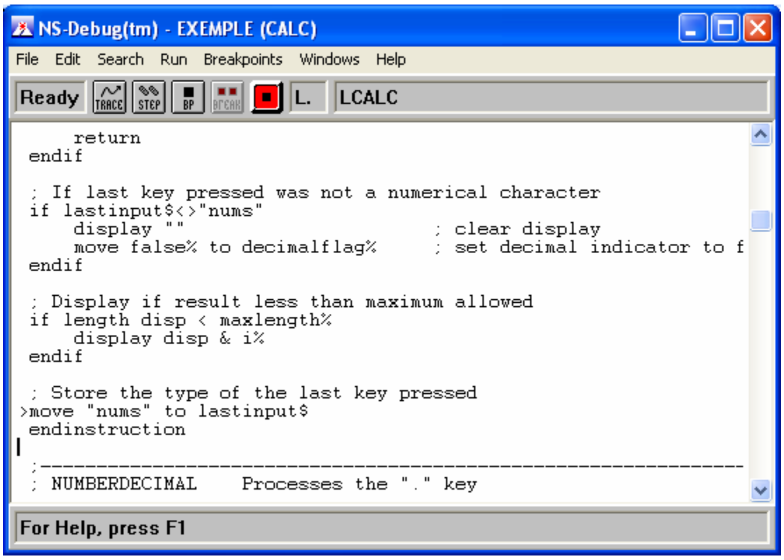
The only difference between these two modes concerns the execution of functions or instructions encountered in a line of code:

- Trace mode lets you debug the content of the function or instruction: the program is interrupted at the first line of code of the function or instruction and its script is displayed in the editing area.
- Step mode (Step) executes the whole of the code of the function or instruction encountered in a line of code: the program is interrupted at the next line of code of the same script.

It is possible to switch between Trace mode and Step mode during execution of a script, their use only depending on the code you wish or don’t wish to execute line by line.

The two modes behave in the same way when the line of code to be executed does not include a call to a function or instruction.

As soon as the debugger enters one of these two modes, the status bar indicates "Ready" and an indicator in the form of a greater-than sign appears at the start of the next line of code to be executed:


NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Ready
TRACE
STEP
BP
BREAK
L.
LCALC
For Help, press F1

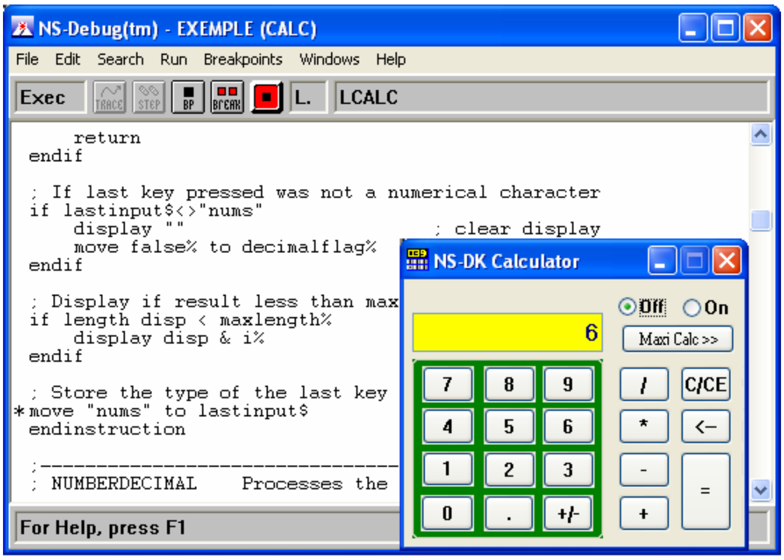
```
     return
 endif

 ; If last key pressed was not a numerical character
 if lastinput$<>"nums"
     display ""                        ; clear display
     move false% to decimalflag%       ; set decimal indicator to f
 endif

 ; Display if result less than maximum allowed
 if length disp < maxlength%
     display disp & i%
 endif

 ; Store the type of the last key pressed
>move "nums" to lastinput$
 endinstruction

 ;--------------------------------------------------------------
 ; NUMBERDECIMAL    Processes the "." key
```

Figure - Indicator of the line of code to be executed

Each time *Run/Trace* or *Run/Step* is selected (or the corresponding icons are clicked on) the current line of code is executed and the indicator is positioned to the next line of code to be executed.

The indicator changes to a star under Windows and the status bar shows "Exec" when the current line of code is being executed.

The shape of the indicator is not changed when the last line to be processed has been executed or if Test mode has been enabled. The current application window is obtained when the focus and the debugger are waiting for a new event to occur:



```
     return
 endif

 ; If last key pressed was not a numerical character
 if lastinput$<>"nums"
     display ""                          ; clear display
     move false% to decimalflag%
 endif

 ; Display if result less than max
 if length disp < maxlength%
     display disp & i%
 endif

 ; Store the type of the last key
*move "nums" to lastinput$
 endinstruction

 ;--------------------------------
 ; NUMBERDECIMAL    Processes the
```

Figure - Line indicator of the code currently being executed

When the program is interrupted, you also have access to the following NS-Debug functionalities:

- execution of the script up to a given line of code (*Go to cursor*).
- displaying the content of variables, controls or expressions (*Watches* and *Quick watch*)
- completing the current script (*Instant*).
- editing a script other than the current script in order to set a breakpoint (*File/Open*).
- relaunching execution in Test, Trace or Step mode.

Interruption

Interrupting execution of the program during debugging lets you take control of the program via the debugger.

Interruption changes the debugger from the "Exec" status (execution of the program in Test mode) to the "Ready" status.

When NS-Debug is in the “Ready” state, you have access to the functionalities described in the previous paragraph in order to continue the debugging session.

In Test mode, an interruption is obtained by:

- activation of a breakpoint.
- selecting Run/Break from the menu, or clicking on the Break icon in the NS-Debug main window.

In Trace and Step mode, an interruption takes place immediately after execution of the current line of code.

Breakpoints (Breakpoints)

A breakpoint lets you indicate at which line of code execution of the program should be interrupted.

Breakpoints can be set before starting a debugging session, during an interruption to execution, or during execution when no event is generated.

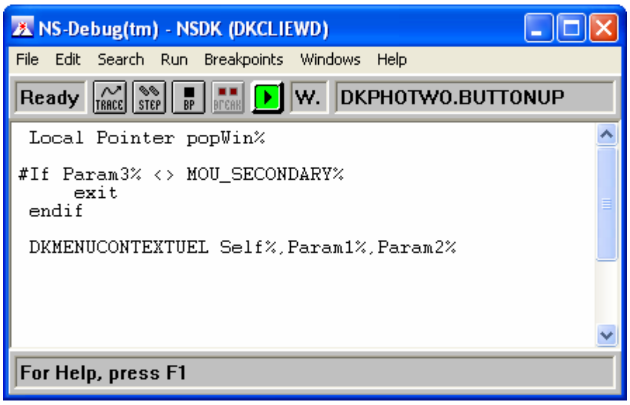
NS-Debug lets you set up to 32 breakpoints.

Setting a breakpoint

A break point is set by:

- displaying the desired script using *File/Open...*
- positioning the cursor to the character in any line of the script where the breakpoint is to be set.
- selecting *Breakpoints/Set breakpoint* (or [F7]) or clicking on the BP icon.

An indicator in the form of the number sign under Windows then appears at the beginning of the line to indicate that a break point has been set:


NS-Debug(tm) - NSDK (DKCLIEWD)
File Edit Search Run Breakpoints Windows Help
Ready
TRACE
STEP
BP
BREAK
W.
DKPHOTWO.BUTTONUP
For Help, press F1

```
 Local Pointer popWin%

#If Param3% <> MOU_SECONDARY%
     exit
 endif

 DKMENUCONTEXTUEL Self%,Param1%,Param2%
```

Figure - A breakpoint is set

Deleting a breakpoint

Two methods let you delete a breakpoint.

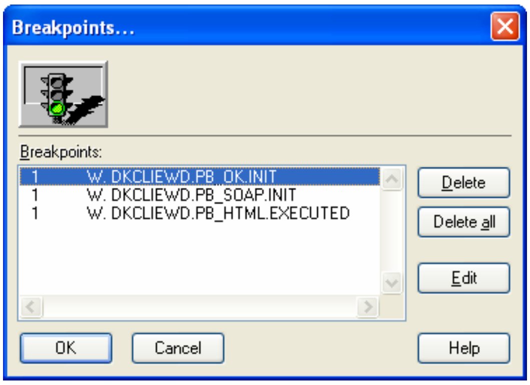
One, the direct method, is to carry out the same operations as for setting a breakpoint: *Set breakpoint* (or [F7]) and the BP icon delete the breakpoint set in the current line.

The other, the indirect method, is to use *Breakpoints/Edit breakpoints*... (or [Shift]+[F7]) in order to look up the list of breakpoints and to delete those that are not required. Using the dialog box then displayed is described in the next paragraph.

Editing breakpoints

The *Breakpoints/Edit breakpoints menu*... (or [Shift]+[F7]) lets you list all the breakpoints, delete some or all of them, and display a script containing a breakpoint.

Selecting *Edit breakpoints* causes the *Breakpoints*... dialog box to appear:



```
1    W. DKCLIEWD.PB_OK.INIT
1    W. DKCLIEWD.PB_SOAP.INIT
1    W. DKCLIEWD.PB_HTML.EXECUTED
```

Figure - Breakpoints… dialog box

Breakpoints

A list of all the break points that have been set.

Each line in the list contains information relating to a breakpoint:

- the line number in the script where the breakpoint is located.

- the name of the script.

The name of the script depends on the resource to which it belongs:

- for a *Windows* or *Templates* resource, it is composed of the prefix W. or T. followed by the name of the resource, the name of the control, then the name of the event.

- for a *Libraries* resources, it is composed of the prefix L. followed by the name of the library.

Delete

Deletes the selected breakpoint from the *Breakpoints* list.

Delete all

Deletes all the breakpoints from the list of *Breakpoints*.

Edit

Closes the dialog box then displays the script containing the breakpoint selected from the list of *Breakpoints*. The cursor is then positioned to the breakpoint.

⚠ Closing the dialog box by clicking on *Edit* renders deleting breakpoints inoperable.

OK

Closes the dialog box and deletes the breakpoints in the various scripts selected from the *Breakpoints* list by *Delete* or *Delete all*.

Cancel

Closes the dialog box without recognizing the deletions.

Help

Displays the on-line help relating to this dialog box.

Action relating to the data of an application

Display

Watches Window

The *Watches window* lets you display data content (local or global variables, controls) or the result of an expression. This display is retained and updated throughout the debugging session.

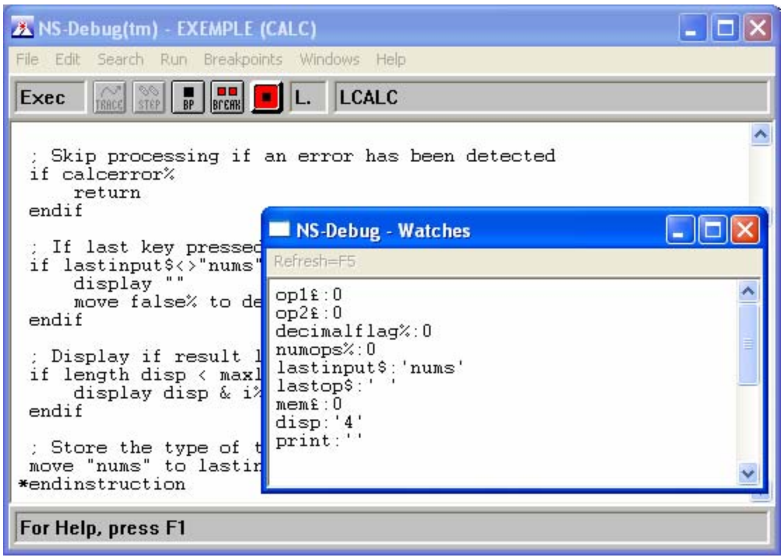
An expression is the representation of a value. It may be composed of assignment instructions, calculation functions, operators, variables or numeric values.

The *Watches window* is opened by selecting the *Windows/Watches* menu or by pressing [Ctrl]+[F4].

On opening, the size and position of the *Watches* window are predefined. They can be changed at your convenience so as not to, for example, overlap other windows that are part of NS-Debug. These changes are then kept until NS-Debug is stopped, whatever the number of sessions carried out before stopping.

The content of an item of data or an expression is obtained by typing its name or text in the editing area and then pressing the [Enter] key or [F5], the keyboard shortcut for the *Refresh* menu.

The content is displayed following the name of the data item or expression, preceded by the “:” character:


NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Exec
TRACE
STEP
BP
BREAK
L.
LCALC
NS-Debug - Watches
Refresh=F5
For Help, press F1

```
 ; Skip processing if an error has been detected
 if calcerror%
     return
 endif

 ; If last key pressed
 if lastinput$<>"nums"
     display ""
     move false% to de
 endif

 ; Display if result l
 if length disp < maxl
     display disp & i%
 endif

 ; Store the type of t
 move "nums" to lastin
*endinstruction
```

```
op1£:0
op2£:0
decimalflag%:0
numops%:0
lastinput$:'nums'
lastop$:' '
mem£:0
disp:'4'
print:''
```

Figure - *Example of a display of the contents*

The names of the data items and expressions should be entered in the editing area, one per line. This input can be carried out using the clipboard: copy from the script displayed in the main window (menu *Edit/Copy* or [Ctrl]+[Ins]) then write by pressing [Shift]+[Ins].

Various errors may be displayed if input is incorrect:

- *Syntax error* if the input is neither data nor an expression.
- *xxx expected* (xxx can be End of line, '(', ')', etc..) if the input is not a complete expression.
- *Unknown identifier* if the input is not the full name of a control.
- *Invalid type for name* if the size of a variable is omitted or incorrect.

A null value is displayed for any non-significant variable like, for example, a local variable no longer corresponding to the script being executed or an unknown variable.

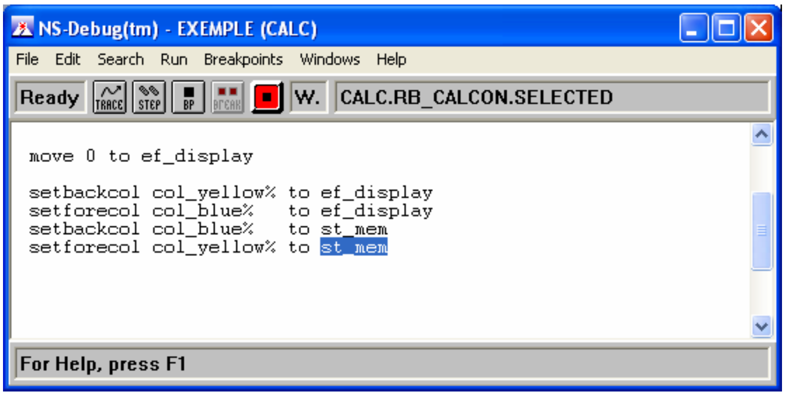
The values displayed in the *Watches* window are updated after execution of each line or part of the script.

⚠ While execution of the program is not interrupted (Exec status of the debugger), the values displayed in the *Watches* window are not significant: they correspond to the values of data and expressions updated during the last Ready status of the debugger.

Quick watch message box

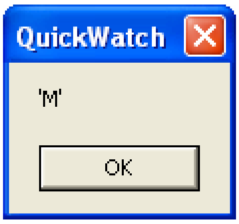
The *Quick watch message box* gives you an immediate display of the contents of a data item or the result of an expression previously selected from the current script.

This message box is opened by selecting the *Windows/Quick watch menu* or by pressing [Shift]+[F1].


NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Ready
TRACE
STEP
BP
BREAK
W.
CALC.RB_CALCON.SELECTED
For Help, press F1

```
move 0 to ef_display

setbackcol col_yellow% to ef_display
setforecol col_blue%   to ef_display
setbackcol col_blue%   to st_mem
setforecol col_yellow% to st_mem
```

Figure - Selecting an expression



'M'

Figure - Quick watch message box

Using *Quick watch* is only possible during an interruption to execution of the program.

In the case of an expression, the result displayed is only significant if the elements in this expression have actually been updated by execution of the preceding lines of code.

Unlike the *Watches* window, the result displayed is lost as soon as the box is closed. *Quick watch* is only used, therefore, to display data and results, the value of which is only of interest to you during execution of the current script.

Various errors may be displayed if the selection made is not correct:

- *Syntax error* if the selection is neither data nor an expression.

- *xxx expected* (xxx can be End of line, '(', ')', etc..) if the selection does not relate to a complete expression.

- *Unknown identifier* if the selection does not relate to the full name of a control.

- *Invalid type for name* if the size of a variable is omitted or incorrect.

A null value is displayed for any non-significant data. This is particularly the case for an unknown variable due to the incomplete selection of the variable name.

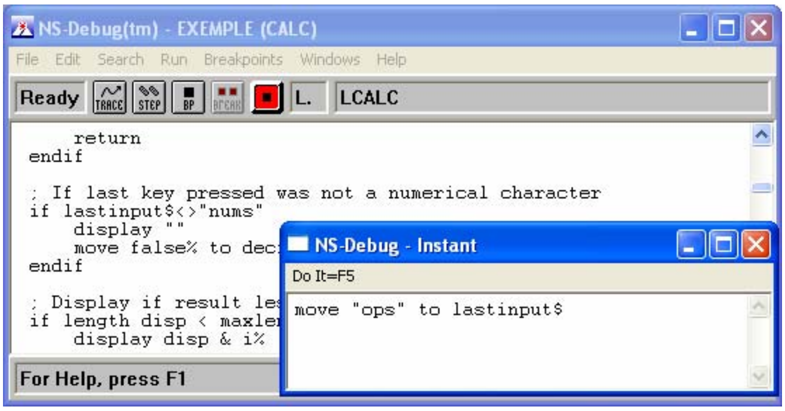
Modification

The *Instant window* lets you change the contents of the data (variables or controls) and complete the script currently executing.

This window is opened by selecting the *Windows/Instant menu* or by pressing [F4].

On opening, the size and position of the *Instant* window are predefined. They can be changed at your convenience so as not to, for example, overlap other windows that are part of NS-Debug. These changes are then kept until NS-Debug is stopped, whatever the number of sessions carried out before stopping.

Modification and completion of a script is carried out by writing lines of NCL code in the editing area of the Instant window. Thus changes to a variable carried out by MOVE, the insertion of a string in a List Box by INSERT AT END, etc…


NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Ready
TRACE
STEP
BP
BREAK
L.
LCALC
For Help, press F1
NS-Debug - Instant
Do It=F5

```
    return
endif

; If last key pressed was not a numerical character
if lastinput$<>"nums"
    display ""
    move false% to dec
endif

; Display if result les
if length disp < maxle
    display disp & i%
```

```
move "ops" to lastinput$
```

Figure - Example of use of the Instant window

Writing one or more lines of code can be done at any time in the editing area of the Instant window. However, execution of the code cannot take place until the program is interrupted: “Ready” status for the debugger and *Do It* in the active Instant window.

The line or lines of code showing in the editing area of the Instant window should be seen as the contents of an instruction.

Execution of this “instruction” takes place:

- by selecting the *Do It* menu or by pressing [F5].

- **before** execution of the current line of script at which the program is interrupted.

An error message is displayed when the code being executed is incorrect. This message tells you the position of the error in the script and its cause.

The details displayed are identical to those displayed by the NS-Test tester during syntax verification.

The Instant window script is executed in the context of the current script. Due to this, you can only access variables and controls actually visible at that moment. You cannot therefore:

- change the content of a control for a window that is not open.

- access a control directly if the current script is that of a function or instruction (in that case you need to specify the name and the window handle to which it belongs).

- change a local variable of an event other than the one displayed in the current script.

Animate

Animate is an option in the *Run menu.* It is a characteristic of Test mode launched by *Run/Go*.

The *Animate* option is enabled when a mark (Check mark) appears to its left. Enabling and disabling alternate each time Animate is selected.

Animate enables you simultaneously to launch an application in Test mode and to display the code being executed in the editing area.

You can thus check the lines of code actually being executed during processing or find out where in the code an infinite loop occurs.

Animate can only be disabled when the program is interrupted by selecting *Run/Break* or by clicking on the *Break icon*.

☞ When an infinite loop occurs, the only means of stopping it is to have already ticked the *Animate* option. This implies, therefore, that you feel such a loop is possible before continuing the debugging session in Test mode.

Stopping a debugging session

A debugging session is stopped by one of the following actions:

- selecting the *Run/Stop* menu,
- clicking on the *Run* button.
- a request to start a new session by selecting the *Run/Restart* menu...
- stopping NS-Debug by selecting the *File/Exit* menu.

In the first three cases, the fact that the session has stopped is indicated by the *Run* button changing to green and “*Program terminated...*” being displayed in the help line.

Chapter 5 Debugging applications

The aim of this chapter is to provide you with some rules for using the debugger.

They concern both the behaviour of particular event-driven applications and the debugger's functionalities that can be used as required for fine tuning an application.

You will find in this chapter

- How to debug applications
- Precautions that need to be taken using NS-Debug

Contents

Event-driven applications

Applications developed with NS-DK are of the event-driven type: all code executed is due to the occurrence of an event resulting directly or indirectly from an action on the part of the user of the application.

The behaviour of such applications determines the way in which the debugger is used.

In Test mode, the application windows being debugged always have the focus. An interruption to execution is only possible if a previously set breakpoint is encountered or if an interruption is explicitly requested by giving the focus to the NS-Debug window and by selecting *Run/Break*.

In Trace mode or Step mode, the current application window only regains the focus during execution of the current line of code or when the current event has been completely processed.

Each time the debugger interrupts the program, "Ready" is displayed in the status bar. It is then possible to use NS-Debug’s various functionalities, then continue the session in your mode of choice.

As soon as "Exec" is displayed in the status bar it is impossible to display or modify variables or controls because the application is waiting for an action from the user and an event to be generated. The only options then are to edit the script and set or delete breakpoints.

Using the debugger

Starting a debugging session

At the start of fine-tuning an application, launch NS-Debug by only specifying the name of the project.

Once NS-Debug has started, you can specify the window concerned by the debugging session using the *Run/Restart* window. Without leaving NS-Debug you can thus designate the windows and associated processing one after the other.

When debugging each unit has been carried out, overall debugging of the application can commence. The application main window can then be specified when launching NS-Debug; in principle this should never be changed during each debugging session.

Debugging mode to use

The mode to use depends essentially on the degree of progress that has been made in debugging.

Initially, Trace and Step modes help you understand the progress of the various processes involved. Using these modes lets you fully fine tune each separate unit of processing (function, instruction, succession of events, succession and display of windows) outside the general context of the application.

It is only once the parts of the code have been fine tuned independently of each other that you can use Test mode and set breakpoints ahead of certain parts of the code that you have not been able to debug without launching the whole application.

Components of NS-Debug

The other NS-Debug windows enable you to display and change application data during debugging.

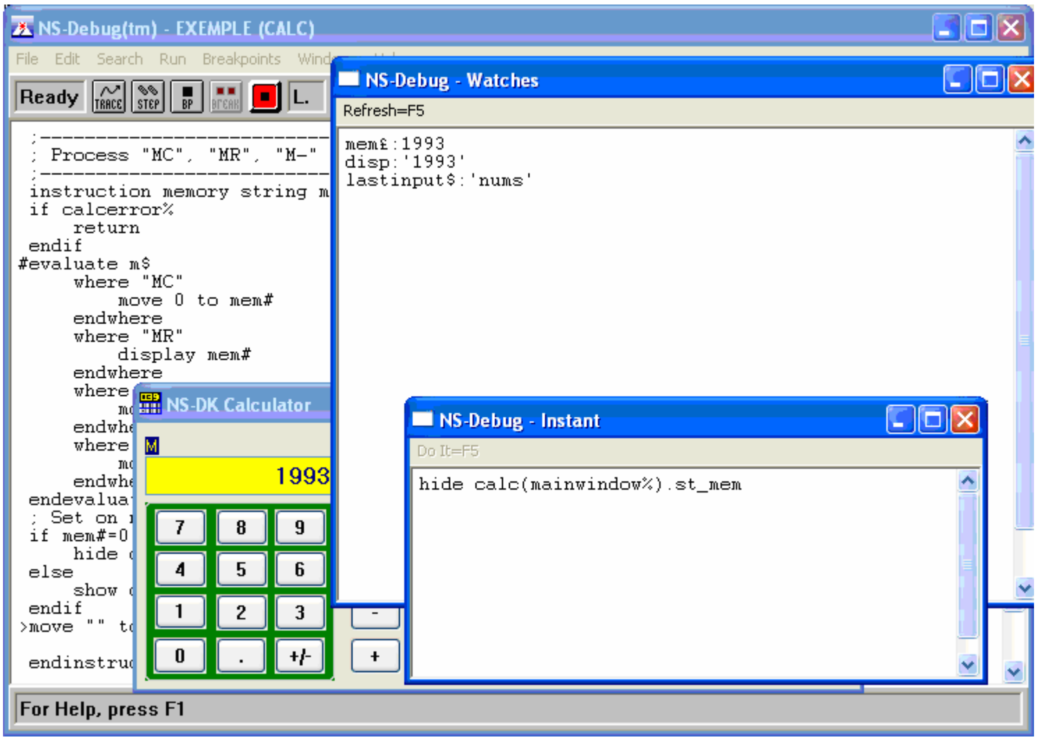
Use the *Watches* window each time the program is interrupted to find out the content of global variables and expressions that are regularly evaluated.

Use the *Quick watch* message box for all the variables and expressions, the content of which is only significant at a given moment.

Use the *Instant* window to change the content of a variable or control and thus correct an incorrect result, the cause of which you have detected. You can thus continue debugging and only make changes in the code when you detect a more serious problem for which correction is essential.

The *Instant* window can also be used to change the result of a calculation in order to check the correct processing of an error situation, the occurrence of which is difficult to reproduce.

It can be useful to have the three NS-Debug windows on the screen simultaneously. In this case, a solution is to open the *Watches* and *Instant* windows as soon as the main NS-Debug window is displayed then reposition and resize them for the period of the debugging session.



```
 ;-------------------------
 ; Process "MC", "MR", "M-"
 ;-------------------------
 instruction memory string m
 if calcerror%
     return
 endif
#evaluate m$
     where "MC"
         move 0 to mem#
     endwhere
     where "MR"
         display mem#
     endwhere
     where
         m
     endwhe
     where
         m
     endwhe
 endevalua
 ; Set on 
 if mem#=0
     hide 
 else
     show 
 endif
>move "" t
 endinstru
```

```
mem£:1993
disp:'1993'
lastinput$:'nums'
```

1993

```
hide calc(mainwindow%).st_mem
```

Figure - *Using NS-Debug windows*

Precautions

Debugging event-driven applications requires a minimum of precautions and understanding of the events generated.

Events may be generated in two ways:

- automatically following an action by the user.

- via programming.

In this latter case, the event is generated while the current event is being processed. This generation may be explicit (SEND) or implicit (SELECT for example). In both cases, processing is synchronous and therefore carried out immediately.

During debugging in Step mode, the script displayed reflects this synchronous succession of events.

If this succession is evident where explicit generation is involved, it may be less so in the case of implicit generation.

This occurs each time you use an NCL verb that equates to an operator action such as SELECT in a List Box (SELECTED event), MOVE in an Entry Field (CHANGED event), etc … If the processing associated with an event does not provide for the occurrence of such events, erroneous behaviour may be the result. In this case, debugging in Step mode lets you find the cause of the malfunction.

If debugging an application enables you to fully understand its running, it may also induce different behaviour for certain events.

When NS-Debug interrupts the application on the occurrence of an event, any other event received by the application is lost while NS-Debug is in the “Ready” state. Normal receipt of events is not resumed until the debugger returns to the “Exec” state.

Several events are sensitive to this behaviour of NS-Debug:

- TIMER: All TIMER events are lost when the debugger is in the "Ready" state. If each occurrence of TIMER is important for the application, it may then be necessary to inhibit start-up of the timer and generate it by SEND TIMER (via the Instant window).

- CHARACTER: Stopping at a CHARACTER event should be prohibited if you are testing for a key to be pressed and released. In effect, pressing the key will be processed, but the CHARACTER event indicating the release will be lost. The only way it for it to be recognized is to generate it (via the Instant window) at the end of processing the current CHARACTER event and by passing the expected KF_*% parameters.

- GETFOCUS/LOSEFOCUS: Passing the focus from one control to another is always sensitive. A GETFOCUS or a LOSEFOCUS may or may not be lost during interruption of the application, something which may freeze the application when the focus could not be given.

- PAINT: The order and number of PAINT events received by the current debugging window may be different because the window may be hidden by NS-Debug during an interruption.

- MOUSEMOVE: If a breakpoint is set in the script of this event, it is enabled each time the mouse pointer is moved from the NS-Debug window to the application window. A method of avoiding this looping is to use keyboard shortcuts when the NS-Debug window obtains the focus.

- BUTTONDOWN/BUTTONUP/BUTTONDBCLK: An interruption in the script of an event associated with pressing a mouse button causes the associated event to be lost when the button is released. You should therefore simulate the release of the button if its processing needs to be recognized for proper continuity of the application.

Chapter 6

Example of the use of NS-Debug

You will find in this chapter

- An example of using NS-Debug with the CALC calculator

Contents

Introduction

The aim of this chapter is to demonstrate the use of NS-Debug's various functionalities in a specific manner and by means of a number of illustrations. In order to do this, we will be relying on the example of the CALC calculator, one of the applications provided with NS-DK.

We will not be dealing with looking for bugs but a basic study of how the calculator's memory is managed. Using the debugger in this case is similar to the approach of a programmer seeking to understand the cause of an error in a section of code.

NS-DK resources associated with this example are:

CALC.SCR, LCALC.NCL, VCALC.VAR

We will suppose that this application is part of the project EXEMPLE.XNP stored in the C:\PROJECTS directory of the development machine and that no main window has been defined for this project.

NS-Debug is launched from the command line of the *Run* dialogue box.

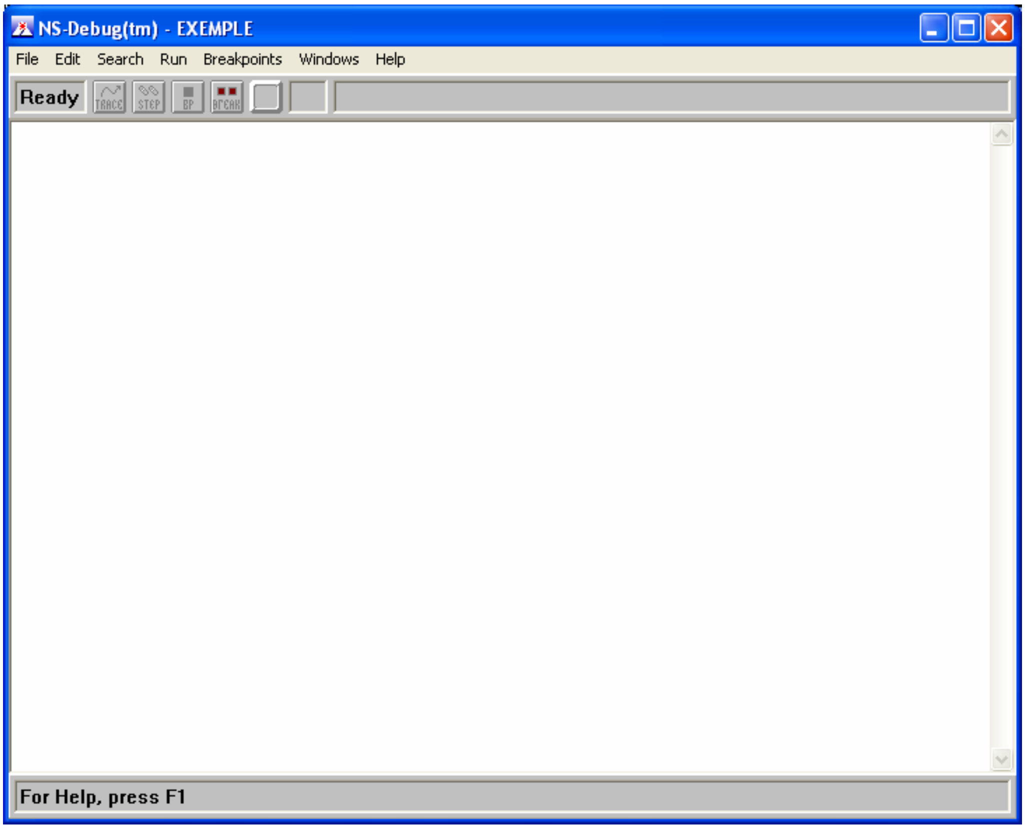
Launching the program is carried out by typing:

```
   NSDEBUG  /PRJ:C:\PROJECTS\EXEMPLE.XNP
```

CALC calculator

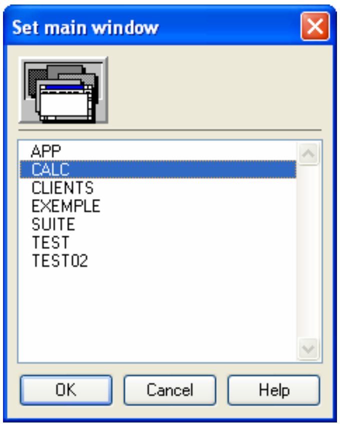
Choosing the start-up window

The NS-Debug main window is displayed.



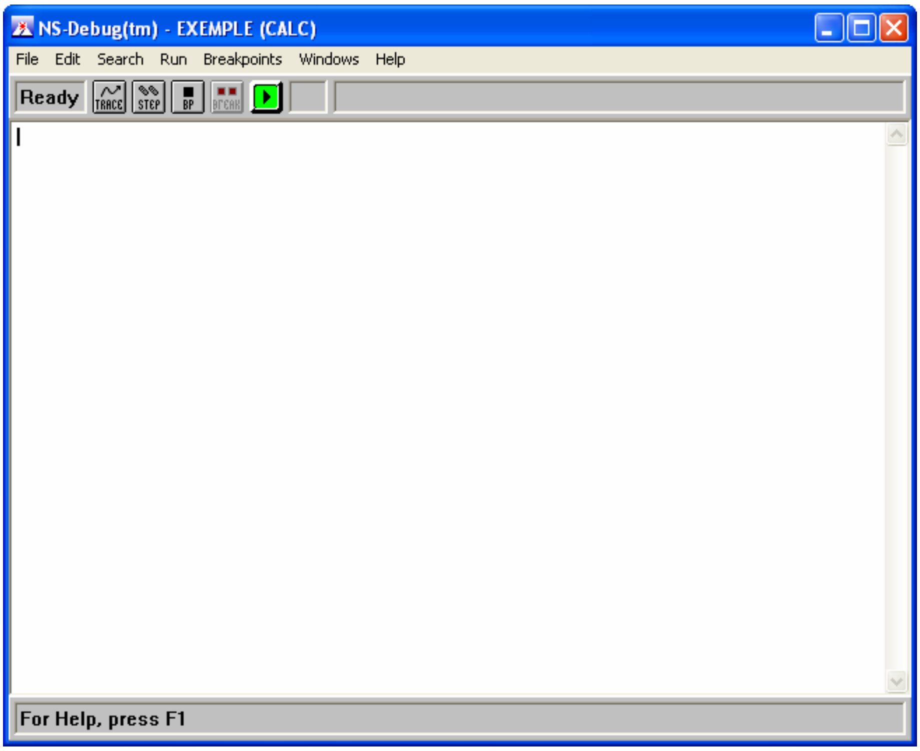
All the icons and menu options enabling you to launch a debugging session are inhibited because no main window was defined in the project or via a launch parameter.

We will choose the start window using the *Run/Restart* menu (or [Shift]+[F5]).



We will then select *CALC* and click on *OK*.

The name of the window is now shown in brackets to the right of the project name and the icons are enabled.

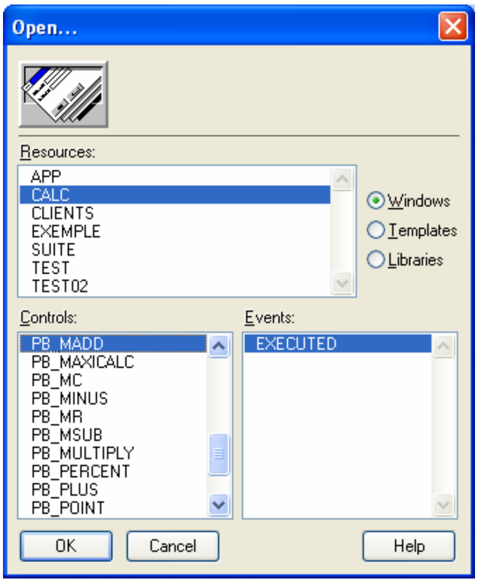


Setting breakpoints

In order to be able to launch the debugger in Test mode, we are going to set some breakpoints in the lines of code where the processing that interests us is carried out.

Initially, we will open the script of the *EXECUTED* event for the *MADD* button corresponding to pressing *M+*.

File/Open (or [Shift]+[F4]) lets us choose Windows resources then select *PB_MADD* and the event *EXECUTED*.



Open...

Resources:

- APP
- CALC
- CLIENTS
- EXEMPLE
- SUITE
- TEST
- TEST02

● Windows
○ Templates
○ Libraries

Controls:

- PB_MADD
- PB_MAXICALC
- PB_MC
- PB_MINUS
- PB_MR
- PB_MSUB
- PB_MULTIPLY
- PB_PERCENT
- PB_PLUS
- PB_POINT

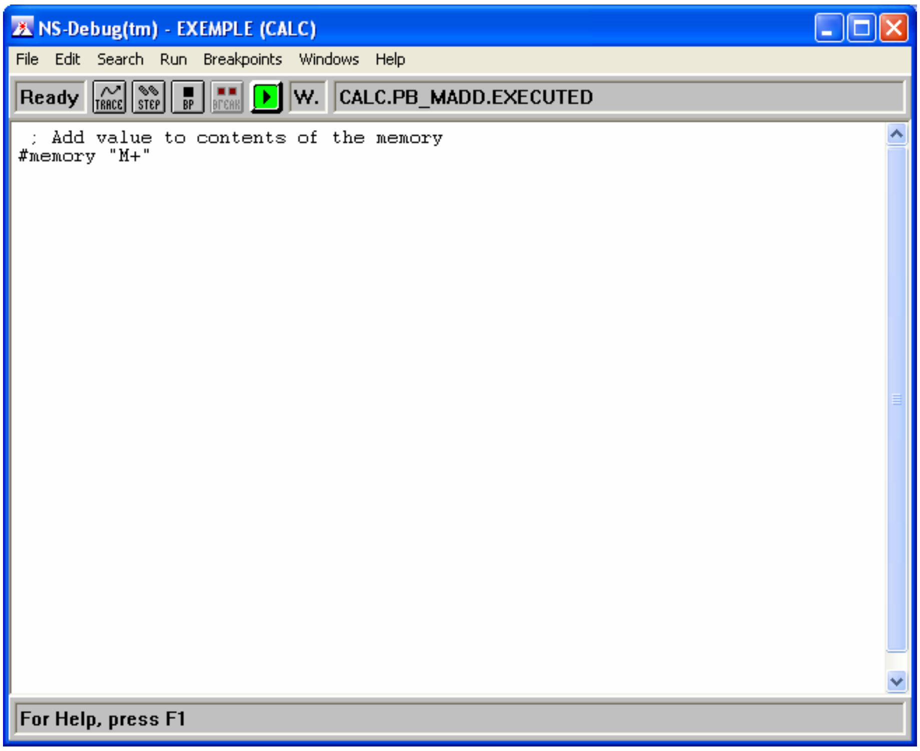
Events:

- EXECUTED

We will set a breakpoint in the line of code figuring in this script.

In order to do this, we will position the cursor on this line then click on the BP icon or press [F7].

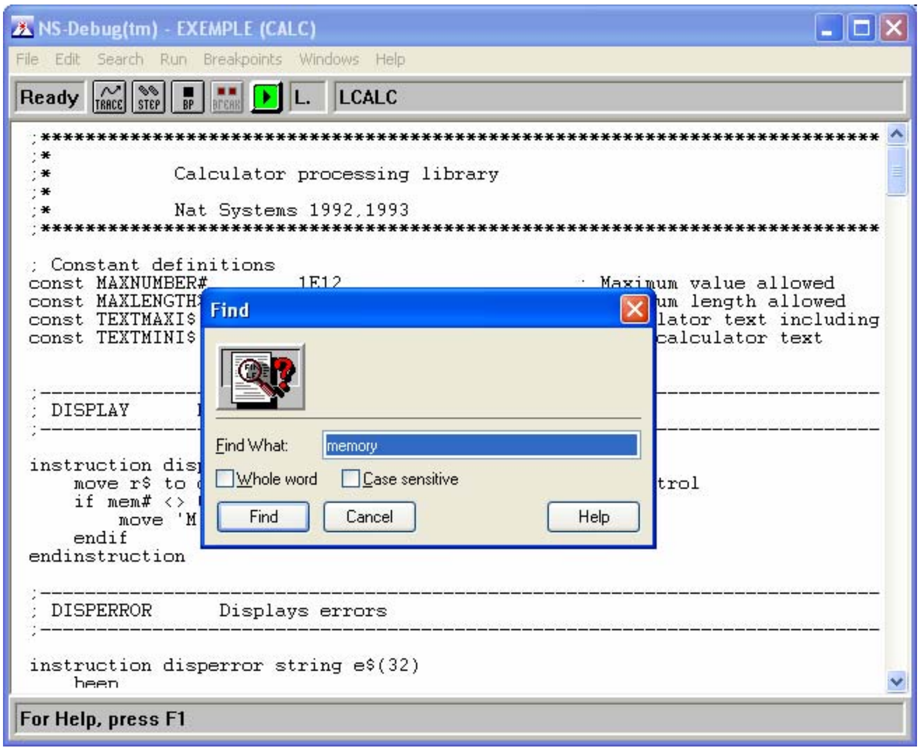
The breakpoint is then shown under Windows by a number sign to the left of the line.


NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Ready
TRACE
STEP
BP
BREAK
W.
CALC.PB_MADD.EXECUTED
For Help, press F1

```
  ; Add value to contents of the memory
#memory "M+"
```

Still using *File/Open*, we will now open the *LCALC* library where we know the *memory* instruction is located.

Once the script of *LCALC* is displayed, we will find the line where the *memory* instruction is coded using *Search/Find*.

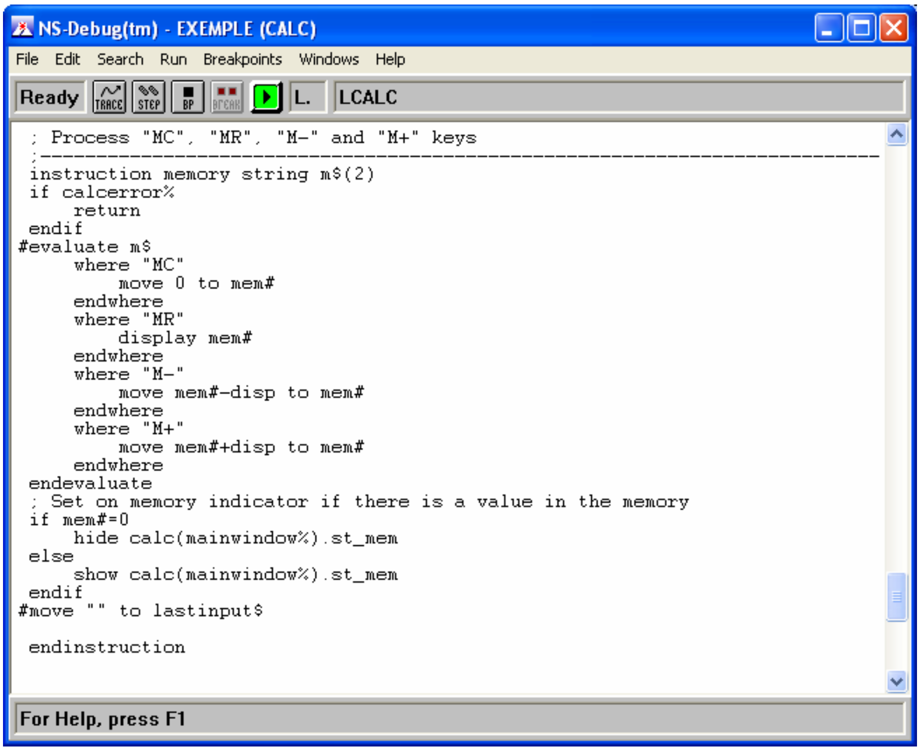

NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Ready
TRACE
STEP
BP
BREAK
L.
LCALC
Find
Find
Cancel
Help
For Help, press F1

```
;*****************************************************************************
;*
;*           Calculator processing library
;*
;*           Nat Systems 1992,1993
;*****************************************************************************

; Constant definitions
const MAXNUMBER#         1E12                        ; Maximum value allowed
const MAXLENGTH?                                         um length allowed
const TEXTMAXI$                                          lator text including
const TEXTMINI$                                          calculator text


;----------------                                        ---------------------
; DISPLAY       I
;----------------                                        ---------------------

instruction disp
    move r$ to c                                         trol
    if mem# <> (
        move 'M
    endif
endinstruction

;-----------------------------------------------------------------------------
; DISPERROR      Displays errors
;-----------------------------------------------------------------------------

instruction disperror string e$(32)
    heen
```

Find What: memory

☐ Whole word ☐ Case sensitive

As this instruction is now displayed, we will set some breakpoints: on evaluation of the memory key and before output of the instruction.

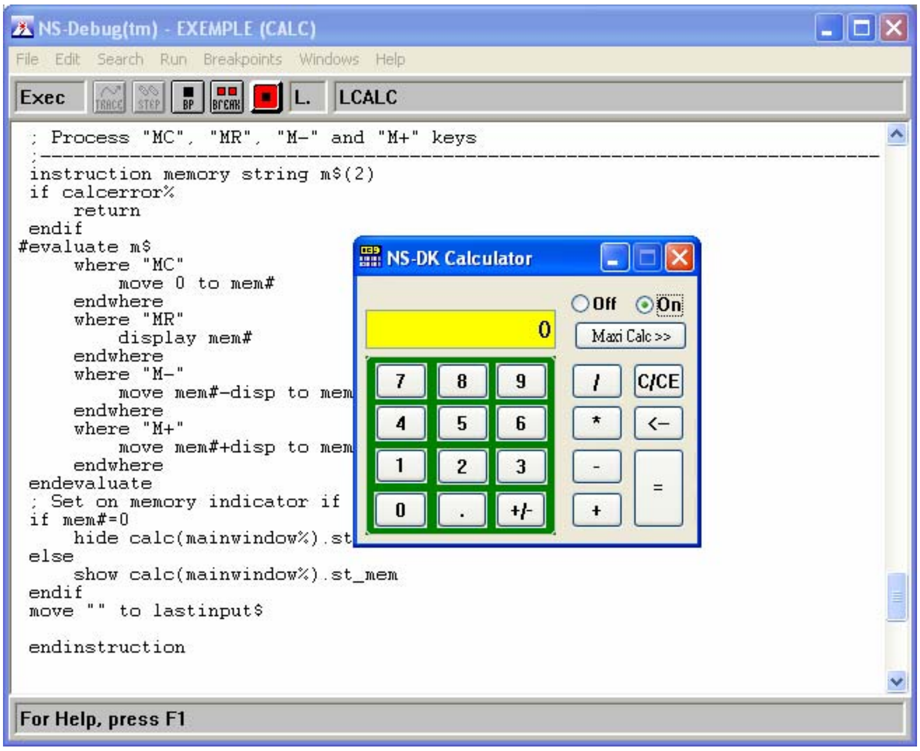


```
 ; Process "MC", "MR", "M-" and "M+" keys
 ;----------------------------------------------------------------------------
 instruction memory string m$(2)
 if calcerror%
     return
 endif
#evaluate m$
     where "MC"
         move 0 to mem#
     endwhere
     where "MR"
         display mem#
     endwhere
     where "M-"
         move mem#-disp to mem#
     endwhere
     where "M+"
         move mem#+disp to mem#
     endwhere
 endevaluate
 ; Set on memory indicator if there is a value in the memory
 if mem#=0
     hide calc(mainwindow%).st_mem
 else
     show calc(mainwindow%).st_mem
 endif
#move "" to lastinput$

 endinstruction
```

Debugging in Test mode

We will now start our debugging session in Test mode.

This is done by pressing the *Run* button or [F5]. The *CALC* window now appears in the foreground of the screen and the debugger changes from the “*Ready*” state to the “*Exec*” state.

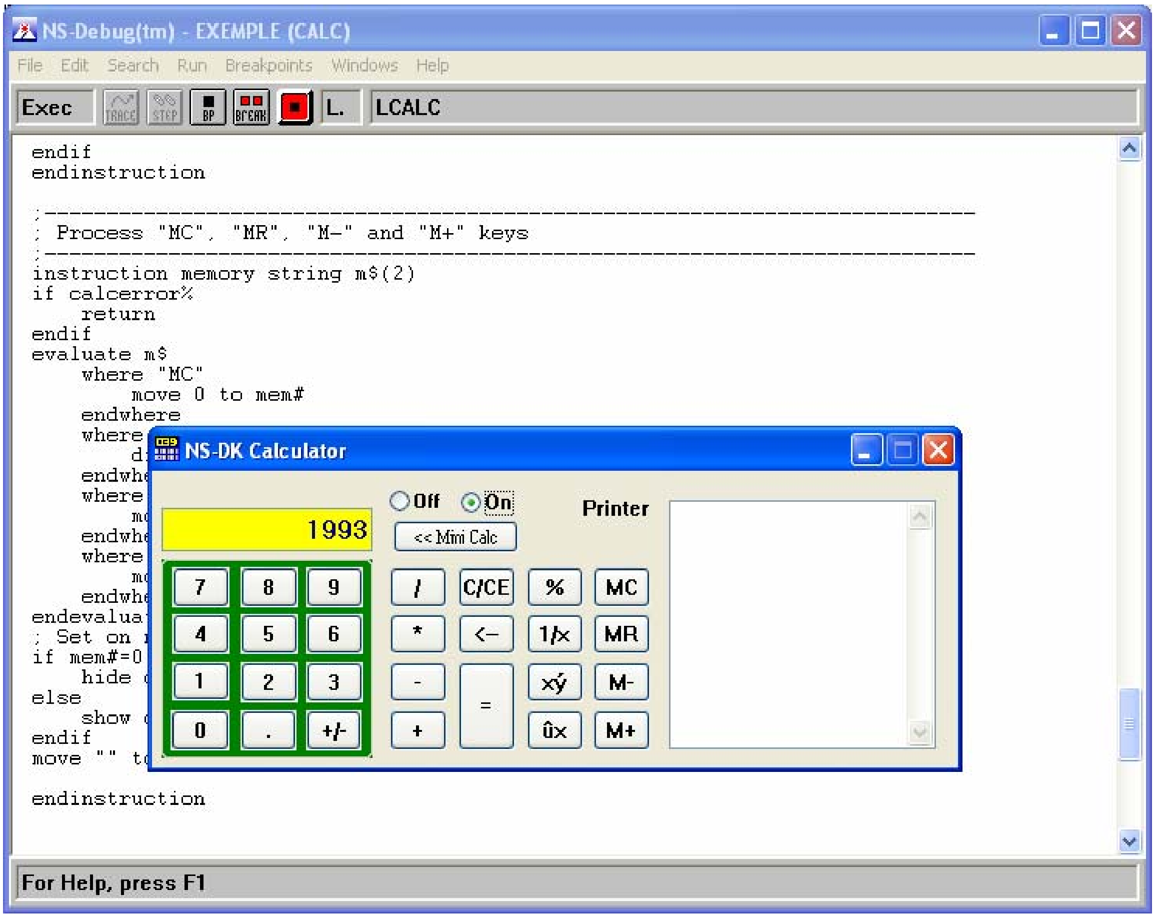


```
 ; Process "MC", "MR", "M-" and "M+" keys
 ;-------------------------------------------------------------------------------
 instruction memory string m$(2)
 if calcerror%
     return
 endif
#evaluate m$
     where "MC"
         move 0 to mem#
     endwhere
     where "MR"
         display mem#
     endwhere
     where "M-"
         move mem#-disp to mem
     endwhere
     where "M+"
         move mem#+disp to mem
     endwhere
 endevaluate
 ; Set on memory indicator if
 if mem#=0
     hide calc(mainwindow%).st
 else
     show calc(mainwindow%).st_mem
 endif
 move "" to lastinput$

 endinstruction
```

Obtaining the memory functions is done by enlarging the format of the calculator. This is done by pressing the *MaxiCalc>>* button.

We will note what happens when the number 1993 is loaded into memory.

First we will enter 1993.



```
endif
endinstruction

;---------------------------------------------------------------------------
; Process "MC", "MR", "M-" and "M+" keys
;---------------------------------------------------------------------------
instruction memory string m$(2)
if calcerror%
    return
endif
evaluate m$
    where "MC"
        move 0 to mem#
    endwhere
    where
        d
    endwhe
    where
        mo
    endwhe
    where
        mo
    endwhe
endevalua
; Set on m
if mem#=0
    hide c
else
    show c
endif
move "" to

endinstruction
```

1993

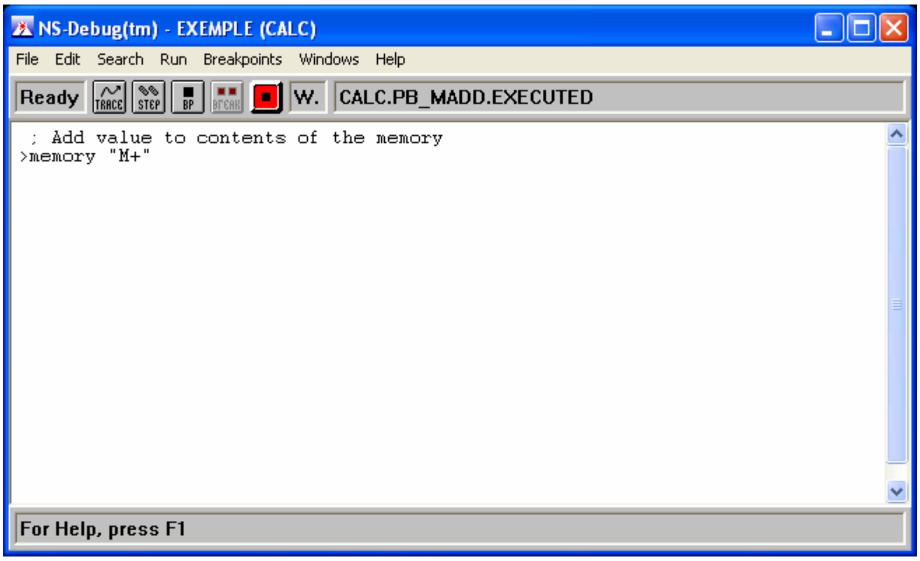
Then press the *M+* key.

Up to now, execution has proceeded in continuous fashion without any intervention from the debugger. Pressing *M+* causes an interruption to execution of processing of the line of code where the first breakpoint was set.

The debugger main window then changes to the foreground and its status changes to “Ready”.

In the script, the line at which the interruption took place is shown by a greater than sign under Windows.

The code of this line has not yet been executed.


NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Ready
TRACE
STEP
BP
BREAK
W.
CALC.PB_MADD.EXECUTED
For Help, press F1

```
 ; Add value to contents of the memory
>memory "M+"
```

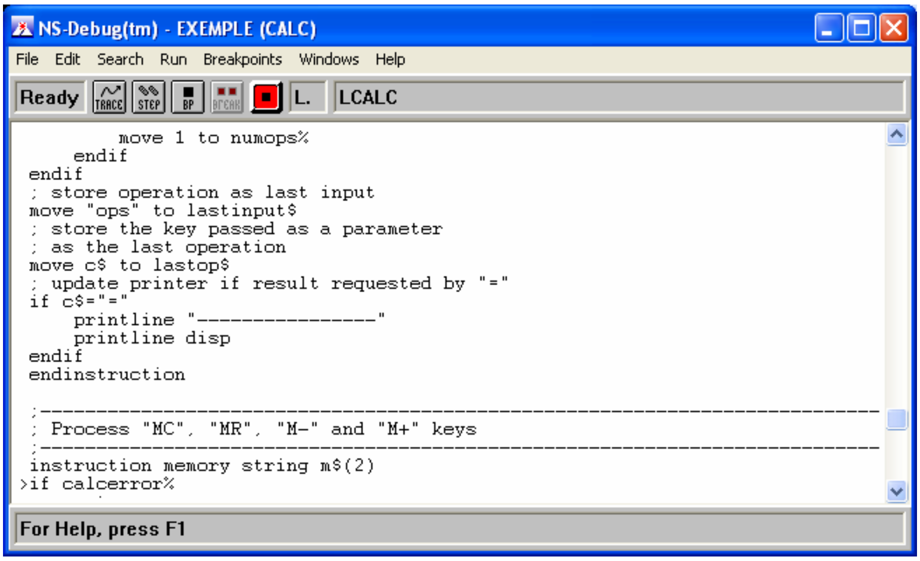
Trace Mode

The current line of code contains the call to an instruction in the *LCALC* library.

In order to be able to display the script of this instruction, we will choose to continue in Trace mode.

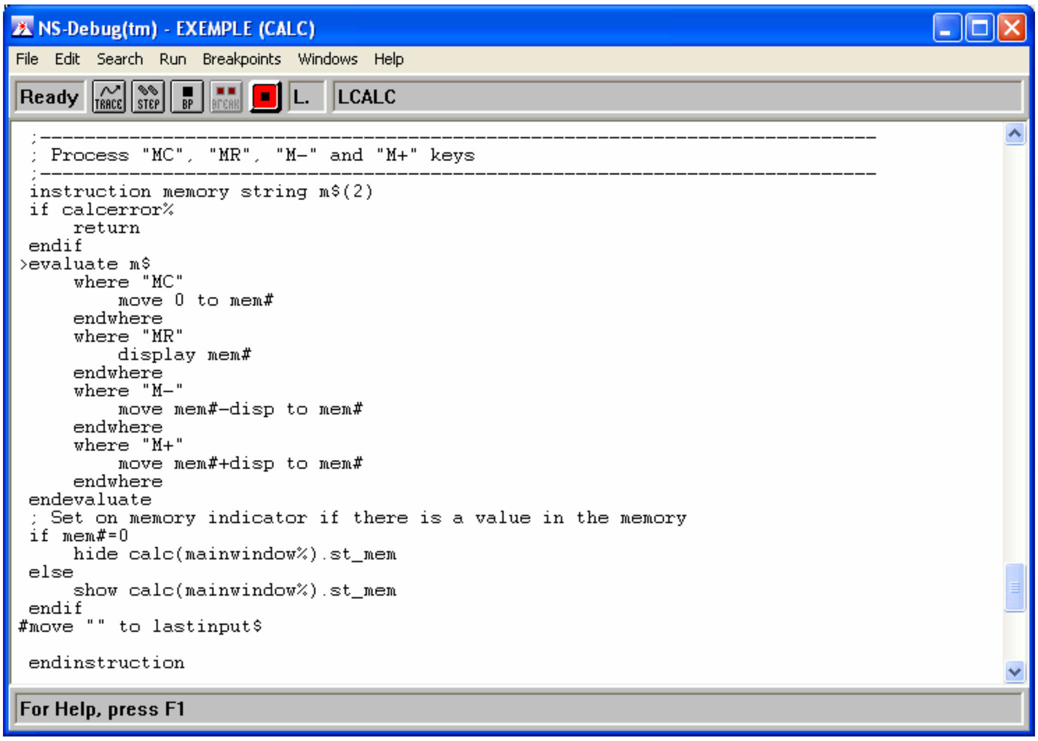
Clicking on the *Trace* icon or by pressing [F8] causes the *memory* instruction to be executed and an immediate interruption at the first line of its code.

The script of the instruction is then displayed in the editing area.


NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Ready TRACE STEP BP BREAK L. LCALC
For Help, press F1

```
        move 1 to numops%
    endif
endif
; store operation as last input
move "ops" to lastinput$
; store the key passed as a parameter
; as the last operation
move c$ to lastop$
; update printer if result requested by "="
if c$="="
    println "----------------"
    println disp
endif
endinstruction

;---------------------------------------------------------------------------
; Process "MC", "MR", "M-" and "M+" keys
;---------------------------------------------------------------------------
instruction memory string m$(2)
>if calcerror%
```

We will relaunch the debugger in Test mode via *Run/Go* or by pressing [F5] in order to be interrupted at the next breakpoint located in the body of the *memory* instruction.

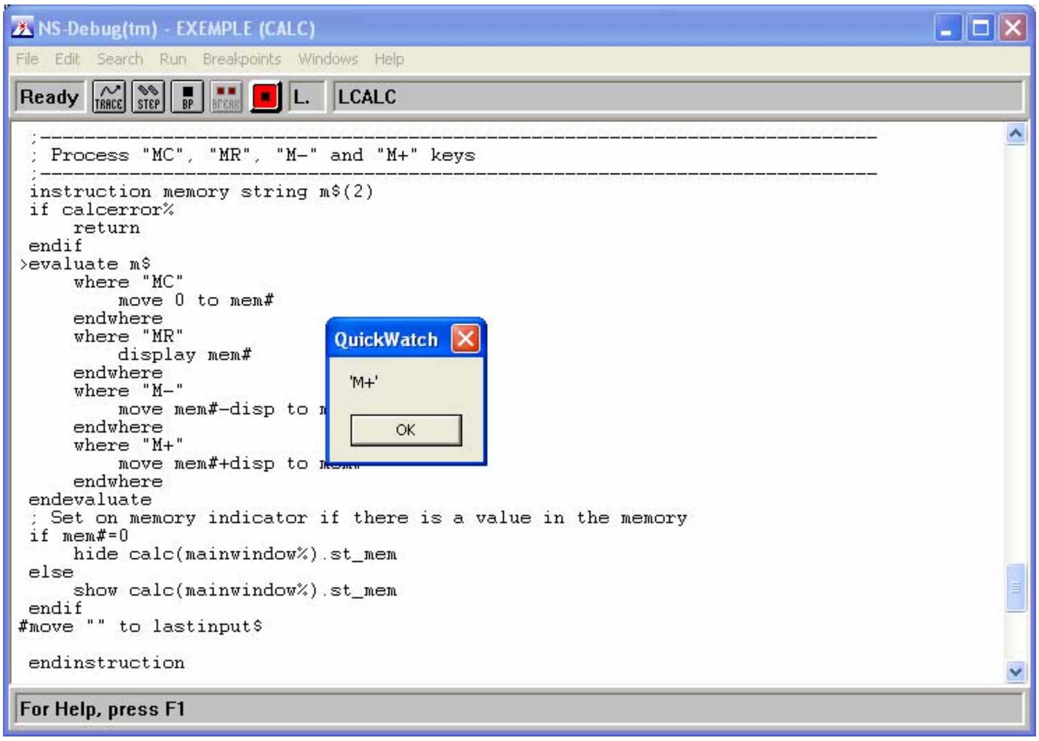

NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Ready TRACE STEP BP BREAK L. LCALC
For Help, press F1

```
 ;---------------------------------------------------------------------------
 ; Process "MC", "MR", "M-" and "M+" keys
 ;---------------------------------------------------------------------------
 instruction memory string m$(2)
 if calcerror%
     return
 endif
>evaluate m$
     where "MC"
         move 0 to mem#
     endwhere
     where "MR"
         display mem#
     endwhere
     where "M-"
         move mem#-disp to mem#
     endwhere
     where "M+"
         move mem#+disp to mem#
     endwhere
 endevaluate
 ; Set on memory indicator if there is a value in the memory
 if mem#=0
     hide calc(mainwindow%).st_mem
 else
     show calc(mainwindow%).st_mem
 endif
#move "" to lastinput$

 endinstruction
```

Action relating to the data and Step mode

We will display the content of the variable *m$* using Quick Watch.

Initially, we will select *m$*.


NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Ready
TRACE
STEP
BP
BREAK
L.
LCALC
QuickWatch
'M+'
OK
For Help, press F1

```
 ;---------------------------------------------------------------------------
 ; Process "MC", "MR", "M-" and "M+" keys
 ;---------------------------------------------------------------------------
 instruction memory string m$(2)
 if calcerror%
     return
 endif
>evaluate m$
     where "MC"
         move 0 to mem#
     endwhere
     where "MR"
         display mem#
     endwhere
     where "M-"
         move mem#-disp to m
     endwhere
     where "M+"
         move mem#+disp to m
     endwhere
 endevaluate
 ; Set on memory indicator if there is a value in the memory
 if mem#=0
     hide calc(mainwindow%).st_mem
 else
     show calc(mainwindow%).st_mem
 endif
#move "" to lastinput$

 endinstruction
```

Then we will press [Shift] + [F1].

The *Quick Watch* message box gives us *M+* as a result.

Processing will therefore continue via execution of the line of code *where "M+"*.

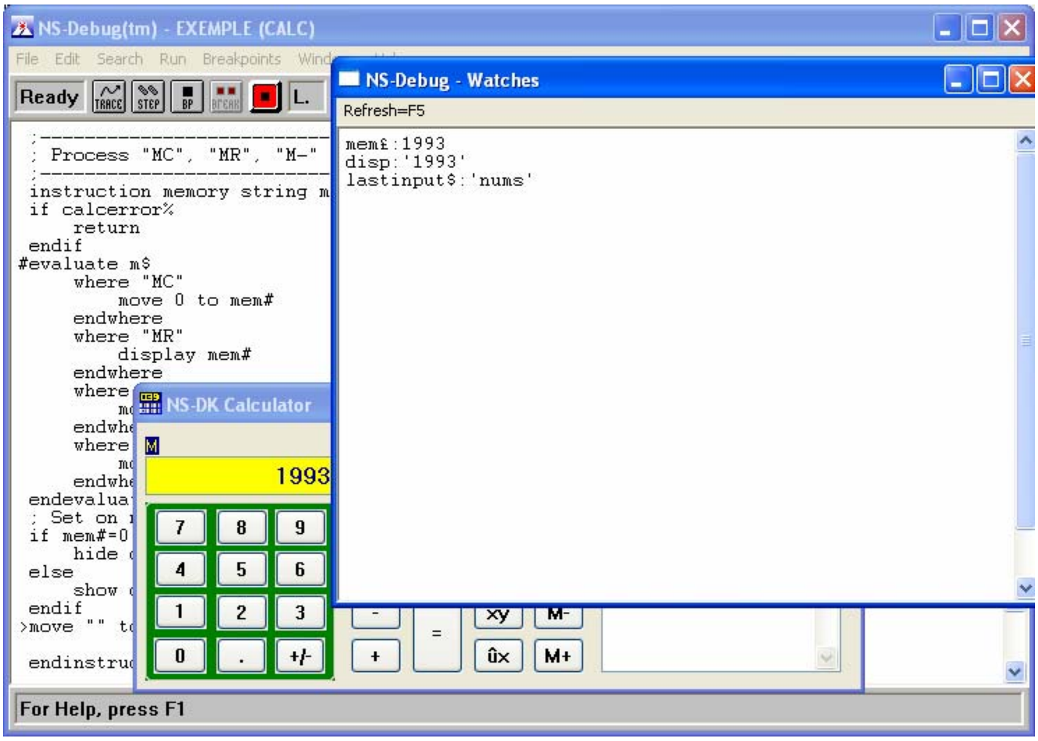
Knowing by deduction the next lines of code that will be executed, we can now position the cursor to the line of code *move mem£ + disp to mem£* and choose the option *Go to cursor* from the *Run* menu or press [Ctrl] + [F5].

All the lines of code are then executed and a new interruption takes place at the line of code at which we are positioned.

We will now open the *Watches* window (menu *Windows/Watches* or by pressing [Ctrl] + [F4]) in order to find out the value of certain global variables as the code is executed.

We will reposition and resize the *Watches* window so that it does not overlap the main NS-Debug window.

Then we will enter the variables *mem£*, *disp* and *lastinput$* in the editing area (by typing them direct or by copying and pasting from the script).



```
 ;---------------------------
 ; Process "MC", "MR", "M-"
 ;---------------------------
 instruction memory string m
 if calcerror%
     return
 endif
#evaluate m$
     where "MC"
         move 0 to mem#
     endwhere
     where "MR"
         display mem#
     endwhere
     where
         mo
     endwhe
     where
         mo
     endwhe
 endevalua
 ; Set on 
 if mem#=0
     hide 
 else
     show 
 endif
>move "" to

 endinstruc
```

```
mem£:1993
disp:'1993'
lastinput$:'nums'
```

From the current line of the script we will continue in Step mode by clicking on the *Step* icon or by pressing [Shift] + [F8].

A line of code is then executed.

The debugger changes from the “Ready” state to the “Exec” state, the *CALC* window obtains the focus then the debugger returns to the “Ready” state and again obtains the focus.

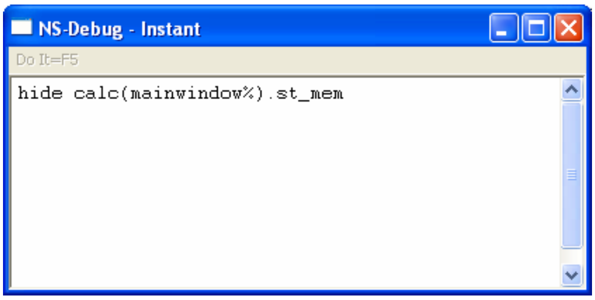
The content of each variable displayed in the *Watches* window is also updated.

Each line of code is thus executed Step by Step with as many clicks on *Step* as there are lines of code to be executed before leaving the *memory* instruction. In this way we will be able to find out changes to the variables as the script is executed.

On arriving at the last line of code of the instruction (*move "" to lastinput$*), we will check that the line of code, the execution of which depended on another value of *mem£*, also works. This equates, therefore, to explicitly hiding the memory storage indicator, the appearance of which was caused by the previous line of code.

To do this we will use the *Instant* window via the *Windows/Instant* menu or by pressing [F4].

After repositioning and resizing it, we will enter the NCL code *hide calc(mainwindow%).st_mem* in the editing area of the *Instant* window.



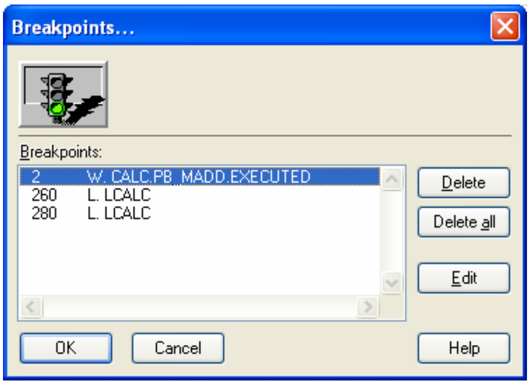
```
hide calc(mainwindow%).st_mem
```

We will launch execution by selecting the *Do It* menu or by pressing [F5].

By giving the focus to the *CALC* window, we can check that the code executes correctly.

We will finish this example by deleting all the breakpoints.

We will edit these breakpoints by selecting the *Breakpoints/Edit breakpoint* menu or by pressing [Shift]+[F7].



Breakpoints:

```
2      W. CALC.PB_MADD.EXECUTED
260    L. LCALC
280    L. LCALC
```

Once the *Breakpoints* dialogue box displays, press *Delete all* then *OK*.

All the breakpoints are now deleted, and we can relaunch the debugger in Test mode by pressing [F5].

The *CALC* window gets the focus and waits for an action on our part.

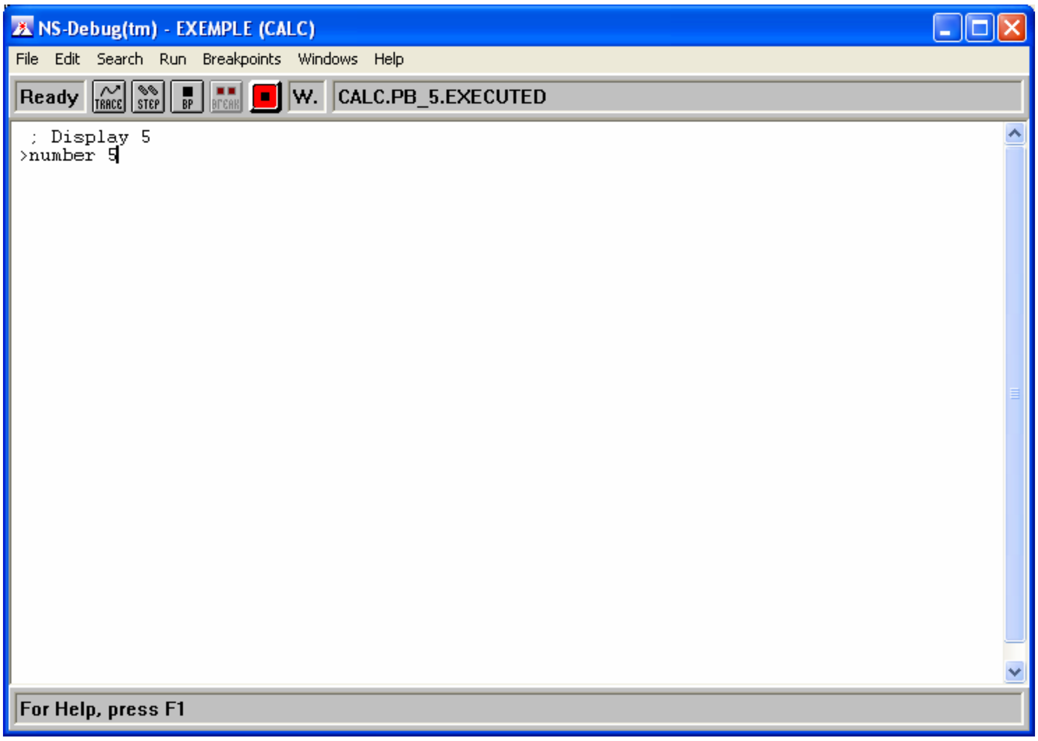
As we will not explicitly request an interruption, execution takes place exactly as under NS-Test.

We will suppose that we wish to interrupt execution at the next action.

We will give the focus back to the main NS-Debug window and select *Run/Break* or click on the *Break* icon BrEAK.

We will return to the *CALC* window and press the 5 key.

Execution is immediately interrupted and the script of the *EXECUTED* event of the 5 button is displayed in the editing area.


NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Ready
TRACE
STEP
BP
BrEAK
W.
For Help, press F1

CALC.PB_5.EXECUTED

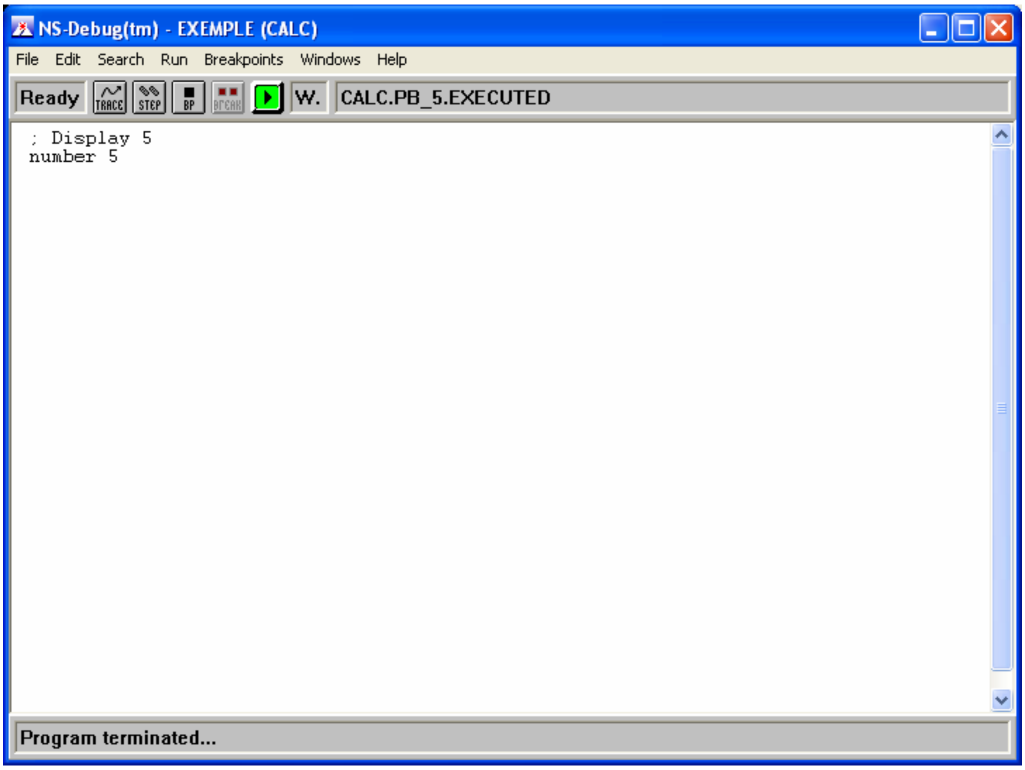
```
 ; Display 5
>number 5
```

We will then continue debugging in Trace, Step or Test mode.

Stopping debugging

Whatever the current mode and status of the debugger, we will finish the debugging session using the *Run/Stop* menu or by pressing the *Run* button.

The message *Program terminated* is then displayed in the help line and the *Run* button changes to green.


NS-Debug(tm) - EXEMPLE (CALC)
File Edit Search Run Breakpoints Windows Help
Ready
TRACE
STEP
BP
BREAK
W.

CALC.PB_5.EXECUTED

```
; Display 5
number 5
```

Program terminated...

We can then either relaunch a new session or quit NS-Debug via *File/Exit* or by pressing [F3].

Appendix A Error messages

This section presents the error messages listed.

The cause of the error and the action that needs to be taken to correct it are given for each of them.

You will find in this chapter

- The various error messages displayed by NS-Debug.

Usage NSDEBUG [/PRJ:<ProjectName>] [MainName]

Cause: Launch parameters incorrect.

Solution: Enter the parameters keeping to the syntax described in the error message and given in chapter 2 of this document.

Memory error ! Not enough memory

Cause: NS-Debug is not able to allocate a sufficiently large memory area in order to operate. This message also appears after the message: *Warning ! Cannot allocate memory for debugging session. Stop execution ?* and the associated *Cancel* button has been pressed.

Solution: Stop one or more applications running on your machine so that the memory they are using can be freed.

Configuration file not found

Cause: The project configuration file (.xnp) specified at launch has not been found.

Solution: Check the path of the XNP file given as a parameter at launch. If it is correct, check if the XNP file shown still exists.
If no project was given as a parameter, this means that no default project has been found. In this case, check the value of the NS-INI environment variable, which must exist and contain the path of the default project.

No project name specified

Cause: No project name was given as a launch parameter and no default project exists.

Solution: Check the value of the NS-INI environment variable which must exist and contain the path for the default project.

Error!: Window does not belong to the project (xxx)

Cause: The window name xxx specified at the launch of NS-Debug does not belong to the project.

Solution: The name of the window was not specified correctly or is not a project resource. Use Run/Restart in order to select a name from the project's Windows resources.

Error!: Cannot find resource (xxx).

Cause: The resource name xxx does not correspond to a file on the disk.

Solution: The resource indicated has only been created in the project but no other action has been taken to associate a file with it: File/Open for a window, Edit for a library. Stop NS-Debug and check the state of progress of the project to be debugged to find out which resources are associated with a file.

Cannot initialize HELP manager properly

Cause: The NS-Design help file, in which NS-Debug components are described, could not be found.

Solution: Press Cancel if you want to continue without using on-line help. Otherwise stop NS-Debug and in NS-Design (Options/Setup) specify the correct path to the NSDEBUG.HLP file, as this is used by NS-Debug.

Cannot evaluate expression

Cause: An expression for which you have requested the result could not be evaluated because its components were not yet initialized.

Solution: Only ask for the contents of an expression to be displayed when all the components are significant. By expression we also mean CONTROL type variables.
This sort of error may cause the system to stop the application being debugged. In this case, this error message only appears after the system error message.

Too many breakpoints

Cause: The number of breakpoints exceeds 32, the maximum permitted.

Solution: Delete a minimum of one breakpoint from those already set before setting a new one.

String xxxx not found

Cause: A string being looked for by Find/Search could not be found in the current script.

Solution: Open another script if the search concerns a string you know belongs to one of the application’s scripts.

Value must be between x and y

Cause: The line number specified in the Goto Line dialogue box is not correct.

Solution: The x and y values indicate respectively the numbers of the first and last line of the current script. You should enter a value between these limits.