

Concepts of the Natural Debugger

The Natural Debugger can be used to temporarily take over control of a Natural session for debugging purposes while a Natural program is executing.

The execution flow of an application is not influenced by the Natural Debugger being applied to it; that is, the application itself need not be adapted to or prepared for its being debugged. By setting debug entries in a program, you can follow the processing flow of the program. The Natural Debugger receives control at any debug entry set, thus allowing various program investigations. This helps you to understand poorly written or poorly documented programs, and to identify redundant or dead (never gets executed) program code. This can be useful, for example, when you take over an application from another developer to maintain, improve or expand it.

In addition, when you write an interface to a program, you can use the Natural Debugger to check the contents of variables at the point when parameters are passed between the program and the interface and thus make sure that parameters are passed correctly.

When Test Mode is set to ON (see Set Test Mode ON/OFF), the Natural Debugger receives control whenever a Natural error occurs during the execution of a program, irrespective of any debug entries defined. You can then, for example, review the contents of the variables in the program to determine the reason for the error.

Below is information on:

- Debug Entries/Spies
 - Debug Window
-

Debug Entries/Spies

Debug entries are also referred to as spies in the Natural Debugger environment. Two types of debug entries (spies) are available: breakpoints and watchpoints. Debug entries for the current debug session can be set, modified, listed, displayed, activated, deactivated and deleted. Debug entries can also be saved for future use as described in Debug Environment Maintenance.

When a debug entry is set or modified, Natural internally stores the library, database ID and file number where the object is located. The object may be located in the current library or in one of its steplibs. If an object of the same name is later executed from another library, the corresponding debug entry is not executed.

The Natural Debugger assigns a name and a unique number (Spy Number) to each debug entry. The name assigned to a debug entry (also referred to as Spy Name) can be either a name specified by the user or a default name created by the Natural Debugger. A debug entry can be selected by its number with the corresponding Natural Debugger commands. If more than one debug entry has to be executed at a specific statement line, they are executed in ascending order of their numbers.

Each debug entry has an initial state and a current state. Possible values are **A** (active) and **I** (inactive). The initial value is specified when setting or modifying the breakpoint or watchpoint and determines the state of the debug entry at environment start or after reset. During the debug session, the state can be changed with the debug commands **ACTIVATE** and **DEACTIVATE** (see also the syntax diagrams in Command Summary and Syntax).

Each debug entry has an event count, which is increased every time the debug entry is executed. A debug entry is not executed if the current state is "inactive". The event count of the breakpoint or watchpoint is not increased either.

The number of executions of a debug entry can be restricted in two ways:

- A number of skips can be specified before the debug entry 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 debug entry is ignored, as soon as the event

count exceeds the specified number of executions.

For each debug entry (breakpoint or watchpoint), up to six debug commands can be specified. These commands are executed at execution time of the breakpoint or watchpoint. You can use all Natural Debugger commands that can be applied during a debug interrupt. The default command is the BREAK command, which displays the Debug Window, as shown below.

Attention:

If you delete the BREAK command when setting a debug entry and you do not enter any command that issues a dialog, there is no way to assume control during program interruption.

Debug Window

When the Natural Debugger receives control of the session, the Debug Window is displayed.

The Debug Window shows the type and name of the debug entry that has caused the break (that is, the name of the corresponding breakpoint or watchpoint), its source-code line number, and the name of the interrupted object. For a detailed description of the functions listed below see Execution Control Commands.

From the Debug Window, you can select the following functions:

Function	Code	Description
Go	G	Continues the execution of the program up to the next debug entry specified.
List Break	L	Lists the program code currently active. The last statement executed is highlighted.
Debug Main Menu	M	Invokes the Debug Main Menu which provides all functions needed to maintain debug entries at which control is to be assumed.
Next	N	Executes the next command specified for the current breakpoint or watchpoint.
Run	R	Continues the program with test mode OFF.
Step	S	Continues the program in step mode.
Variable Function	V	Displays the program variables currently active and modifies the contents of these variables.