

## **Natural for UNIX**

### **Debugger**

Version 6.3.8 for UNIX

February 2010

This document applies to Natural Version 6.3.8 for UNIX.

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

Copyright © 1992-2010 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, United States of America, and/or their licensors.

The name Software AG, webMethods and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

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).

## Table of Contents



1 Debugger .....	1
2 General Information .....	3
Remote Debugging .....	4
3 Using the Debugger .....	9
Preparing Natural Objects .....	10
Starting the Debugger .....	10
Leaving the Debugger .....	11
Operating the Debugger .....	12
Debugger Source Window .....	14
Watchvariables Control Bar .....	21
Variables Control Bar .....	21
Watchpoints and Breakpoints Control Bar .....	22



# 1 Debugger

---

This documentation explains how to debug Natural applications. It is organized under the following headings:

	<b>General Information</b>	Information on remote debugging and how to set up your environment for remote debugging.
	<b>Using the Debugger</b>	How to start and use the debugger.



# 2 General Information

---

- Remote Debugging ..... 4

## 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.

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

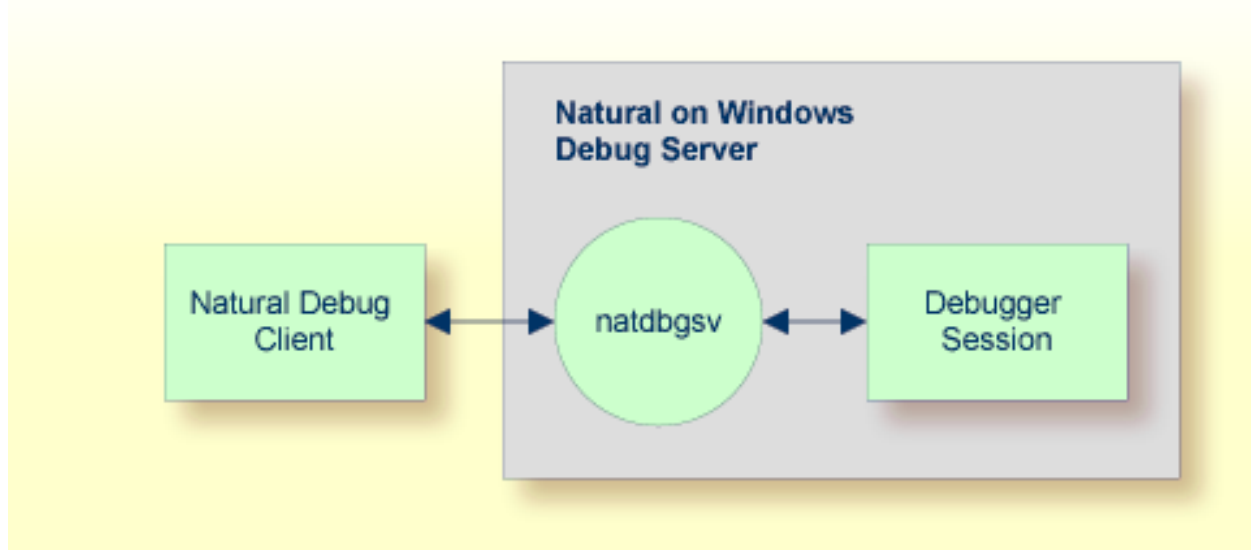
There are different scenarios of how you can use remote debugging: A single Natural client runs under the control of one remote debugging session or a distributed Natural application runs under the control of several remote debugging sessions. Such a distributed application may include both Natural RPC 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

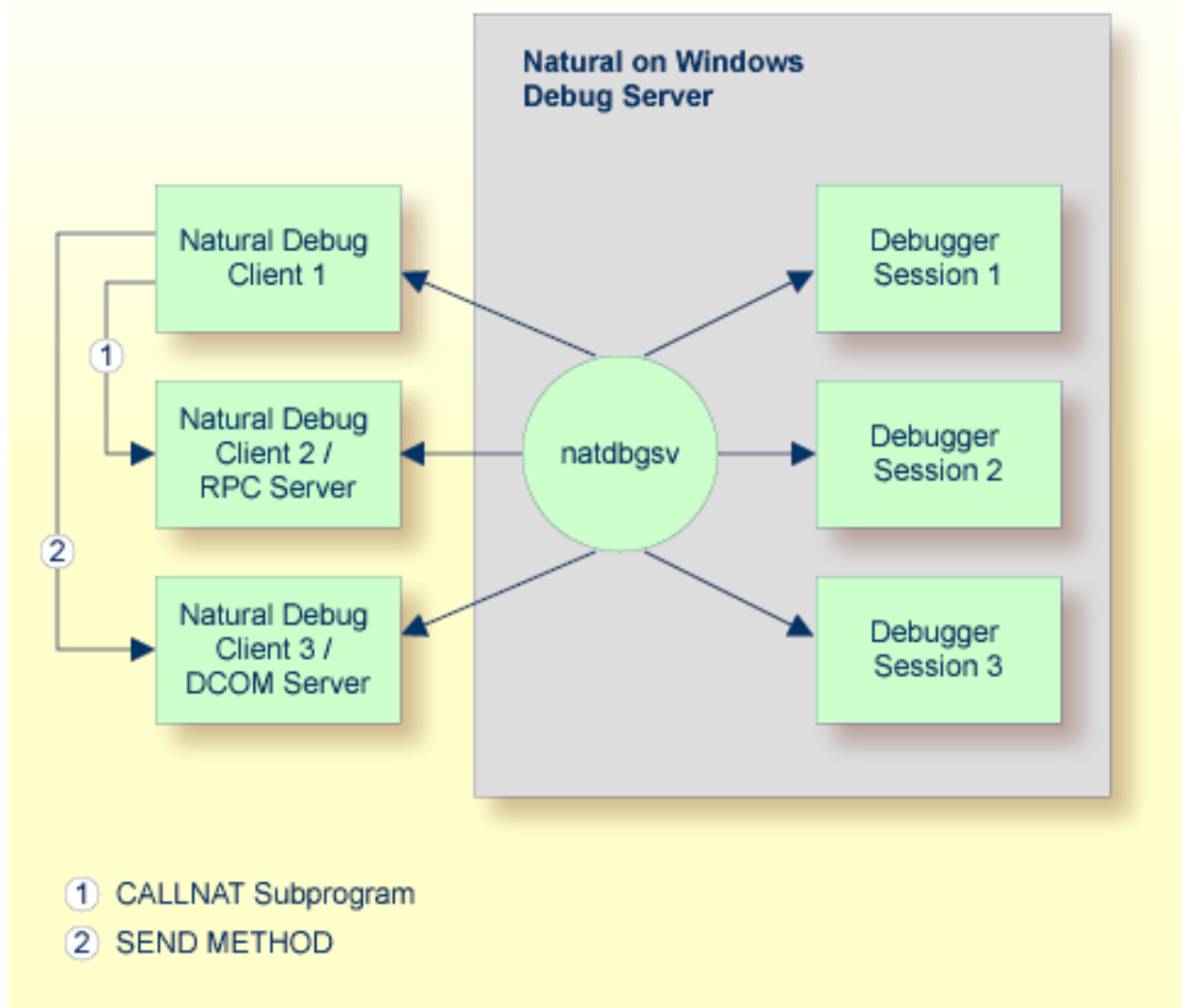
The diagram below illustrates debugging in a single Natural application.



## Scenario 2: Debugging a Distributed Natural Application

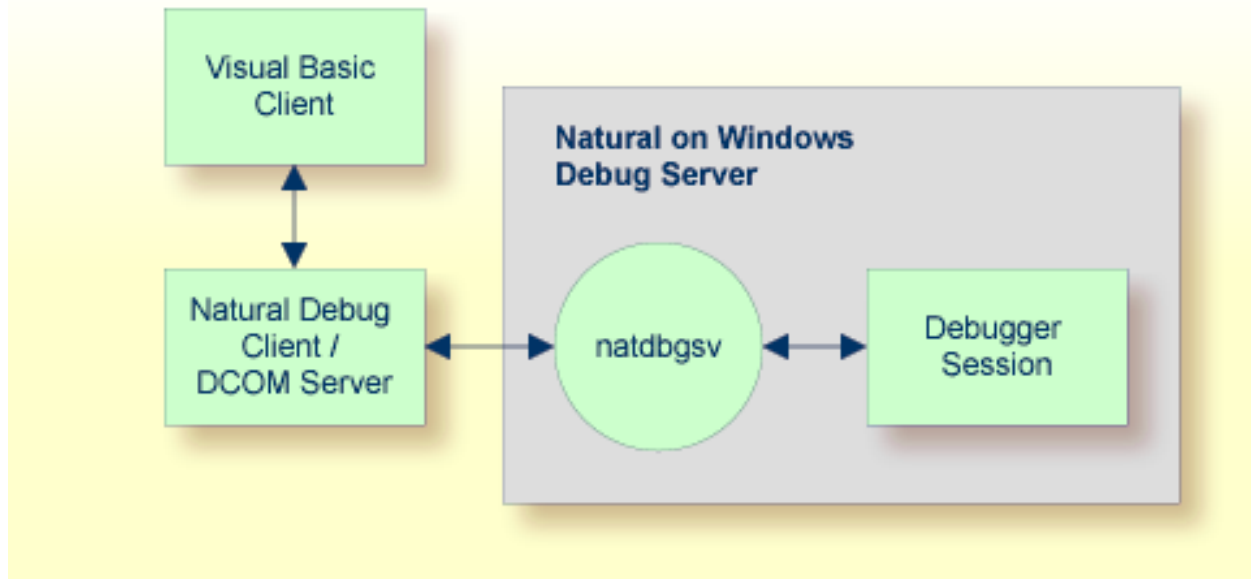
To debug each component of the following distributed Natural application, you enter `DEBUG objectname` in the command line of Natural Debug client 1. The first time the Natural Debug Client calls a subprogram on a Natural RPC server, a new debug session is opened for the RPC server. Then, the RPC server's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



# 3 Using the Debugger

---

- Preparing Natural Objects ..... 10
- Starting the Debugger ..... 10
- Leaving the Debugger ..... 11
- Operating the Debugger ..... 12
- Debugger Source Window ..... 14
- Watchvariables Control Bar ..... 21
- Variables Control Bar ..... 21
- Watchpoints and Breakpoints Control Bar ..... 22

## 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.

