

Natural

Debugger

Version 6.3.13 for Windows

October 2012

Natural

This document applies to Natural Version 6.3.13 for Windows.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1992-2012 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, United States of America, and/or their licensors.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://documentation.softwareag.com/legal/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

Document ID: NATWIN-NNATDEBUG-6313-20121005

Table of Contents

Preface	v
1 General Information	1
About the Debugger	2
Remote Debugging	3
2 Starting and Leaving the Debugger	9
Preparing to Use the Debugger	10
Starting the Debugger	10
Restarting the Debugger	11
Leaving the Debugger	12
3 Elements of the Debugger	13
Debugger Information in the Title Bar	14
Menu Commands	14
Toolbar	14
Trace Position in Editor Window	15
Debugger Windows	16
4 Moving through the Code	21
Stepping Through the Code	22
Going to the Next Breakpoint or Watchpoint	23
Going to the Next Event	24
Going to the Cursor Position	24
Going to the Next Statement	25
5 Setting Breakpoints and Watchpoints	27
About Breakpoints and Watchpoints	28
Adding and Removing a Breakpoint	29
Modifying a Breakpoint	30
Adding a Watchpoint	31
Modifying a Watchpoint	33
Deactivating Breakpoints and Watchpoints Temporarily	34
Showing the Source Code for a Defined Breakpoint or Watchpoint	35
Deleting Breakpoints and Watchpoints	35
Symbols Used in the Editor Window	36
6 Modifying and Watching Variables	37
Modifying a Variable	38
Adding a Watchvariable	41
Managing the Variables in the Variables Window	42
7 Using the Call Stack	47
About the Call Stack	48
Displaying the Source Code of a Different Object	48
Returning to the Object at the Current Trace Position	49
8 Using the Old Debugger	51
Preparing Natural Objects	52
Starting the Debugger	52
Leaving the Debugger	53

Operating the Debugger	54
Debugger Source Window	56
Watchvariables Control Bar	63
Variables Control Bar	63
Watchpoints and Breakpoints Control Bar	64

Preface

This documentation, which is complementary to the *Using Natural Studio* documentation, explains how to debug Natural applications. It is organized under the following headings:

General Information	About the debugger which is integrated in Natural Studio. Information on remote debugging and how to set up your environment for remote debugging.
Starting and Leaving the Debugger	Information on the SYMGEN parameter. How to start, restart and leave the debugger.
Elements of the Debugger	Information on additional elements which are available in the Natural Studio window when the debugger has been started.
Moving through the Code	How to execute the code by stepping through it or by going to breakpoints, watchpoints, events or to the cursor position.
Setting Breakpoints and Watchpoints	How to add breakpoints and watchpoints, and how to manage them in the break- and watchpoints window.
Modifying and Watching Variables	How to modify a variable, how to add watchvariables, and how to manage the variables in the variables window.
Using the Call Stack	How to manage the objects in the call stack window.

In addition to the above topics which describe the new debugger which is integrated in Natural Studio, the documentation for the old debugger is also provided in the section *Using the Old Debugger*. The old debugger may appear when an old version of Natural is installed on the development server; see also *General Information*.

1 General Information

- About the Debugger 2
- Remote Debugging 3

About the Debugger

Starting with Natural for Windows Version 6.2, the debugger is integrated in Natural Studio. The complete Natural Studio functionality can thus be used in parallel to the debugger. For example, when the debugger is active, you can navigate to another object in the library workspace or you can search for a specific object using the **Find Object** command.

The debugger is used to debug Natural applications in the following environments:

- in the local environment, and
- in a remote environment, that is: on a mapped development server (SPoD). The prerequisite is that one of the following versions is installed on the development server:
 - Natural for Mainframes Version 4.2 or above.
 - Natural for UNIX Version 6.2 or above.

No additional settings are necessary for debugging. Natural Studio handles all steps internally (such as setting up or terminating the communication with the corresponding server).

See also *Environments and Views in the Library Workspace* in the documentation *Using Natural Studio* and *Accessing a Remote Development Environment* in the documentation *Remote Development Using SPoD*.



Note: Several differences exist when you debug applications in a remote mainframe environment. These differences are listed in the platform-specific Natural Development Server (NDV) documentation which applies to this Natural release. The NDV documentation is available separately; it is not part of this Natural for Windows documentation.

When is the Old Debugger Still Used with SPoD?

When you are working with Natural Studio and invoke the debugger for an object on a mapped development server, it is possible that the old debugger is invoked instead of the new integrated debugger. This is the case when an old version of Natural is installed on the development server. Old versions are:

- Natural for UNIX Version 6.1.1 or below.

See [Using the Old Debugger](#).


Remote Debugging

With one of the next versions, remote debugging will no longer be supported. Instead, you will have to use the debugger which is integrated in Natural Studio.

Remote debugging is done when you debug a native Natural for UNIX application from a Windows computer, or when you debug a Natural dialog application remotely from another PC. This is done outside the context of SPoD.

To enable remote debugging, you have to proceed as follows:

- Install the debug front-end on a Windows computer. This also installs the remote debugging service `natdbgsv` which must be active for remote debugging. See [Installing the Remote Debugger](#).
- Define the parameters `RDNODE`, `RDPORT` and `RDACTIVE` in the environment which contains the application that is to be debugged. See [Setting Up Your Environment for Remote Debugging](#) for further information.
- Invoke the debugger by entering the system command `DEBUG object-name` in the environment which contains the application that is to be debugged.
 - [Installing the Remote Debugger](#)
 - [Setting Up Your Environment for Remote Debugging](#)
 - [Scenarios for Remote Debugging](#)

 **Important:** For running the remote debugger, the Microsoft Windows Personal Firewall must be deactivated. See [Configuring the Microsoft Windows Personal Firewall to Run Natural](#) in the *Operations* documentation for Natural for Windows.

Installing the Remote Debugger

If you have Natural for Windows installed, you must use the remote debugger delivered with Natural for Windows. If the remote debugger has not yet been installed, use the **Modify** option of the Natural installation package to add the remote debugger to your Natural for Windows installation. See [Maintaining Your Natural or Natural Runtime Environment](#) in the *Installation* documentation for Natural for Windows.

You only need to install the remote debugger stand-alone, if you do not have Natural for Windows installed. If you want to debug a Natural application which is stored on a UNIX platform, copy `$NATDIR/$NATVERS/dbrmt/I386/nrd.exe` from the UNIX installation medium to your Windows computer (for example, to a temporary directory) and unzip it. Run `setup.exe` to start the installation of the remote debugger.

Setting Up Your Environment for Remote Debugging

The following topics are covered below:

- [Windows Side without Terminal Services](#)
- [Windows Side with Terminal Services](#)
- [Natural Side](#)

Windows Side without Terminal Services

Either install the remote debugger (the corresponding files can be found on the UNIX installation medium) or install Natural for Windows (the remote debugger can optionally be installed with a Natural for Windows custom installation; see the *Installation* documentation for Natural for Windows). This also installs the Natural remote debugging service `natdbgsv`.

To uninstall the remote debugging service, enter `natdbgsv -u` in the command line. To view the current service's port name and version, enter `natdbgsv -s`. To re-install the service on a different port, uninstall it first and then enter `natdbgsv -i portnumber`, where *portnumber* is the value of the `RDPORT` profile parameter. If the port number is already used, a dialog appears where you can enter a new port number.



Note: Before you install the remote debugging service on a port other than 2600 (default value), you have to change the value of the `RDPORT` profile parameter to match the port number of the client computer where the Natural application is being debugged.

Windows Side with Terminal Services

Install the remote debugger (the corresponding files can be found on the UNIX installation medium) or install Natural for Windows (the remote debugger can optionally be installed with a Natural for Windows custom installation; see the *Installation* documentation for Natural for Windows). This also creates the debugger shortcut in the **Start** menu (in the same programs folder in which you can find the shortcuts for Natural) which represents the listener process `natdbgsv`. To use remote debugging, `natdbgsv` must be started. The first time the listener process is launched in a specific user session, a free port number is displayed which must be entered in the corresponding field of the `RDPORT` profile parameter.

Any subsequent activation of `natdbgsv` causes the listener to be started with the same port number. If this number is already used by a different application, then the user must provide `natdbgsv`'s port dialog with a new port number and `RDPORT` must be adjusted accordingly.

Natural Side

Start Natural with the following profile parameter settings:

- RDACTIVE set to "ON".
- RDNODE set to the node name of the Windows server.
- RDPORT set to "2600" or another port number: the number of either port with which you have installed the remote debugging service (see [Windows Side without Terminal Services](#)), or with which port the listener process was started (see [Windows Side with Terminal Services](#)).

Scenarios for Remote Debugging

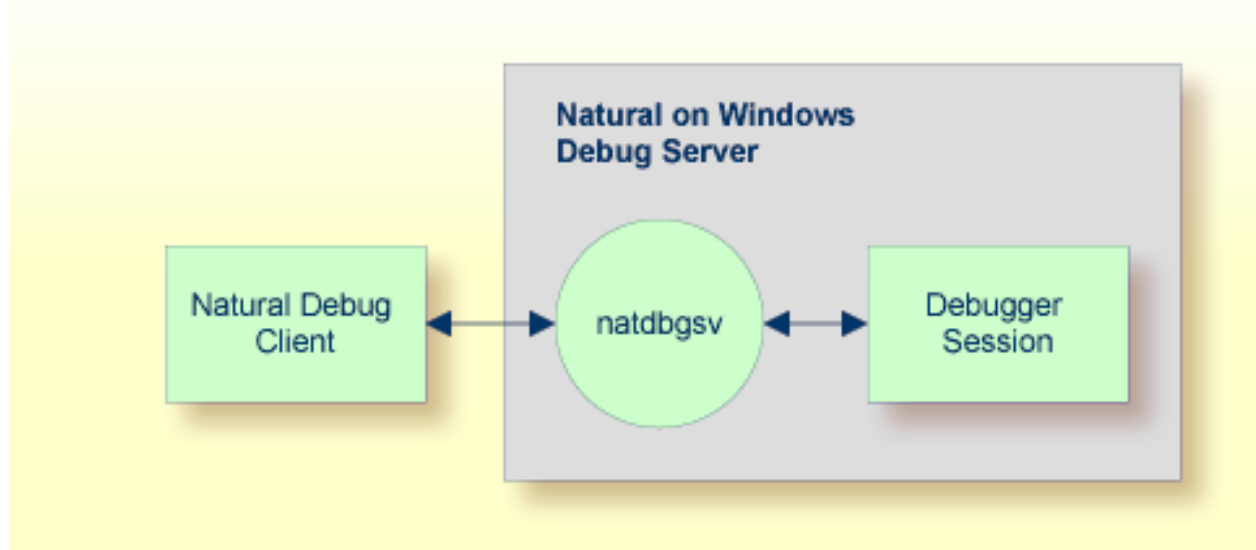
There are different scenarios of how you can use remote debugging: A single Natural client runs under the control of one remote debugging session or a distributed Natural application runs under the control of several remote debugging sessions. Such a distributed application may include both Natural RPC and DCOM servers or even components not written in Natural, such as Visual Basic clients.

The following topics are covered below:

- [Scenario 1: Debugging a Single Natural Application](#)
- [Scenario 2: Debugging a Distributed Natural Application](#)
- [Scenario 3: Debugging the Natural Part of a Heterogeneous Application](#)

Scenario 1: Debugging a Single Natural Application

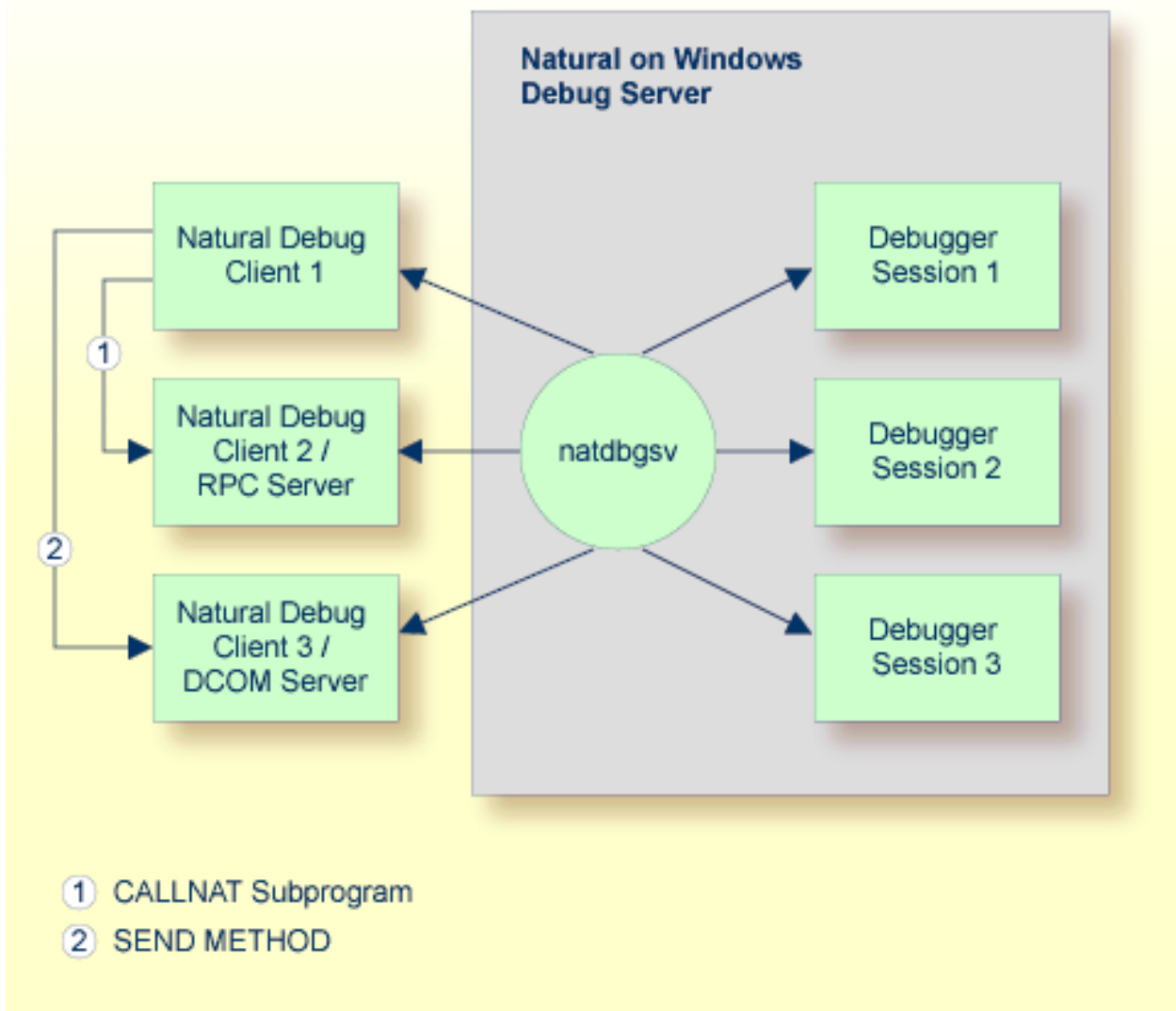
The diagram below illustrates debugging in a single Natural application.



Scenario 2: Debugging a Distributed Natural Application

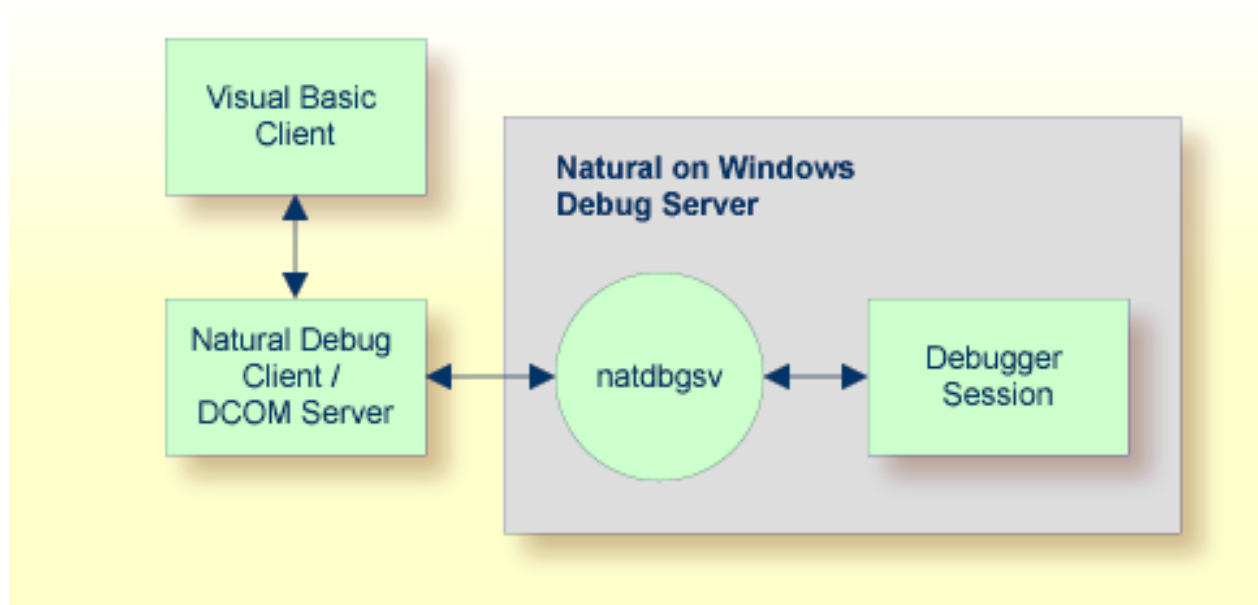
To debug each component of the following distributed Natural application, you enter `DEBUG objectname` in the command line of Natural Debug client 1. The first time the Natural Debug Client calls a subprogram on a Natural RPC server, a new debug session is opened for the RPC server. Then, the RPC server's processing is debugged. The debug session is closed as soon as the RPC server is terminated.

The same applies to a Natural DCOM server.



Scenario 3: Debugging the Natural Part of a Heterogeneous Application

As in the previous scenario, the first time a method on the DCOM server is called, a new debug session is opened for the DCOM server, the DCOM server's processing is debugged, and the debugger session is closed as soon as the DCOM server is terminated:



2 Starting and Leaving the Debugger

- Preparing to Use the Debugger 10
- Starting the Debugger 10
- Restarting the Debugger 11
- Leaving the Debugger 12

Preparing to Use the Debugger

To exploit the full functional scope of the debugger, you must set the parameter `SYMGEN` to "ON". You can set this parameter in one of the following ways:

- dynamically when starting Natural,
- only for the current session by changing the session parameters, or
- in your parameter file using the Configuration Utility.

When you catalog or stow an object and `SYMGEN` is set to "ON", a symbol table is generated as part of the generated program. Since this table contains the information relevant to the variables active for this object, variables cannot be accessed without `SYMGEN` being specified, although it is still possible to debug the object.



Note: It is not necessary to set the parameter `SYMGEN` when debugging in a SPoD environment on a mainframe.

Starting the Debugger

The debugger can be used with stowed or cataloged Natural programs and dialogs. It can be used in the local environment and in the remote environment.

See also the description of the system command `DEBUG`.

▶ To start the debugger

- 1 Open the editor for the object that is to be debugged.

Or:

Select the object in the library workspace.

- 2 From the **Debug** menu, choose **Start**.

Or:

Press `CTRL+F7`.

Or:

When the Debug toolbar is shown, choose the following toolbar button:



Or:

When you have selected an object in the library workspace, invoke the context menu and choose **Debug**.

When the editor for the selected object has not yet been opened, it is opened now.

For a dialog, the dialog source is now shown in a separate window.

When the debugger has been started, additional elements are available in the Natural Studio window. See *Elements of the Debugger* for further information.

Restarting the Debugger

When you restart your debugging session, the debugger repositions to the beginning of the application while all your current settings for breakpoints, watchpoints and watchvariables are kept. Thus, restarting a debugging session is useful if you want to rerun your application without having to specify the settings relevant for debugging again.

▶ To restart the debugger

- From the **Debug** menu, choose **Restart**.

Or:

Press CTRL+SHIFT+F7.

Or:

When the Debug toolbar is shown, choose the following toolbar button:



Leaving the Debugger

The debugger is terminated automatically if the application ends without an error. You can also stop the debugger before it terminates automatically; see the description below.



Note: Closing the editor window does not stop the debugger.

When the debugger is terminated or stopped, your breakpoint, watchpoint and watchvariable settings are automatically stored. All these settings will be restored the next time you start the debugger.

In the case of an error, the corresponding source is displayed and the trace position indicates the line which caused the error. A message window appears with the appropriate error message and a choice to either continue or end the debugging session. Continuing the debugging session may be useful, for example, if your application contains any error processing (including error transactions) or if you want to display any variables before you end your debugging session.

If an error is found in a Natural source and you want to continue debugging with the changed and cataloged source, you first have to stop the debugger. After that, you can start the debugger again with the changed and cataloged source.

▶ To stop the debugger

- From the **Debug** menu, choose **Stop**.

Or:

Press **SHIFT+F7**.

Or:

When the Debug toolbar is shown, choose the following toolbar button:



The debugging session is terminated and control is returned to Natural.

3 Elements of the Debugger

- Debugger Information in the Title Bar 14
- Menu Commands 14
- Toolbar 14
- Trace Position in Editor Window 15
- Debugger Windows 16

When the debugger has been started, additional elements are available in the Natural Studio window.

Debugger Information in the Title Bar

The title bar of the Natural Studio window shows one of the following:

- **[break]**
When "[break]" is shown in the title bar, the debugger has control.
- **[running]**
When "[running]" is shown in the title bar, the Natural application currently being debugged has control.

When you are debugging an object in a remote environment using SPoD, the title bar also shows the port number of the host.

Menu Commands

The commands in the **Debug** menu apply to the debugger.

As long as the debugger has not been started, only the command **Start** is enabled in the **Debug** menu. When the debugger has been started, the remaining commands in the **Debug** menu are enabled and the **Go** command is shown instead of the **Start** command.











When an editor window is active and the debugger has been started for the object in this window, the context menu shows commands which apply to the debugger. As long as the debugger has not been started, only the debug command **Toggle Breakpoint** is available in the context menu.

Detailed descriptions of these commands are provided later in this documentation.

Toolbar

The debugger has a special toolbar which provides fast access to the commands available in the **Debug** menu. As long as the debugger has not been started, only the toolbar buttons for the commands **Start** and **Toggle Breakpoint** are enabled in the Debug toolbar. When the debugger has been started, all other toolbar buttons are enabled.

The buttons in the Debug toolbar represent the following menu commands:

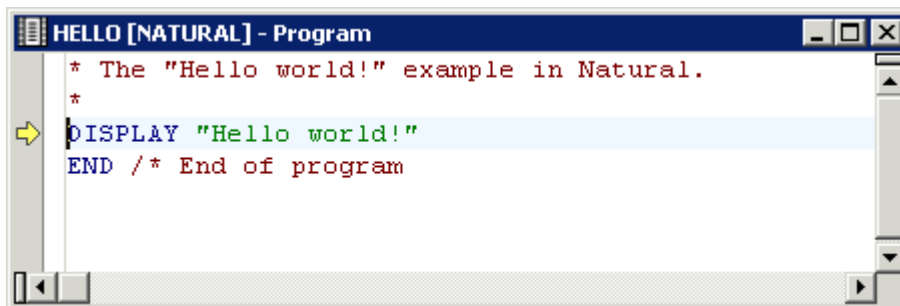
-  **Start** (only shown when the debugger has not yet been started)
-  **Go** (only shown after the debugger has been started)
-  **Restart**
-  **Stop**
-  **Step Over**
-  **Step Into**
-  **Step Out**
-  **Show Trace Position**
-  **Toggle Breakpoint**
-  **Modify Variable**

The display of the Debug toolbar can be switched on and off. See *Customizing Natural Studio* in the *Using Natural Studio* documentation for further information.

Trace Position in Editor Window

The current trace position is indicated by an arrow in the left margin of the editor window.

When the debugger is started, the trace position is shown at the first executable source code line. Example:



When you have scrolled the editor window so that the trace position is no longer visible, you can return to the trace position as described below.

 **Note:** See also *Returning to the Object with the Current Trace Position*.

▶ **To return to the trace position**

- From the **Debug** menu, choose **Show Trace Position**.

Or:

Press ALT+NUM*.



Note: NUM* is the key on the numeric keypad which is used for multiplication.

Or:

When the Debug toolbar is shown, choose the following toolbar button:



Debugger Windows

When the debugger has been started, the following windows are shown:

- Variables
- Break- and Watchpoints
- Call Stack

Each tab of a debugger window offers a context menu which contains either the commands which can be used in combination with the entire tab (when an entry is not selected) or the commands which can be used with the selected entry. These commands are described later in this documentation.

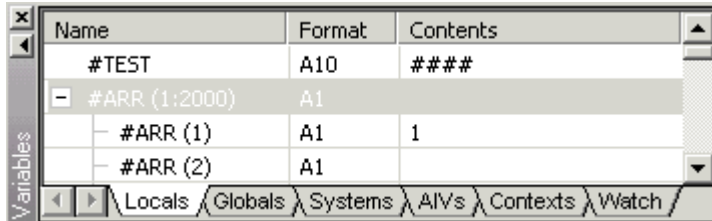
The debugger windows are moveable and dockable. See *Dockable Windows* in the documentation *Using Natural Studio*.




Note: When the display of the debugger window has previously been switched on using the corresponding command in the **View** menu, this debugger window (which shows the breakpoints and watchpoints that have been defined in the active environment) will be replaced by the debugger windows described below. See also *Debugger Window* in the documentation *Using Natural Studio*.

Variables

This window shows all variables which are available at current state of the program execution.



An expand or collapse toggle to the left of the variable name indicates a group, an array or redefined field. The toggle for a redefined field additionally contains an "R" (for example: .

The variables are grouped in different categories. A tab is provided for each category:

- **Locals**
Shows the local variables used in the active generated program.
- **Globals**
Shows the global variables of the referenced global data area.
- **Systems**
Shows all system variables on the current platform. For example, when you are currently debugging an application in a mapped UNIX environment, all system variables which are valid for UNIX are shown.
- **AIVs**
Shows the currently available application-independent variables (AIVs) in the application.
- **Contexts**
Shows the currently available context variables in the application.
- **Watch**
Shows the variables that you have added yourself in order to watch them. See [Adding a Watchvariable](#).

You can switch between the display of the different types of variables by choosing the corresponding tab at the bottom of the variables window.

See [Modifying and Watching Variables](#) for further information.

▶ To switch the variables window display on and off

- From the **Debug** menu, choose **Windows > Variables**.

Or:

Press CTRL+ALT+1.

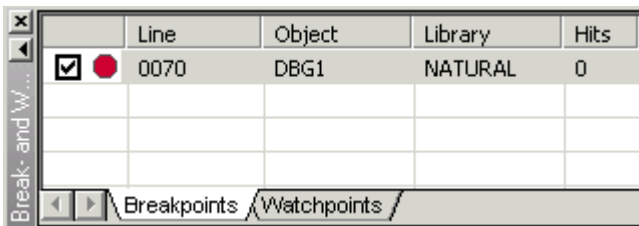
When the variables window is displayed in the Natural Studio window, a check mark is shown next to this menu command.

▶ **To activate the variables window using a shortcut key**

- When the variables window is displayed in the Natural Studio window, press CTRL+SHIFT+V to activate it.

Break- and Watchpoints

This window shows all currently defined breakpoints and watchpoints.



You can switch between the display of the watchpoints and breakpoints by choosing the corresponding tab at the bottom of the break- and watchpoints window.

See [Setting Breakpoints and Watchpoints](#) for further information.

▶ **To switch the break- and watchpoints window display on and off**

- From the **Debug** menu, choose **Windows > Break- and Watchpoints**.

Or:

Press CTRL+ALT+2.

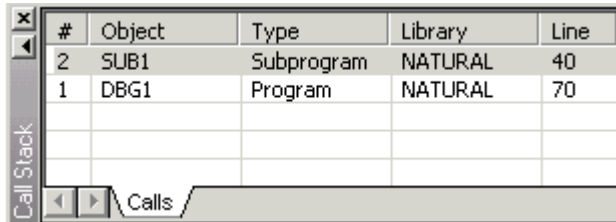
When the break- and watchpoints window is displayed in the Natural Studio window, a check mark is shown next to this menu command.

▶ **To activate the break- and watchpoints window using a shortcut key**

- When the break- and watchpoints window is displayed in the Natural Studio window, press CTRL+SHIFT+B to activate it.

Call Stack

This window shows the objects which have been called during the current debugging session in hierarchical order.



#	Object	Type	Library	Line
2	SUB1	Subprogram	NATURAL	40
1	DBG1	Program	NATURAL	70

See [Using the Call Stack](#) for further information.

▶ To switch the call stack window display on and off

- From the **Debug** menu, choose **Windows > Call Stack**.

Or:

Press CTRL+ALT+3.

When the call stack window is displayed in the Natural Studio window, a check mark is shown next to this menu command.

▶ To activate the call stack window using a shortcut key

- When the call stack window is displayed in the Natural Studio window, press CTRL+SHIFT+C to activate it.

4 Moving through the Code

- Stepping Through the Code 22
- Going to the Next Breakpoint or Watchpoint 23
- Going to the Next Event 24
- Going to the Cursor Position 24
- Going to the Next Statement 25

Stepping Through the Code

You can instruct the debugger to execute the next program step. Different commands are available for this purpose:

- [Stepping Over Another Object](#)
- [Stepping Into Another Object](#)
- [Stepping Out Of Another Object](#)

Stepping Over Another Object

When you instruct the debugger to step over another object, the next program step is executed and the trace position is shown at the corresponding source code line. If this source code line invokes or includes a further Natural object, the debugger steps over this object; that is, all source code of this object is executed at once. The debugger stops, however, if this object contains watchpoints or breakpoints.

▶ To step over another object

- From the **Debug** menu, choose **Step Over**.

Or:

Press F10.

Or:

When the Debug toolbar is shown, choose the following toolbar button:



Stepping Into Another Object

When you instruct the debugger to step into another object, the next program step is executed and the trace position is shown at the corresponding source code line. If this source code line invokes or includes a further Natural object, the debugger steps into this object and the trace position is shown at the first executable line.

▶ To step into another object

- From the **Debug** menu, choose **Step Into**.

Or:

Press F11.

Or:

When the Debug toolbar is shown, choose the following toolbar button:



Stepping Out Of Another Object

When you instruct the debugger to step out of another object, the debugger returns to the previous program level. The debugger stops, however, if a watchpoint or breakpoint is found before this previous level is reached.

This command is useful if you debug a subprogram and want to continue with the execution of the rest of the subprogram. The execution continues without interruption and stops after the position in the invoking program from which the subprogram has been invoked.

▶ To step out of another object

- From the **Debug** menu, choose **Step Out**.

Or:

Press CTRL+F11.

Or:

When the Debug toolbar is shown, choose the following toolbar button:



Going to the Next Breakpoint or Watchpoint

You can instruct the debugger to execute the object until the next active breakpoint is found or until a watchpoint condition becomes true. In this case, the debugger stops at the watchpoint or breakpoint and the trace position is shown at the corresponding source code line.

▶ To go to the next watchpoint or breakpoint

- From the **Debug** menu, choose **Go**.

Or:

Press F7.

Or:

When the Debug toolbar is shown, choose the following toolbar button:



Going to the Next Event

In an event-driven application, you can instruct the debugger to execute the object until the next event is sent to the application. The debugger stops, however, if an active watchpoint or breakpoint occurs before the next event is sent.

▶ To go to the next event

- From the **Debug** menu, choose **Go Until Next Event**.



Note: In a non-event driven application, this command has the same effect as the **Go** command.

Or:

Press ALT+F7.

Going to the Cursor Position

You can instruct the debugger to execute the object until the source code line at the current cursor position is reached.

▶ To go to the cursor position

- 1 Place the cursor in the source code line at which execution is to be paused.
- 2 Invoke the context menu in the editor and choose **Run to Cursor**.

Or:

Press CTRL+F10.

Going to the Next Statement

You can instruct the debugger to skip code and to resume execution of the object with the source code line in which you have placed the cursor. The skipped code is not executed.



Caution: Depending on the code you want to skip, this command may lead to unpredictable results. Use this command with care.

▶ To go to the next statement

- 1 Place the cursor in the source code line with which you want to resume execution.
- 2 Invoke the context menu in the editor and choose **Set Next Statement**.



Note: This command is only available for the object which is currently processed.

5

Setting Breakpoints and Watchpoints

▪ About Breakpoints and Watchpoints	28
▪ Adding and Removing a Breakpoint	29
▪ Modifying a Breakpoint	30
▪ Adding a Watchpoint	31
▪ Modifying a Watchpoint	33
▪ Deactivating Breakpoints and Watchpoints Temporarily	34
▪ Showing the Source Code for a Defined Breakpoint or Watchpoint	35
▪ Deleting Breakpoints and Watchpoints	35
▪ Symbols Used in the Editor Window	36

About Breakpoints and Watchpoints

Two types of entries can be defined in a program for debugging purposes:

■ Breakpoints

A breakpoint is a point at which control is returned to the user while a Natural object is executing.

Breakpoints cannot be set on any statement line other than the first one if a single statement occupies more than one line.

If you accidentally try to set a breakpoint on a non-executable line (for example, a comment line), the breakpoint is automatically moved to the next executable line.

■ Watchpoints

Using watchpoints, you can rapidly detect unexpected alterations to Natural variables by objects that contain errors.

By default, watchpoints are used to instruct the debugger to interrupt the execution of Natural objects when the content of a variable changes. However, by specifying a certain value to the variable together with a watchpoint operator when setting a watchpoint, a condition can be set which only activates the watchpoint when the condition becomes true.

A variable is considered to have changed either when its current value differs from the value recorded when the watchpoint was last triggered, or when it differs from the initial value.

Each breakpoint or watchpoint is displayed in the corresponding tab of the **break- and watchpoints** window. For each breakpoint, the number of the line is shown in which the breakpoint has been defined. For each watchpoint, a name is assigned that corresponds to the name of the variable to which it belongs and the break condition is shown.

Using the check box in the first column of a tab, a breakpoint or watchpoint can be activated or deactivated at any time during a debugging session. See [Deactivating Breakpoints and Watchpoints Temporarily](#).

Every breakpoint or watchpoint has a hit count which increases every time the debug entry is passed. The number of executions of a debug entry, however, can be restricted in the following ways:

- A number of skips can be specified before the breakpoint or watchpoint is executed. The debug entry is then ignored until the event count is higher than the number of skips specified.
- A maximum number of executions can be specified, so that the breakpoint or watchpoint is ignored as soon as the event count exceeds the specified number of executions.

When a breakpoint or watchpoint is hit inside another object but the currently active one, a new editor window is opened displaying the source of this new object.

Adding and Removing a Breakpoint

You can add a breakpoint for the current cursor position to the **Breakpoints** tab of the break- and watchpoints window. Or, if a breakpoint already exists for this cursor position, you can remove it from the **Breakpoints** tab.

A dialog box does not appear in this case. If you want to modify the breakpoint (for example, to define the maximum number of breaks), see [Modifying a Breakpoint](#).

See also [Deleting Breakpoints and Watchpoints](#).



Note: In the local environment and in a remote environment (SPoD), it is also possible to set and remove breakpoints as described below when the debugger is not active. Each breakpoint which has been defined in the editor will then be verified when the debugger is started. When the breakpoint is not allowed at the defined position, it will then be moved to the next line in which it is possible to define a breakpoint. See also *Debugger Window* in the documentation *Using Natural Studio*.

▶ To toggle a breakpoint

- 1 Select the line on which you want to set or remove a breakpoint.
- 2 Invoke the context menu in the editor and choose **Toggle Breakpoint**.

Or:

Press F9.

Or:

When the Debug toolbar is shown, choose the following toolbar button:



Or:

In the left margin of the editor window (where the symbols for the breakpoints are usually shown), click a position next to the required line.

When a breakpoint is set, a symbol is now shown in the left margin of the editor window. See [Symbols Used in the Editor Window](#). An entry for the breakpoint is also shown on the **Breakpoints** tab of the break- and watchpoints window.

When a breakpoint has been removed, the corresponding symbol is no longer shown. The entry for the breakpoint is removed from the **Breakpoints** tab of the break- and watchpoints window.

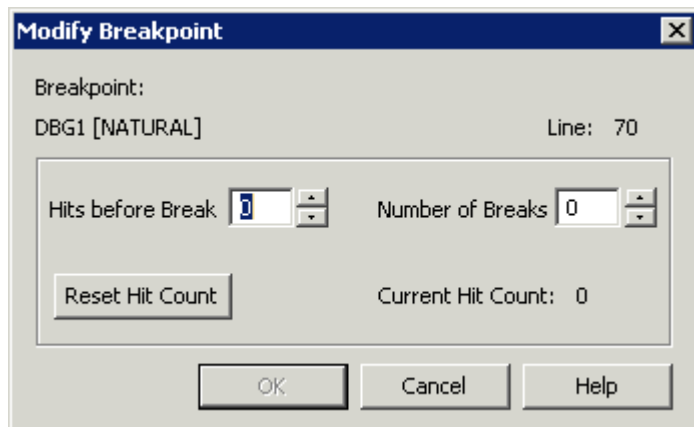
Modifying a Breakpoint

You can modify each breakpoint which is currently shown in the break- and watchpoints window.

▶ To modify a breakpoint

- 1 Select the required breakpoint, invoke the context menu and choose **Modify**.

The following dialog box appears:



- 2 Set the required options:

Hits before Break

The number of skips before execution of the breakpoint if it is not to be executed until the program has run a certain number of times. The default is 0.

Number of Breaks

The maximum number of executions of the breakpoint. After this number has been reached, the breakpoint is ignored. The default is 0.

Reset Hit Count

When you choose this command button, the current hit count is reset to 0.

- 3 Choose the **OK** button.

Adding a Watchpoint

Watchpoints are shown on the **Watchpoints** tab of the break- and watchpoints window.

You can add watchpoints in different ways:

- [Adding a Watchpoint from the Editor Window](#)
- [Adding a Watchpoint from the Variables Window](#)
- [Adding a Watchpoint Using a Dialog Box](#)

Adding a Watchpoint from the Editor Window

You can add the variable at the current cursor position in the editor window to the **Watchpoints** tab of the break- and watchpoints window. A dialog is not shown in this case.

▶ To define a variable as a watchpoint

- 1 Select the variable in the editor by placing the cursor at any position within the variable name.
- 2 Invoke the context menu and choose **Add to Watchpoints**.

Or:

Press CTRL+SHIFT+W.

Or:

Select the variable in the editor. Use the mouse to drag the selected variable to the **Watchpoints** tab and drop it there.

Adding a Watchpoint from the Variables Window

You can add a variable from the variables window to the **Watchpoints** tab of the break- and watchpoints window. A dialog is not shown in this case.

▶ To define a variable as a watchpoint

- 1 Select the desired variable in the variables window.
- 2 Invoke the context menu and choose **Add to Watchpoints**.

Or:

Press CTRL+SHIFT+W.

Adding a Watchpoint Using a Dialog Box

You can use a dialog box to add a watchpoint to the **Watchpoints** tab of the break- and watchpoints window.

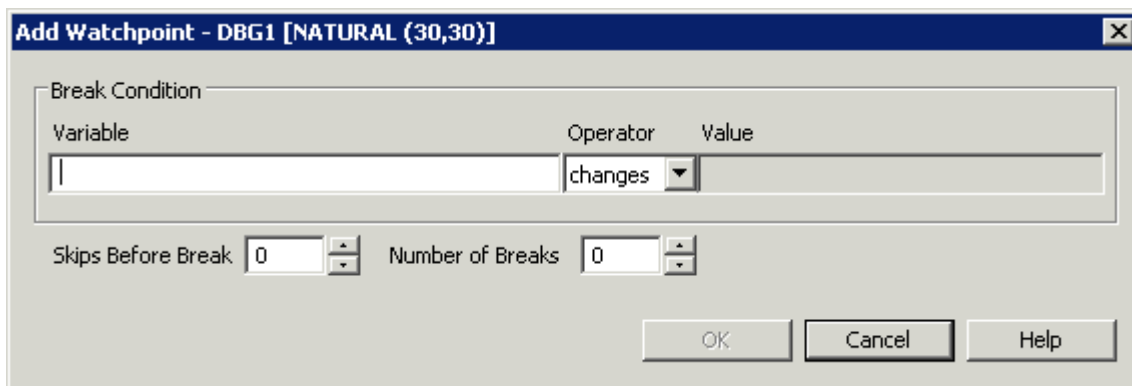
▶ To add a watchpoint

- 1 From the **Debug** menu, choose **Add Watchpoint**.

Or:

In the **Watchpoints** tab of break- and watchpoints window, invoke the context menu and choose **Add**. Make sure that no other entry is selected. Otherwise, the context menu does not show this command.

The **Add Watchpoint** dialog box appears. The title bar indicates the names of the current program and library as well as the database ID and file number of the current FUSER.



- 2 Set the required options:

Variable

The variable that is to be watched in the debugged program.

Operator/Value

To define a condition for the watchpoint, select an appropriate watchpoint operator and specify a value for this operator. If you do not specify a condition, the default setting ("changes") applies.

The watchpoint operators are:

Operator	Activation of the Watchpoint
changes	Each time the variable is changed. Default.
EQ (=)	Only when the current value of the variable is equal to the specified value.
NE (!=)	Only when the current value of the variable is not equal to the specified value.
GT (>)	Only when the current value of the variable is greater than the specified value.
LT (<)	Only when the current value of the variable is less than the specified value.
GE (>=)	Only when the current value of the variable is greater than or equal to the specified value.
LE (<=)	Only when the current value of the variable is less than or equal to the specified value.

Skips before Break

The number of skips before execution of the watchpoint if it is not to be executed until the program has run a certain number of times. The default is 0.

Number of Breaks

The maximum number of executions of the watchpoint. After this number has been reached, the watchpoint is ignored. The default is 0.

- 3 Choose the **OK** button.

The name of the selected variable is now shown on the **Watchpoints** tab of the break- and watchpoints window.

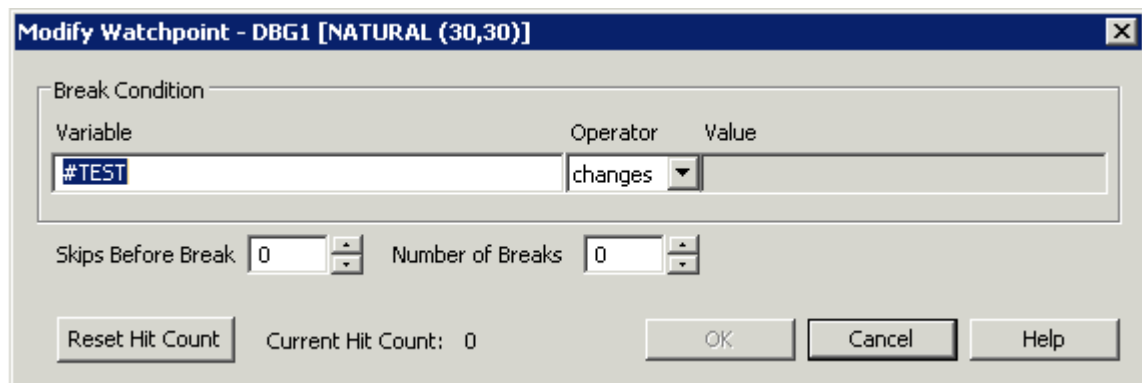
Modifying a Watchpoint

You can modify each watchpoint which is shown in the break- and watchpoints window.

▶ To modify a watchpoint

- 1 Select the required watchpoint, invoke the context menu and choose **Modify**.

The **Modify Watchpoint** dialog box appears.



This dialog box provides the same options as the **Add Watchpoint** dialog box. See [Adding a Watchpoint Using a Dialog Box](#) for a description of the options which can be specified in this dialog box.

In addition, this dialog box provides the **Reset Hit Count** button. When you choose this command button, the current hit count is reset to 0.

- 2 Make all required changes and choose the **OK** button.

Deactivating Breakpoints and Watchpoints Temporarily

Each defined breakpoint or watchpoint can be deactivated temporarily.

▶ To deactivate a breakpoint or watchpoint

- 1 Select the required tab in the break- and watchpoints window.
- 2 For the desired entry, select the first column of the tab to remove the check mark.

Or:

Invoke the context menu and choose **Activate/Deactivate**.

When you have deactivated a breakpoint, the symbol which is shown in the left margin of the editor window changes. See [Symbols Used in the Editor Window](#).

▶ To activate a deactivated breakpoint or watchpoint

- 1 Select the required tab in the break- and watchpoints window.
- 2 For the desired entry, select the first column of the tab so that a check mark is shown again.

Or:

Invoke the context menu and choose **Activate/Deactivate**.

When you have activated a breakpoint, the symbol which is shown in the left margin of the editor window changes. See [Symbols Used in the Editor Window](#).

▶ To deactivate or activate all breakpoints or watchpoints

- 1 Select the required tab in the break- and watchpoints window.
- 2 Make sure that no entry is selected (otherwise, the context menu does not show the required command), invoke the context menu and choose either **Deactivate All** or **Activate All**.

When you have deactivated or activated all breakpoints, the symbols which are shown in the left margin of the editor window change. See [Symbols Used in the Editor Window](#).

Showing the Source Code for a Defined Breakpoint or Watchpoint

For each defined breakpoint or watchpoint which is shown in the break- and watchpoints window (no matter whether it is active or not), you can go to the source in which this breakpoint or watchpoint has been defined.

▶ To go to the source in which a breakpoint or watchpoint has been defined

- 1 Select the required tab in the break- and watchpoints window.
- 2 Select the required entry, invoke the context menu and choose **Go To Source Code**.

Or:

Double-click the required entry.

For a breakpoint, the trace position is shown next to the source code line in which the breakpoint has been defined.

For a watchpoint, the entire source code in which the watchpoint has been defined is shown.

Deleting Breakpoints and Watchpoints

You can either delete selected breakpoints or watchpoints or you can delete all breakpoints or watchpoints.

See also [Adding and Removing a Breakpoint](#).

▶ To delete a breakpoint or watchpoint

- 1 Select the required tab in the break- and watchpoints window.
- 2 Select the required entry.
- 3 Invoke the context menu and choose **Delete**.

Or:







Press DEL.

▶ **To delete all breakpoints or watchpoints**

- 1 Select the required tab in the break- and watchpoints window.
- 2 Make sure that no entry is selected (otherwise, the context menu does not show the required command), invoke the context menu and choose **Delete All**.

Symbols Used in the Editor Window

The following symbols may appear in the left margin of the editor window.

-  This line contains an active breakpoint.
-  This line contains a deactivated breakpoint.
-  This line contains an active breakpoint. It also contains the trace position.
-  This line contains a deactivated breakpoint. It also contains the trace position.
-  This line contains a breakpoint which has not yet been validated (that is, the debugger has not yet reached the marked line). The state can either be shown as active (red background) or inactive (white background).
-  This line contains an invalid breakpoint (for example, when the breakpoint has been set on a line after the END statement). The state can either be shown as active (red background) or inactive (white background).

6

Modifying and Watching Variables

- Modifying a Variable 38
- Adding a Watchvariable 41
- Managing the Variables in the Variables Window 42

Modifying a Variable

You can modify a variable in different ways:

- [Modifying a Variable in the Editor Window](#)
- [Modifying a Variable in the Variables Window](#)

Modifying a Variable in the Editor Window

You can modify the variable which is shown at the current cursor position in the editor window. In the resulting dialog box, you can also enter the name of another variable to be modified.

▶ To modify the variable at the cursor position

- 1 Select the variable in the editor by placing the cursor at any position within the variable name.
- 2 Invoke the context menu and choose **Modify Variable**.

Or:

Press **SHIFT+F9**.

Or:

When the Debug toolbar is shown, choose the following toolbar button:



The **Modify Variable** dialog box appears showing the content of the selected variable. In the case of an array, the node is expanded by default.

Variables (and occurrences) which you have modified are indicated in red.

- 4 When you activate the **Hexadecimal Display** check box, the content of the variable is shown in hexadecimal format.
- 5 Choose the **Apply** button.

Your changes are immediately saved when you choose the **Apply** button. They are not yet shown the in variables window.

- 6 To close the dialog box, choose the **Close** button.

Your changes are now shown the in variables window.

Modifying a Variable in the Variables Window

You can modify a variable listed in the variables window.

It is not possible to modify an entry which can further be expanded (such as a view). This is only possible for the individual variables after the entry has been expanded.

Different colors are used for the entries in the variables window:

- **Gray**

Variables which cannot be modified (such as unmodifiable system variables) are indicated in gray.

- **Red**

Variables which you have modified are indicated in red.

- ▶ **To modify a variable in the variables window**

- 1 Select the required tab in the variables window.
- 2 Select the required entry.
- 3 Invoke the context menu and choose **Modify**.

The **Modify Variable** dialog box appears showing the content of the selected variable.

For further information on this dialog box, see [Modifying a Variable in the Editor Window](#).

Adding a Watchvariable

If you want to watch specific variables, you can add them to the **Watch** tab of the **variables window**.

You can add a watchvariable in different ways:

- [Adding a Watchvariable from the Editor Window](#)
- [Adding a Watchvariable from the Variables Window](#)



Note: It is not possible to modify the content of a watchvariable.

Adding a Watchvariable from the Editor Window

You can define the variable at the current cursor position in the editor window as a watchvariable.

▶ To add a watchvariable to the variables window

- 1 Select the variable in the editor by placing the cursor at any position within the variable name.
- 2 Invoke the context menu in the editor and choose **Add to Watchvariables**.

Or:

Press CTRL+SHIFT+T.

Or:

Select the variable in the editor. Use the mouse to drag the selected variable to the **Watch** tab of the variables window and drop it there.

Adding a Watchvariable from the Variables Window

You can add variables from the first tabs of the variables window to the **Watch** tab of the same window.

▶ To define a variable as a watchvariable

- 1 Select the desired variable in the variables window.
- 2 Invoke the context menu and choose **Add to Watchvariables**.

Or:

Press CTRL+SHIFT+T.

Managing the Variables in the Variables Window

The following topics are covered below:

- [Showing the Last Modified Variable](#)
- [Finding a Variable](#)
- [Showing the Content of a Variable in Alphanumeric or Hexadecimal Format](#)
- [Refreshing the Display](#)
- [Deleting Watchvariables](#)

See also [Adding a Watchpoint from the Variables Window](#).

Showing the Last Modified Variable

You can define that the variables which are modified during debugging are always visible in the variables window. This is helpful when you debug a program which has more variables than can be displayed in the variables window at the same time. When a variable is modified during debugging which is currently not visible in the variables window, the display of the variables window is scrolled in such a way so that the modified variable is visible.

▶ To switch this feature on or off

- 1 Select any tab in the variables window.
- 2 Make sure that no entry is selected on the tab (otherwise, the context menu does not show the required command), invoke the context menu and choose **Show Last Modified**.

When this feature is active, a check mark is shown next to this menu command.

Finding a Variable

When the variables window is active, you can search for a variable on the currently selected tab.



Note: When a node in the variables window is not expanded, its content is not considered in the search.

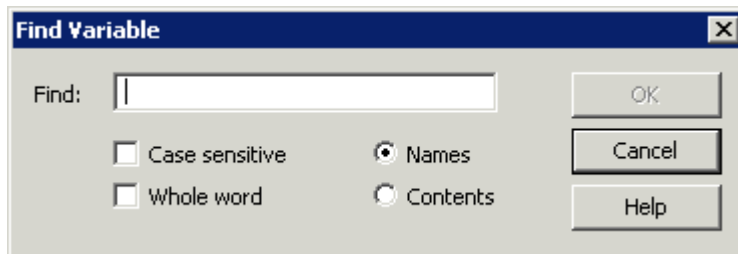
▶ To find a variable

- 1 Select the required tab in the variables window.
- 2 Press CTRL+F.

Or:

Make sure that no entry is selected on the tab (otherwise, the context menu does not show the required command), invoke the context menu and choose **Find**.

The **Find Variable** dialog box appears.



- 3 Specify your search criteria:

Option	Description
Find	The string to be found.
Case sensitive	If this check box is selected, only strings are found that exactly match the entry in the Find text box. If not selected, any combination of upper- and lower-case letters will be found.
Whole word	If this check box is selected, the search is restricted to whole words only. If not selected, all occurrences of the string will be found.
Names	When this option button is selected, the string in the Find text box applies to a variable name.
Contents	When this option button is selected, the string in the Find text box is applies to the contents of a variable. That is: you want to find a variable which contains the specified contents.

- 4 Choose the **OK** button.

When a variable which corresponds to the specified criteria can be found on the current tab, its name is highlighted.



Note: A message is briefly displayed indicating whether the specified text has been found or not.

▶ **To find the next variable with the specified search criteria**

- Press F3.

Or:

Make sure that no entry is selected on the tab (otherwise, the context menu does not show the required command), invoke the context menu and choose **Find Next**.

Showing the Content of a Variable in Alphanumeric or Hexadecimal Format

You can define whether the contents of the variables is shown in alphanumeric or hexadecimal format in the variables window.

▶ To toggle the format

- 1 Select the required tab in the variables window.
- 2 Make sure that no entry is selected on the tab (otherwise, the context menu does not show the required command), invoke the context menu and choose **Hexadecimal Display**.

When the value in the **Contents** column was previously shown in the alphanumeric format, it is now shown in hexadecimal format, and vice versa.

When the hexadecimal format is used, a check mark is shown next to this menu command.

Refreshing the Display

Usually when something changes in Natural Studio, the display is automatically refreshed. In the debugger, this happens when the content of a variable changes. This automatic refresh requires that the corresponding option has been set in the workspace options.

When the automatic refresh has been deactivated in the workspace options and the content of one or more variables changes in the currently selected tab, you have to refresh the display manually in order to see the current values.

There is one exception: Watchvariables are always refreshed automatically, independent of the setting in the workspace options.

▶ To refresh the display manually

- 1 Select any tab in the variables window (except the **Watch** tab).
- 2 Press F5.

Or:

Make sure that no entry is selected on the tab (otherwise, the context menu does not show the required command), invoke the context menu and choose **Refresh**.

Deleting Watchvariables

You can either delete selected watchvariables or all watchvariables from the variables window.

▶ To delete a watchvariable

- 1 Select the desired watchvariable in the **Watch** tab of the variables window.
- 2 Invoke the context menu and choose **Delete**.

Or:

Press DEL.

▶ To delete all watchvariables

- Make sure that no entry is selected in the **Watch** tab of the variables window (otherwise, the context menu does not show the required command), invoke the context menu and choose **Delete All**.

7 Using the Call Stack

- About the Call Stack 48
- Displaying the Source Code of a Different Object 48
- Returning to the Object at the Current Trace Position 49

About the Call Stack

The **call stack window** lists the objects which have been called during the current debugging session in hierarchical order.

The latest object is always shown at the top of the list. The **variables window** shows all variables which belong to this object by default. For example, when you step into a subprogram, this subprogram is shown at the top of the list and the variables window automatically shows the variables for this subprogram.

You can bring the editor window for a specific object to the front by double-clicking the corresponding entry in the call stack window.



Notes:

1. A gray arrow in the editor window indicates the position at which the previous object in the call stack hierarchy was invoked.
2. If copycode is debugged, the call stack does not contain an additional entry for this copycode.

Displaying the Source Code of a Different Object

For each object listed in the call stack, you can display the source code and thus bring its editor window to the front. There are different commands for this purpose:

■ Go To Source Code

When you choose this command, the variables for the object in the activated editor window are not considered in the variables window. It still shows the variables of the previously called object.

■ Switch To Call Level

When you choose this command, the variables for the object in the activated editor window are shown in the variables window.

▶ To go to the source code of a different object

- In the call stack, select the object for which you want to display the source code and from the context menu, choose **Go To Source Code**.

The editor window for this object is activated.

▶ **To go to the source code of a different object and display the variables of this object**

- In the call stack, select the object for which you want to display the source code and from the context menu, choose **Switch To Call Level**.

The editor window for this object is activated. The content of the variables window changes; it now shows variables of this object.

Returning to the Object at the Current Trace Position

When you have displayed the source code of a different object, you can return to the object at the current trace position (which is indicated by an arrow) and thus bring its editor window to the front.

▶ **To return to the object at the current trace position**

- From the **Debug** menu, choose **Show Trace Position**.



Note: See also *Trace Position in Editor Window*.

The editor window containing the current trace position is activated. The content of the variables window changes; it now shows the variables of this object.

8 Using the Old Debugger

▪ Preparing Natural Objects	52
▪ Starting the Debugger	52
▪ Leaving the Debugger	53
▪ Operating the Debugger	54
▪ Debugger Source Window	56
▪ Watchvariables Control Bar	63
▪ Variables Control Bar	63
▪ Watchpoints and Breakpoints Control Bar	64

The old debugger may appear when an old version of Natural is installed on the development server; see [General Information](#) for further information.

Preparing Natural Objects

To exploit the full functional scope of the Natural debugger, you must specify the following Natural profile parameter either dynamically or in your Natural parameter file:

SYMGEN set to "ON"

When an object is cataloged or stowed and SYMGEN is set to "ON", a symbol table is generated as part of the generated program. Since this table contains the information relevant to the variables active for this object, variables cannot be accessed without SYMGEN being specified, although it is still possible to debug the object.

Starting the Debugger

The debugger can be applied to stowed or cataloged Natural programs and dialogs only.

▶ To start the debugger

- Enter the following Natural command:

```
DEBUG objectname
```

where *objectname* is the name of the Natural object you wish to debug.

The title bar shows one of the following:

- **[break]**
When "[break]" is shown in the title bar, the debugger has control.
- **[waiting]**
When "[waiting]" is shown in the title bar, the Natural application currently being debugged has control.

When the remote debugger becomes active on the Windows operating system, the following information is shown in the title bar: "Debugging remote Natural client (*nodename*::*username*::*process-id*)", where *nodename* is the name of the computer where Natural is running, *username* is the name of the Natural user and *process-id* is the Natural process ID.

The debugger window contains a child window with a source listing of the specified object that is to be debugged.

In conjunction with this object source following information are displayed using control bars:

Control Bar	Function
Breakpoints and Watchpoints	This control bar consists of two tab areas. One maintains breakpoints whereas the other one maintains watchpoints.
Variables	This control bar displays the active variables and their actual content. These variables are displayed under the following categories: Locals, Globals, Systems, AIVs and Contexts.
Watchvariables	This control bar displays the user-selected variables of any category available in the variables control bar.

The individual control bars are described in more detail in the remainder of this section.

Leaving the Debugger

You can leave the debugger from any point within an application by choosing either **Exit** (see below) or the corresponding toolbar button.

The debugger is also terminated if the application ends without an error; the trace cursor is then placed on the source code line last executed.

In the case of an error, the corresponding source is displayed in the source window and the trace cursor is placed on the line which caused the error. A message window appears with the appropriate error message and a choice to either continue or end the debugging session. Continuing the debugging session may be useful if, for example:

- your application contains any error processing (including error transactions);
- you want to display any variables before you end your debugging session.

When you leave the debugger, your breakpoint, watchpoint and watchvariable settings are automatically saved together with the window and toolbar settings. All these settings will be restored the next time you invoke the debugger again.



Note: When, in the case of remote development, you leave the debugger on a remote system, the program execution will continue, but the debugging control of the program execution will stop.

Exit Command

Exit terminates the debugging session and returns control to Natural. The **Exit** command is available on the first menu in the main window of each of the five debugger main facilities.

Operating the Debugger

Before going into detail about the debugger's source window and other main facilities, this section provides you with general information on the debugger.

- [Windows and Menus](#)
- [Toolbar Buttons](#)
- [Shortcut Keys](#)
- [Watchpoints and Breakpoints](#)
- [Restarting the Debugging Session](#)

Windows and Menus

The debugger provides various windows, control bars, toolbars and menus.

Menu commands which are assumed to be used very often, are also available as [toolbar buttons](#) in the corresponding toolbars.

Instead of using the menus, you can choose toolbar buttons or use [shortcut keys](#).

In contrast to Natural itself:

- the debugger has no command line.
- the debugger's **Tools** menu contains the following options:
 - **Customize**, which allows you to modify your menu and toolbar appearance as well as define shortcuts for frequently used commands;
 - **Fonts**, which allows you to modify the font of the source window;
 - **Warning messages**, which allows you to decide whether warning messages on missing source code or symbolic information are to be displayed or not. A message that informs you whether the currently displayed source code is newer than the corresponding generated program is also affected.

Toolbar Buttons

The toolbars provide you with fast access to frequently used commands. To display a short description of a command, place the mouse pointer over the corresponding button. The description appears in the status bar at the bottom of the debugger's main window. If a command is currently not applicable, the button is disabled.

Shortcut Keys

A further way to execute a debugger command is by entering a corresponding shortcut by using the keyboard. By default the following shortcuts are defined:

Menu	Shortcut	Function
File	Ctrl+O	Open
Edit	Ctrl+F	Find
	F3	Find Next
Debug	F4	Close
	F5	Go
	F6	Step Over
	F7	Step In
	Ctrl+F7	Step Out
	Ctrl+F6	Run To Cursor
	Alt+*	Show Trace Position
	F9	Toggle Breakpoint
Variables	Ctrl+M	Modify Variable
	Ctrl+D	Display Variable
	Ctrl+V	Add to Watchvariables
	Ctrl+W	Add to Watchpoints

Watchpoints and Breakpoints

Two types of entries can be defined in a program for debugging purposes: **watchpoints** and **breakpoints**. Each watchpoint or breakpoint is displayed in its corresponding control bar. For each watchpoint, a name is assigned that corresponds to the name of the variable it belongs to.

Each watchpoint or breakpoint can be activated or deactivated at any time during a debugging session using its corresponding check box.

Every watchpoint or breakpoint has an event count, which increases every time the debug entry is passed. The number of executions of a debug entry, however, can be restricted in two ways:

1. A number of skips can be specified before the watchpoint or breakpoint is executed. The debug entry is then ignored until the event count is higher than the number of skips specified.
2. A maximum number of executions can be specified, so that the watchpoint or breakpoint is ignored as soon as the event count exceeds the specified number of executions.

Restarting the Debugging Session

When you restart your debugging session, the debugger repositions to the beginning of the application while all your current settings (for example, watchpoints or breakpoints) are kept and all counters as well as the **calls** history are newly initialized. Thus, restarting a debugging session is useful if you want to rerun your application without having to specify the settings relevant for debugging again. You can restart your debugging session from any point within an application by choosing either the "Restart" command or the corresponding toolbar icon. The **Restart** command is available in the debug menu.



Note: If you are running a debugging session in a remote environment, the **Restart** command is not available, and if you are debugging a DCOM or RPC server, the **Restart** command restarts the called method or subprogram.

Debugger Source Window

When the debugger is invoked, it receives control of the specified Natural object and displays the corresponding source in the source window. When the source is not available, the window remains empty. The trace cursor is placed on the first executable source code line.

When a user opens a new object or when a watchpoint or breakpoint is hit inside another object but the currently active one, a new source window is opened displaying the source of this new object.

The following topics are covered below:

- [Debug Menu](#)
- [Variables Menu](#)
- [Dialog Boxes](#)
- [Selecting Variables](#)
- [Marking Text in the Source Window](#)
- [Display](#)
- [Modify](#)
- [Quick Watch](#)
- [Add Watch](#)
- [Add Watchpoint](#)
- [File Menu](#)

- [Edit Menu](#)

Debug Menu

The following commands of the **Debug** menu are available in conjunction with the source window:

Step Into

When you choose the **Step Into** command, the next program step is executed and the trace cursor is placed on the corresponding source code line.

If this source code line invokes or includes a further Natural object, the debugger steps into this object.

Step Over

When you choose the **Step Over** command, the next program step is executed and the trace cursor is placed on the corresponding source code line. This time, however, the debugger steps over any invoked or included Natural object, but stops if this object contains watchpoints or breakpoints.

Step Out

When you choose the **Step Out** command, the debugger returns to the previous program level, but stops if it finds a watchpoint or breakpoint before this previous level is reached.

Animated Step Into

When you choose the **Animated Step Into** command, the program is automatically executed step by step until the end of the program. The debugger steps into any Natural object invoked or included.

Animated Step Over

When you choose the **Animated Step Over** command, the program is automatically executed step by step until the end of the program. The debugger steps over any invoked or included Natural object; if a watchpoint or breakpoint is set, it jumps to the corresponding statement line and continues animation.

Go

When you choose the **Go** command, the program is executed until the next active watchpoint or breakpoint, and the trace cursor is placed on the corresponding source code line.

Go Until Next Event

When you choose the **Go Until Next Event** command, this will have the same effect as the **Go** command in a non-event driven application. In an event-driven application, however, the object is executed until the next event is sent to the application; it stops if an active watchpoint or breakpoint occurs before the next event is sent.

Run to Cursor

When you choose the **Run to Cursor** command, the program is executed until the source line at the current cursor position is reached.

Show Trace Position

When you choose the **Show Trace Position** command, the current trace cursor will be displayed.

Toggle Breakpoint

When you choose the **Toggle Breakpoint** command, a breakpoint for the current trace position is added to the breakpoints control bar. If a breakpoint already exists for this cursor position, it will be removed from the breakpoints control bar.

Calls

The **Calls** submenu provides you with a list (history) of the most recently called Natural objects including copycodes and inline subroutines. Up to 20 objects can be listed; the most recently called object appears at the top of the list.

The objects list consists of the following information:

- The program level of the called object without counting copycodes and inline subroutines.
- The program level of the called object counting copycodes and inline subroutines.
- The name of the called object.
- The type of the called object.
- The event and control handle of the event handler to be processed (with event-driven applications only).

The status bar at the bottom of the debugger's main window displays additional information on the called object:

- The name of the calling object:

"Natural" is displayed as the calling object if the called object is the application start-up program or a program activated from the Natural stack (including error transaction programs and programs activated by a RUN statement from inside the application).

- The source code line in which the object was called:

If you select an object from the list, except with "Natural", the source of the calling program is displayed in the middle of the source window with the cursor placed at the beginning of the line in which the call occurred.

Variables Menu

The **Variables** menu is used to:

- Display the contents of selected variables.
- Modify the contents of selected variables.
- Quick watch the contents of the variable at the current trace position.
- Add variables to the watchvariables control bar.
- Add variables to the watchpoints control bar.

Dialog Boxes

When you choose the **Display**, **Modify**, **Add Watch** or **Add Watchpoint** command, a dialog box appears, which displays a list of all local, global, AIV or system variables active in the current debugging context. The following controls are part of this dialog box:

- The **Variable** text box, which shows the currently selected variable.
- The **Line Reference** or **Context ID** box, which shows the source code line number of the variable or context variable currently contained in the **Variable** text box.

The **Line Reference** box is only displayed if the line reference is needed to make the variable selection unambiguous. This is the case if:

- the variable belongs to a map; then the box contains the source code line number of the corresponding `RULEVAR` syntax element generated by the map editor;
- the variable is either a database variable (reporting mode only) or one of the following variables: `*ISN`, `*COUNTER`, `*NUMBER`; then the box contains the source code line number of the corresponding database loop or access statement;
- the variable is defined in reporting mode, but without a `DEFINE DATA` statement.

The **Context ID** box is only displayed if the variable is a context variable; then the box contains the “ctx-Id” (context ID).

- The **History List** list box (**Display** command only), which contains the most recently selected variables (up to 20) using a first-in first-out mechanism.

The history list helps you to quickly locate a variable that has been already selected before. Variables can be selected from the history list in the same way as from the variable list.

- The **Variable** list box, which contains the corresponding variable listing.

Selecting Variables

When you choose a variable in the variable list of a dialog box, it is shown in the corresponding **Variable** text box.

When you choose a variable in the variable list of a dialog box, a further dialog box is displayed (except with the **Watch** command).

When you choose an array or variable group:

- the individual array or group elements are displayed in the second dialog box (display),
- the array is displayed in the second dialog box for modification; groups cannot be modified (modify),
- a corresponding error message is displayed (watchpoint).

You can also choose a variable by first marking it directly in the source window and then select the **Display**, **Modify**, **Add Watch** or **Add Watchpoint** command respectively. Then, the **Variable** text box of the corresponding dialog box exactly shows the piece of source code you have marked, which then can be modified.

Marking Text in the Source Window

In the source window, you can mark variables or character strings for selection with either the mouse or the keyboard.

When marking text using the mouse, place the mouse pointer on the first character to be selected, drag the pointer to the last character you want to select, and release the mouse button. To cancel a selection, choose anywhere in the document.

When using the keyboard to mark text, cursor movement keys are used. First place the cursor on a character by using an arrow key, then press and hold down the **SHIFT** key and use the following keys for text selection:

- the **LEFT-ARROW** key to mark the area to the left of your cursor position,
- the **RIGHT-ARROW** key to mark the area to the right of your cursor position,
- the **END** key to mark the area until the end of the source code line,
- the **HOME** key to mark the area until the beginning of the source code line.

Display

With this command, a variable can be selected from the listing in the dialog box for display along with its current content in a second **Display Variable** dialog box, where you can choose between alphanumeric and binary representation of the variable value.

When you select an array, a handle variable or a group of variables, the individual elements and their values are listed in the second dialog box. With arrays, any variable index expression is evaluated.

The element listing can be expanded or contracted by choosing the **Expand/Contract** button. Whenever the number of arrays, groups or dialog element on the list exceeds a certain display limit, a "More" line appears, which can be used to display further objects. Alternatively, the **Expand** command can also be used.

A variable, array or group of variables can also be selected for display in the **Display Variable** dialog box by choosing it with the left mouse button directly in the source window.

Modify

With this command, a variable can be selected from the listing in the dialog box for display together with its current value in a second **Modify Variable** dialog box, where its value can be modified.

If you want to modify a system variable, only system variables which can be modified are displayed in the first dialog box.

If you want to modify an array, only its name but no values are displayed in the second dialog box. The value you enter will then be valid for all array elements.

Groups of variables cannot be selected for modification.

Quick Watch

With this command, a dialog box appears displaying the contents of the variable at the current cursor position.

Add Watch

With this command, variables, arrays or groups of variables can be selected from the list in the dialog box in order to add them to the watchvariables control bar.

Add Watchpoint

With this command, single variables and individual group or array elements can be selected from the listing in the dialog box for the definition of a watchpoint in a second **Set Watchpoint** dialog box; arrays and groups of variables cannot be selected.

The second **Set Watchpoint dialog box** displays the name of the watchpoint (which corresponds to the name of the selected variable) together with its line reference (if applicable), and the names of the corresponding Natural object and library.



Note: With system variables, the corresponding watchpoint is not attached to a specific library and object; therefore, the object and library name will always be `SYSTEM`.

To define a watchpoint, you specify the following items in the corresponding boxes:

- the state of the watchpoint,
- a condition for the watchpoint to be activated (optional),
- the number of skips before execution of the watchpoint,
- the maximum number of executions of the watchpoint.

File Menu

The following commands of the **File** menu are available in conjunction with the source window:

Open

With the **Open** command you can specify a further source program to be loaded into the source window. The **Open Source** dialog box appears, in which you specify the program name and the appropriate library name if the program is not contained in the current library (default).

You can also select a character string for being placed into the **Open Source** dialog box by **marking** its name in the source window and then choosing the **Open** command.

Close

The **Close** command will close the currently active source window. If the source window you are about to close contains the trace bar, the window will be iconized.

Exit

The **Exit** command will exit the debugger and end the current program execution.

Edit Menu

The following commands of the Edit menu are available in conjunction with the source window:

Find

With the **Find** command, you can search up or down through the active window to locate each occurrence of a specified word or character string.

The **Find** dialog box appears, where you can enter the text to be located in the **Find** text box. In addition, you can turn the **Match Upper/Lower Case** and **Whole Words Only** options on or off.

If found, the first occurrence of the specified text is highlighted (selected), whereas a message lets you know if the text could not be found.

With the **Match Upper/Lower Case** option, you can specify whether the find operation is to look for an exact match (ON) or for the same characters only, regardless of case (OFF).

With the **Whole Words Only** option, you can specify whether the find operation is to look for occurrences that are whole words only, not part of a character string (ON), or for all occurrences of the specified text, whole words and parts of a character string (OFF).

To change the direction of the find, choose the **Up** button to search upwards, to the top of the text, or the **Down** button to search downwards, to the bottom of the text; **Down** is the default.

If the find does not start at the top (or bottom) of the text, and the specified text cannot be found, a dialog appears. You can choose **Yes** to continue the find at the top (or bottom) of the text or **No** to cancel the search.

You can also select a character string to be placed into the **Find** text box by **marking** it directly in the source window and then choosing the **Find** command.

Find Next

With this command, you can repeat the previous find operation and locate the next occurrence of the text specified with the **Find** command.

Watchvariables Control Bar

The watchvariables control bar is primarily intended to display previously selected variables for closer and permanent observation of their content.

It offers a context menu which either displays the commands which can be used in combination with the entire control bar or displays the commands which can be used with each individual watchvariable.

To open the context menu, choose with the right mouse button either on the control bars caption or on a particular watchvariable.

Variables Control Bar

The variables control bar displays all variables which are available at current state of the program execution. All variables are grouped in different categories. These categories are **Locals**, **Globals**, **Systems**, **AIVs** and **Contexts**. You can switch between these categories by choosing the corresponding tab at the bottom of the control bar. In order to modify the content, select the content field of a particular variable. Some system variables are read-only and therefore cannot be modified.

The Variables control bar offers a context menu which either displays the commands which can be used in combination with the entire control bar or displays the commands which can be used with each individual variable.

To open the context menu, choose with the right mouse button on either the control bars caption or on a particular variable.

Watchpoints and Breakpoints Control Bar

The Watchpoints and Breakpoints control bar is used to add and maintain watchpoints and breakpoints. You can switch between the watchpoints and breakpoint by choosing the corresponding tab at the bottom of the control bar.

Watchpoints

Using watchpoints, you can rapidly detect “illegal” alterations to Natural variables by objects that contain errors.

By default, watchpoints are used to instruct the debugger to interrupt the execution of Natural objects when the contents of a variable change. However, by specifying a certain value to the variable together with a watchpoint operator when setting a watchpoint, a condition can be set which only activates the watchpoint when condition becomes true.

A variable is considered to have changed either when its current value differs from the value recorded when the watchpoint was last triggered or when it differs from the initial value.

In order to deactivate a watchpoint temporarily, remove the check mark from the check box of the corresponding watchpoint entry.

The watchpoint tab of this control bar offers a context menu which either displays the commands which can be used in combination with the entire tab or displays the commands which can be used with each individual watchpoint.

To open the context menu, choose with the right mouse button on either the tabs caption or on a particular watchpoint.

Add Watchpoint

A new watchpoint can be added either by selecting the **Add Watchpoint** command from the variables menu or by selecting the command **Add** from the context menu of the watchvariables tab.

The **Add Watchpoint** dialog box allows you to select single variables, arrays and individual group elements from the list of available variables. Closing the dialog with the **OK** button will open the **Set Watchpoint** dialog box which allows you specify a condition for this watchpoint.



Note: With system variables, the corresponding watchpoint is not attached to a specific library and object; therefore, the object and library name will always be `SYSTEM`.

Set Watchpoint Dialog Box

The **Set Watchpoint** dialog box displays the name of the watchpoint (which corresponds to the name of the selected variable) together with its line reference/context ID (if applicable) and the names of the corresponding Natural object and library.



Note: With system variables, the corresponding watchpoint is not attached to a specific library and object; therefore, the object and library name will always be `SYSTEM`.

To define a watchpoint, you can specify the following items in the corresponding boxes:

- The state of the watchpoint to be set; valid states are "active" (default) and "pending".
- A condition for the watchpoint to be activated (optional).

You can specify an appropriate value and watchpoint operator; if no operator and value (that is, condition) is specified, the default setting (MOD) applies (for a description of the individual watchpoint operators, see below).

- The number of skips before execution of the watchpoint if it is not to be executed until the program has run a certain number of times; the default is 0.
- The maximum number of executions of the watchpoint; the default is 0.

A watchpoint will not be set until you either choose the **OK** button or press `ENTER`. If you choose the **Cancel** button or press `ESC`, no watchpoint will be set.

Once a watchpoint has been specified, it remains until you delete it explicitly.

Watchpoint Operators

Watchpoint operators are set via option buttons; the available watchpoint operators are:

Operator	Stands for	Description
MOD	Modification	The watchpoint is activated each time a modification of the variable occurs. Default.
LT	Less Than	The watchpoint is activated only when the current value of the variable is less than the specified value.
LE	Less or Equal	The watchpoint is activated only when the current value of the variable is less than or equal to the specified value.
GT	Greater Than	The watchpoint is activated only when the current value of the variable is greater than the specified value.
GE	Greater or Equal	The watchpoint is activated only when the current value of the variable is greater than or equal to the specified value.
EQ	Equal	The watchpoint is activated only when the current value of the variable is equal to the specified value.

Operator	Stands for	Description
NE	Not Equal	The watchpoint is activated only when the current value of the variable is not equal to the specified value.

Breakpoints

A breakpoint is a point at which control is returned to the user while a Natural object is executing.

In order to deactivate a breakpoint temporarily, remove the check mark from the check box of the corresponding breakpoint entry.

The breakpoint tab of this control bar offers a context menu which either displays the commands which can be used in combination with the entire tab or displays the commands which can be used with each individual breakpoint.

To open the context menu, choose with the right mouse button on either the tabs caption or on a particular breakpoint.

Add Breakpoint

With the **Add** command, you can define a new breakpoint. The **Add Breakpoint** dialog box is displayed, where you define the breakpoint by specifying the following items in the corresponding boxes:

- The state of the breakpoint to be set; valid states are "active" (default) and "pending".
- The name of the Natural object to contain the breakpoint; the default object name is the name of the object currently in the source window.
- The name of the Natural library that contains the object with the breakpoint; the default library name is the name of the library which contains the object currently in the source window.
- The line number of the object's source code where the breakpoint is to be executed.

Begin means that the breakpoint is to be set at the first executable line of code of the specified object; **End** means that the breakpoint is to be set at the last executable line of code of the specified object.

- The number of skips before execution of the breakpoint if it is not to be executed until the program has run a certain number of times; the default is 0.
- The maximum number of executions of the breakpoint; the default is 0.

A breakpoint will not be set until you either choose the **OK** button or press ENTER. If you choose the **Cancel** button or press ESC, no breakpoint will be set.

Breakpoints can also be set directly in the program currently contained in the source window by double-clicking the appropriate statement line with the right mouse button. This way, a breakpoint

is defined with all default values and the corresponding source code line number. It can be displayed and/or modified by using the corresponding functions.

Breakpoints cannot be set on comment lines or on any statement line other than the first one if a single statement occupies more than one line.

Once a breakpoint has been defined, it remains until you delete it explicitly.

