Natural

Debugger

Version 8.3.8

January 2017

ADABAS & NATURAL

This document applies to Natural Version 8.3.8.

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

Document ID: NATWIN-NNATDEBUG-838-20170102

Table of Contents

Preface

This documentation, which is complemental 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.
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.

1 General Information

About the Debugger

The debugger is integrated in Natural Studio. The complete Natural Studio functionality can 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 local environment. To be able to debug Natural applications, it is required to set the Natural parameter `SYMGEN` to "ON" before you catalog the Natural application (see also ***Preparing to Use the Debugger***). 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.

Remote Debugging

Remote debugging is no longer supported. Instead, you will have to use the debugger of NaturalONE. Using the NaturalONE debugger, it is possible, for example, to debug Natural RPC applications or to debug workplace applications which have been created with Natural for Ajax.

2 Starting and Leaving the Debugger

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 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

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**

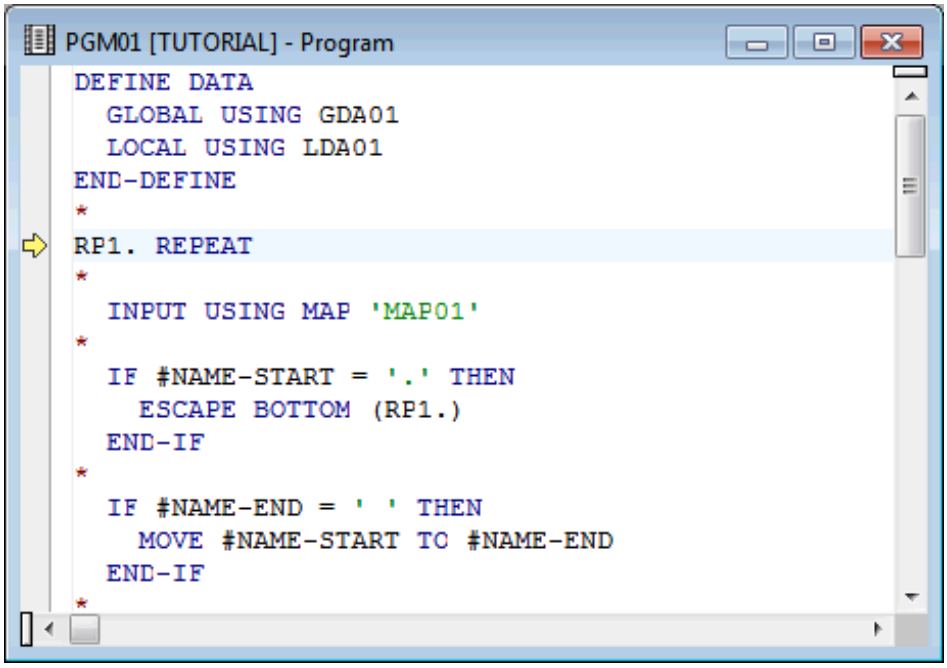
 **Modify Variable**

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

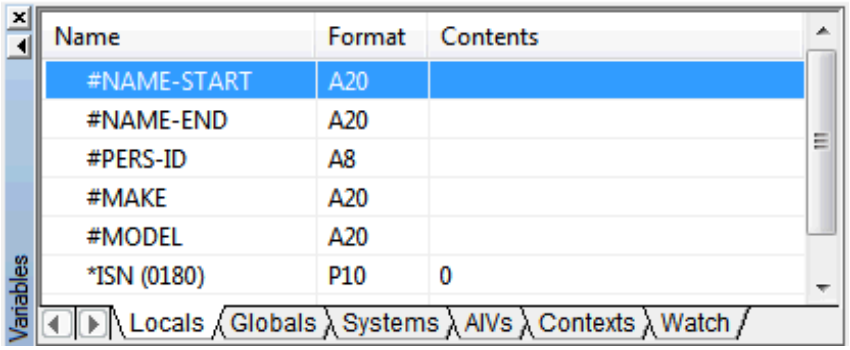
Each tab of a debugger window offers a context menu which contains either the commands which can be used in combination with the entire tab (when an entry is not selected) or the commands which can be used with the selected entry. These commands are described later in this documentation.

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: +R).

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.

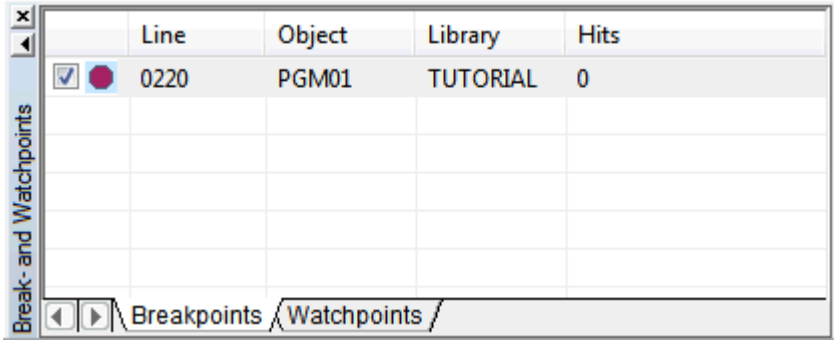
  When the variables window is displayed in the Natural Studio window, a check mark is shown next to this menu command.

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.

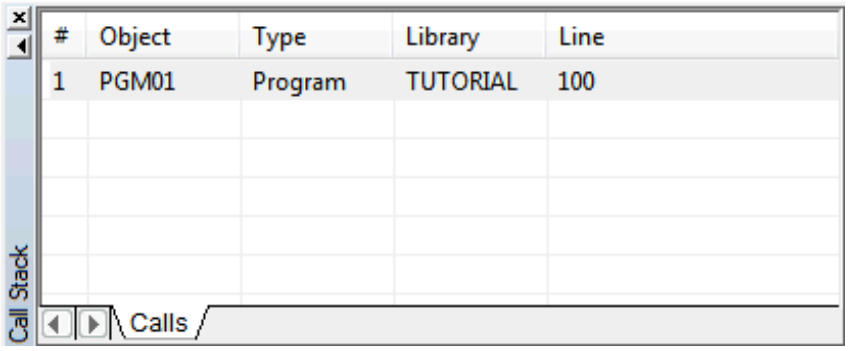
  When the break- and watchpoints window is displayed in the Natural Studio window, a check mark is shown next to this menu command.

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.



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

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

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.

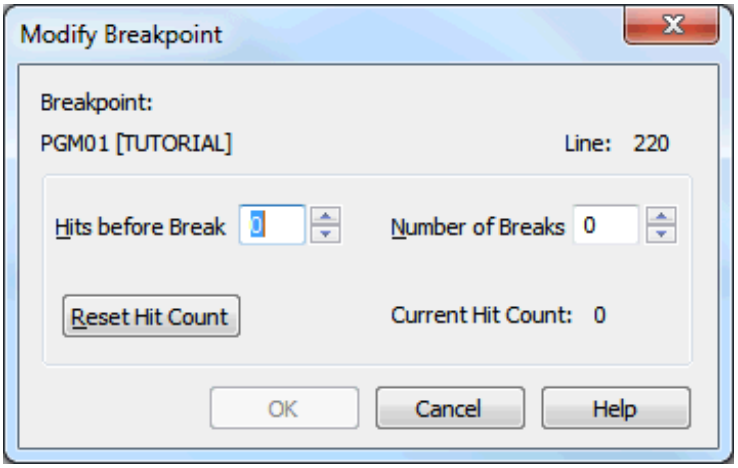
Modifying a Breakpoint

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.

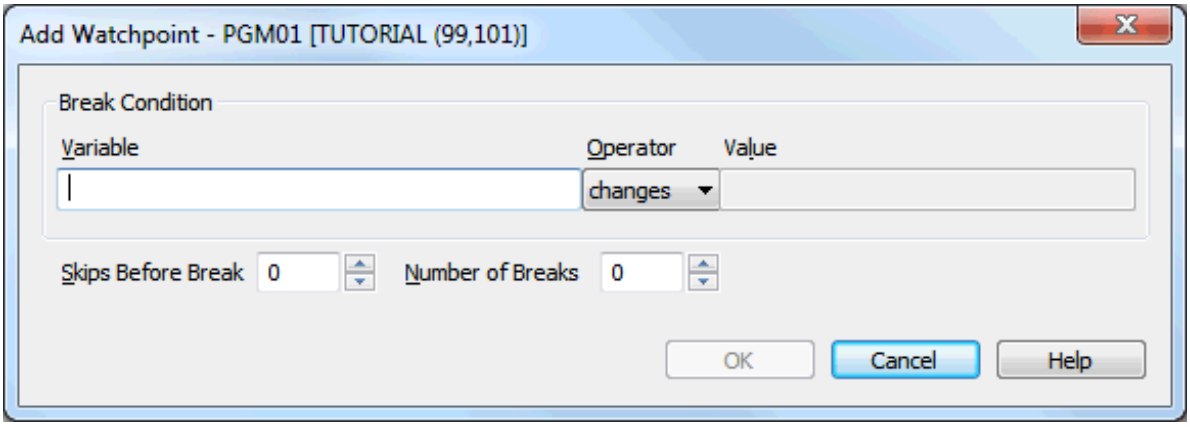
To add a watchpoint

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.

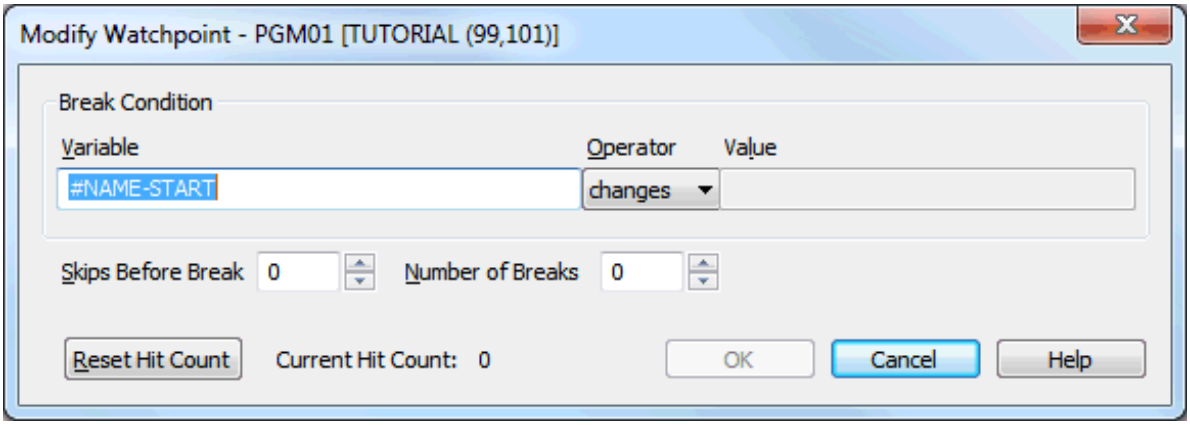
Modifying a Watchpoint

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

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**.

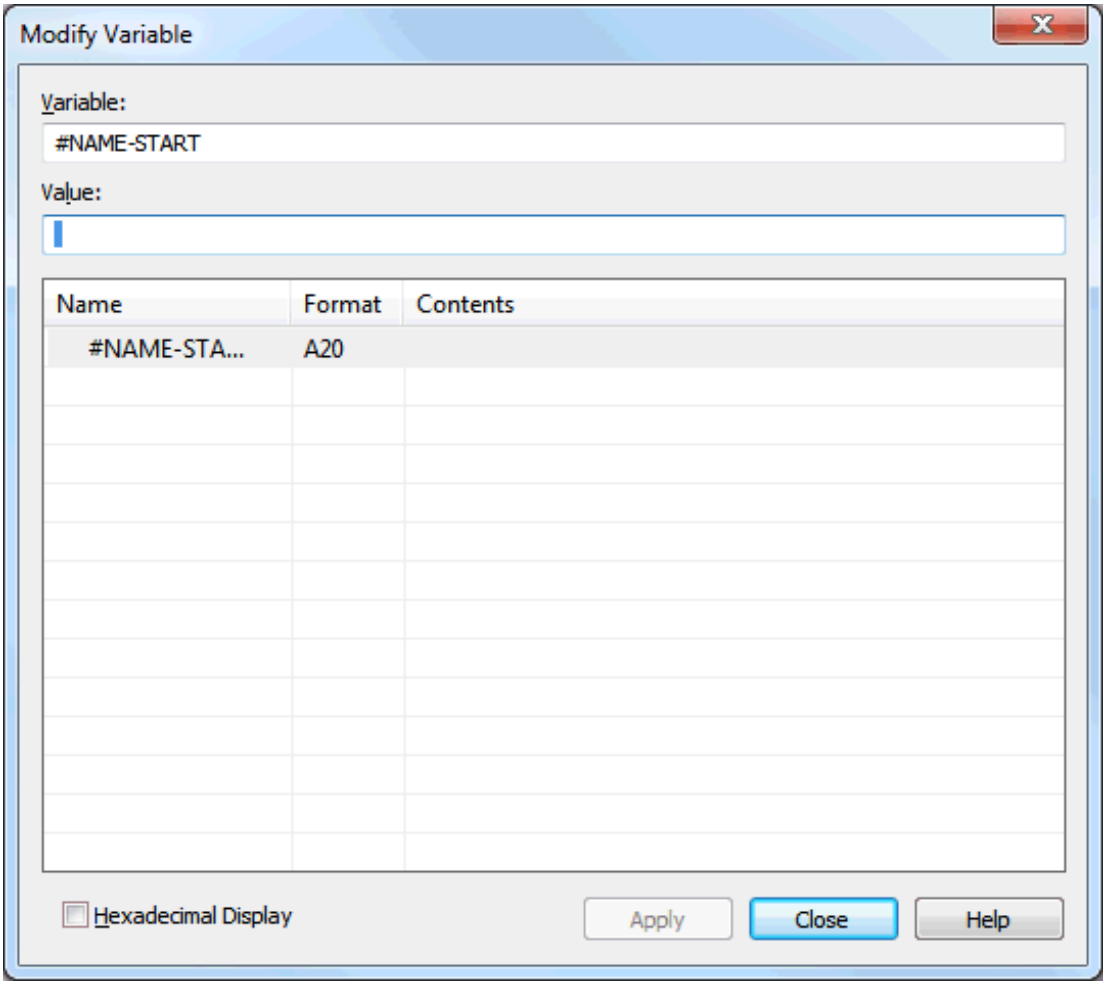
Or:

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.



3 To modify the content of the variable, enter the new value in the **Value** text box.

Note: For variables which cannot be modified (such as unmodifiable system variables) the **Value** text box is dimmed.

You can also use the **Modify Variable** dialog box in the following ways:

- When you enter the name of another existing variable in the **Variable** text box, the content of this variable is immediately shown in the dialog box and you can modify it.
- In the case of an array, you can modify the occurrences of this array as follows:
  - The same value can be defined for all occurrences: Select the top-level node and enter the new value.
  - Each occurrence can be modified separately: Select the required occurrence and enter the new value.
  - Specific occurrences can be modified at the same time: Specify the required occurrences in the **Variable** text box. For example, when you change `#MYVAR(1:6)` to `#MYVAR(2:3)`,

only the second and third occurrence is shown in the dialog box. When you enter a new value, it applies only to these occurrences.

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:

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 an 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

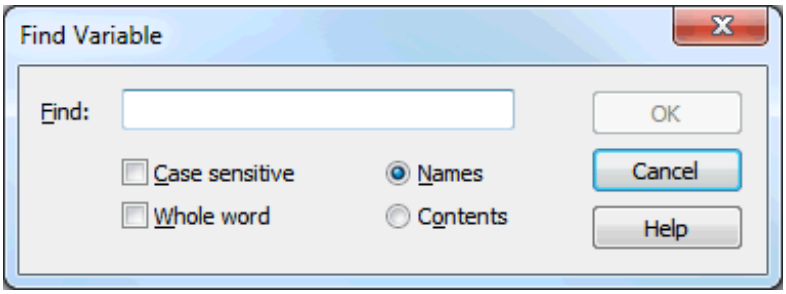
1 Select the required tab in the variables window.

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

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.