XDebug Tutorial

Example Session

Instructions File: Makefile File: galaxy.h File: collapse.c File: create.c File: form.c File: main.c

Images **Loading an Application** **Process Page** **Source Code** **Stack Page** **Message Queue** **Message Detail** **Data Display** **Command Line** **Find Dialog** **Help Index** **Source Path Dialog**

XDebug Tutorial: Introduction

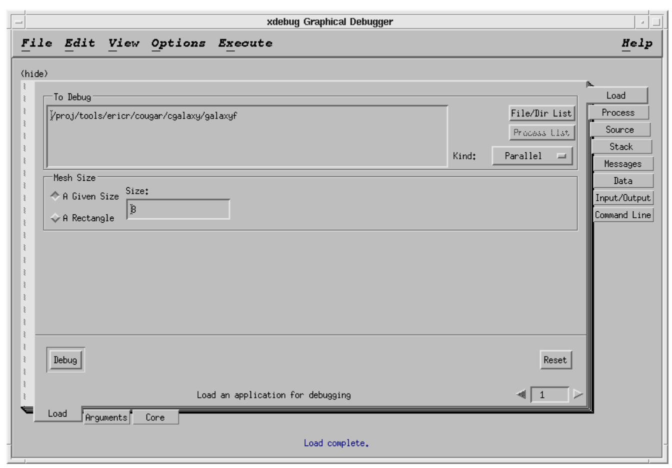
XDebug is a graphical debugger for parallel applications running on the Teraflops system.

- Point and click debugging.
- Renders application state, stack, message queue.
- Designed for occasional user.

XDebug Tutorial: Basic GUI Layout

The XDebug GUI is based on a notebook. Each significant aspect of debugging is contained within a page.

- Menus: File, Edit, View, Options, Execute, Help.
- Notebook pages for major debugging activities.
- Pages can be torn out and later returned by clicking on dashed "perforation."
- On-line help system describes features.
- Dialogs for finding source, searching text, and advanced action points.
- Goal is spartan design.



XDebug Tutorial: How to Load an Executable

The first panel displayed in XDebug is the load panel.

- XDebug invocation can be used, like yod, for fast loading.
- Enter debug target, choose if parallel (default) or not.
- Size selection.
- Sub-page for arguments and yod specific attributes.
- Select the **Debug** button.

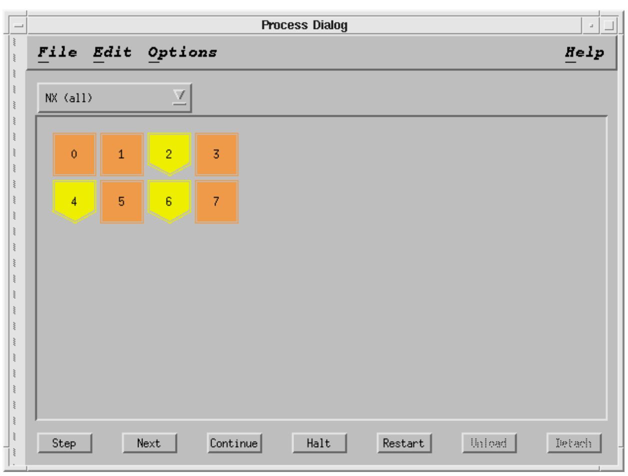
Note:

Source location can be stated via **Source Path** dialog.

XDebug Tutorial: How to View Process State and Set Context

The process panel arranges icons for each process that represent the state of the process and whether the process is in or out of context.

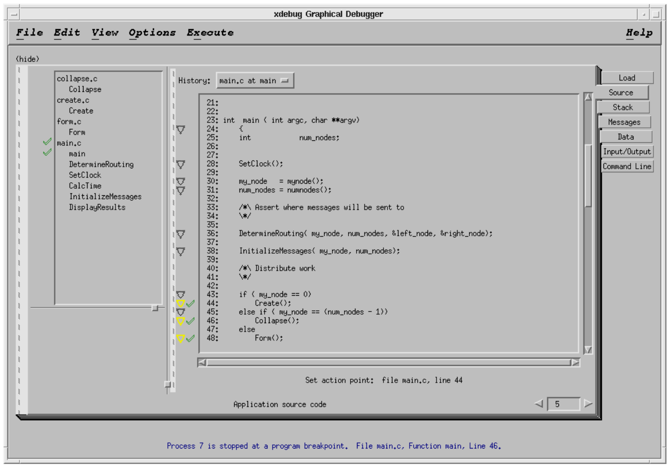
- Process panel depicts state and context.
- Clicking on process toggles context state in/out.
- Edit menu's **Select** items speed context selection.
- Moving cursor over process shows state and location.



XDebug Tutorial: How to View Source Code

The source page contains both a listing of all files and routines in the application and a text area for a file's source.

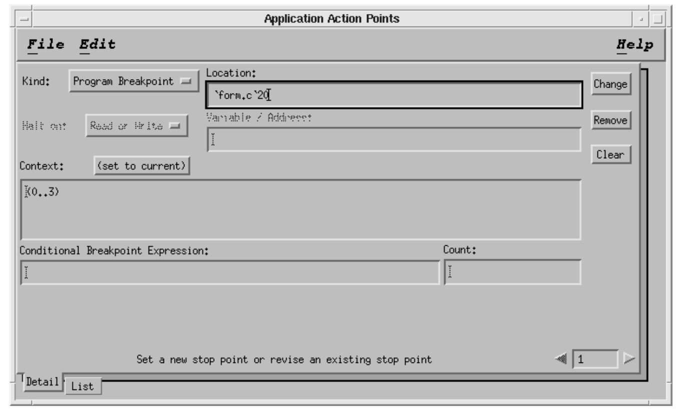
- Source listing, class listing on left side of page.
- Click on file: top of file source displayed.
- Click on routine: top of routine source displayed.
- Executable lines indicated via dimmed break point icon next to source lines.
- Active routines and lines indicated by check mark.
- Button-2 click over source code text will *dive* into the text (display routine, variable's value, type info...).

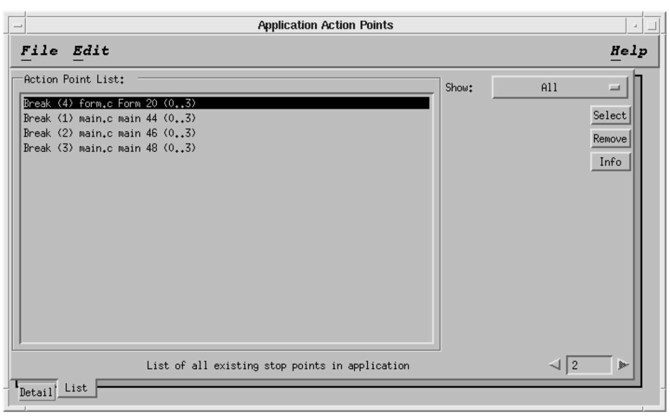


XDebug Tutorial: How to Set a Breakpoint

Break points and trace points can be placed via the source page by clicking next to the appropriate executable line.

- Dimmed break point icons appear next to executable lines.
- Click on line to toggle set / not-set.
- Switch between break and trace via Options menu.
- See action list dialog for setting with count or conditional expression or for setting watchpoints.

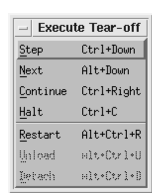




XDebug Tutorial: How to Control Execution

Processes within the current context, depending on their state, can be stepped into source, stepped over a line, started into continuous execution, halted, or restarted.

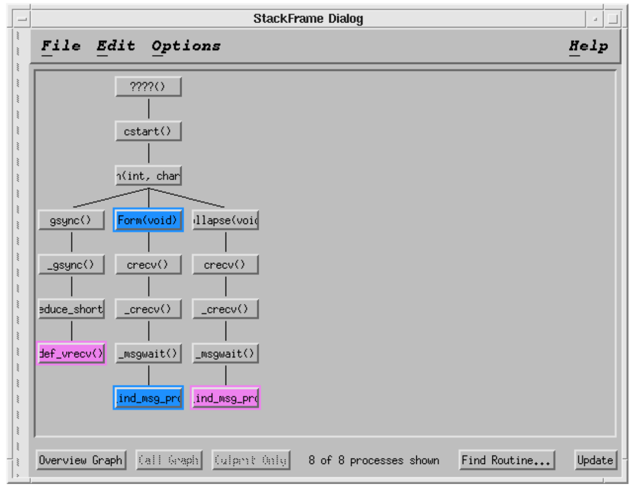
- Process panel buttons along bottom.
- Execute menu (can be torn off).
- Controls: Step, Next, Continue, Halt, Restart.
- Controls apply to current context.
- Menu accelerator keys useful in this case.



XDebug Tutorial: How to Examine Stack

The stack frame page provides a graphical view of the call stack for the processes within context.

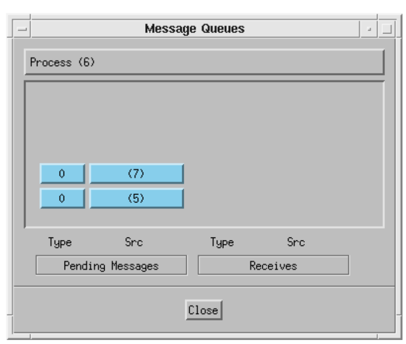
- Based on light-weight core file browser Ptools project.
- Stack panel: select **Update** button.
- Tree view for all processes in context.
- Views: overview, call graph, culprit only.
- Tree can be displayed as vertical or horizontal.
- Clicking on tree components brings up information.



XDebug Tutorial: How to Examine Message Queues

The message queue page allows an overview of what processes in the context have messages pending.

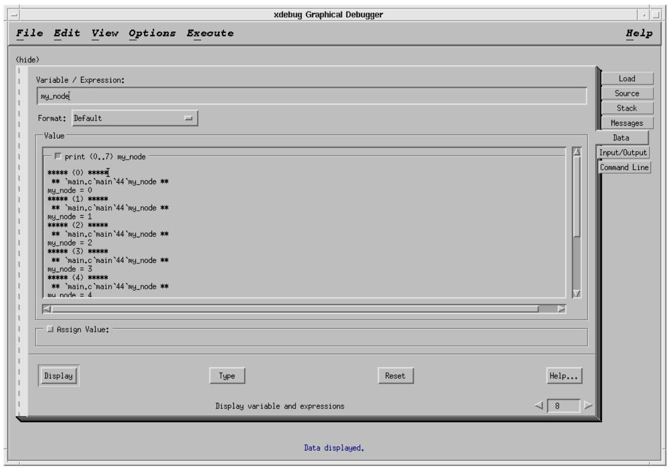
- Based on message queue manager Ptools project.
- Graphical encoding of messages sent/pending for context.
- Move over square to get process number.
- Click on square to get more detail about messages to or from the process.



XDebug Tutorial: How to Examine Data

The data display page is used to examine the value of a variable or expression.

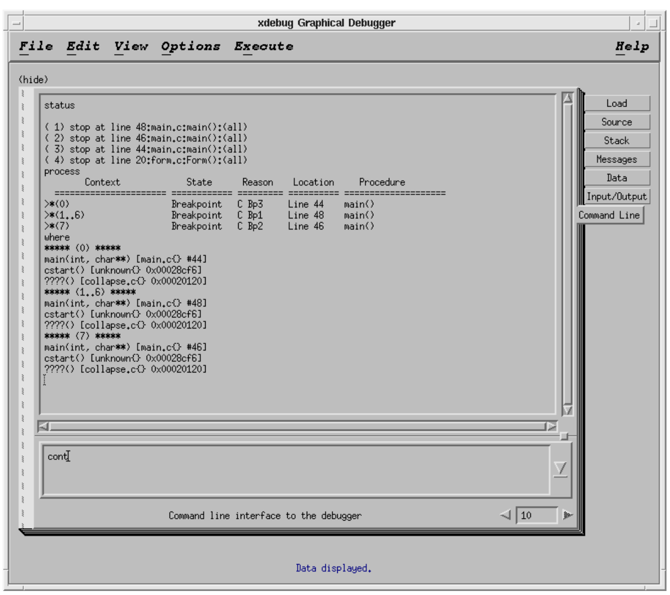
- Enter expression in data panel.
- Output appears at top of scrolling region.
- Open **Assign Value** area to change value of variable.
- Type and Communicator information is also displayed here.



XDebug Tutorial: How to Enter 'debug' Commands Directly

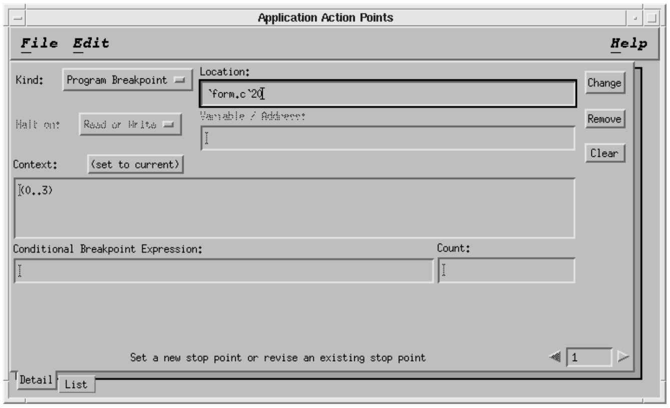
The command line panel provides direct access to the text debugger for users who know the debugger's command line syntax.

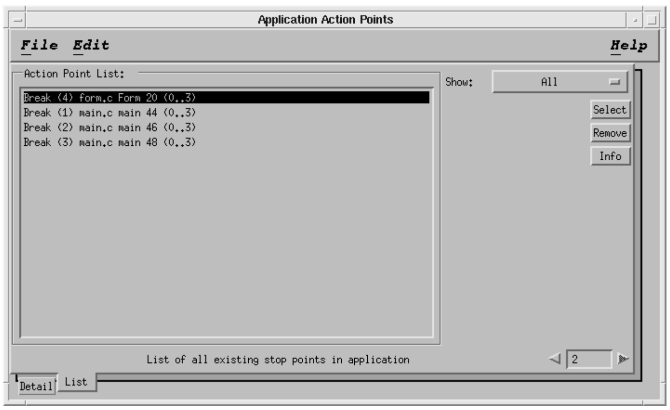
- Command line panel accepts debug syntax in input area.
- Output area displays output of command.
- Command history kept in pull down menu.
- All commands supported except *wait*.
- XDebug notices what commands are executed and updates the interface in response.



The action point dialog provides more options for placing action points. It also allows manipulation of all user code action points.

- Use the Edit menu to display the action point dialog.
- Use to set action points with counts or conditionals.
- Use to set data break points on expressions.
- Action point list sub-panel displays all action points placed in user code.
- List can be used to manipulate groups of action points.

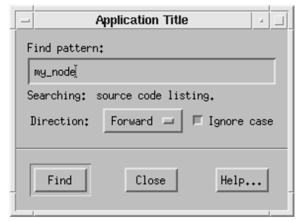




XDebug Tutorial: How to Search for Text

The **Find** utility allows text searching within scrollable lists and scrollable text.

- Give focus (click within) to text or list that you want to search.
- Bring up *Find* dialog.
- Enter text (can be egrep(1) expression).
- *Find* begins search from current point. Match is highlighted.
- *Find Again* resumes search.



XDebug Tutorial: Instructions for Example

This example session will introduce you to various parts of *xdebug*.

You must first collect the example source files and Makefile and build the executable `galaxyf`. This example code is a small demonstration program that does NX message passing.

Starting xdebug

- Change your directory so that you are present in the directory where the example source files and executable are.
- Start xdebug on eight nodes:
  `xdebug -sz 8 galaxyf`
- xdebug should come up, pause while loading the processes, and then switch to the process page.
- *Note*: If an error occurred during the load, switch to the input/output page to see the error message.

Notebook design

- The main notebook contains pages of major debugging activity.
- Clicking on a tab, or using the View menu, displays specific pages in the notebook.
- Notebook pages can be torn out into dialogs in case you want to display more than one page at a time.
- Switch to the process page.
- Click on the vertical dashed line.
- Process page now in own dialog that can be moved around.
- Click on the vertical dashed line within the process page dialog to return to the main notebook.

Viewing Source and Setting Breakpoints

- Switch to the source page.
- Click on the routine `main` within the file `main.c`
- Scroll down so that lines 44, 46, and 48 are visible. Should look like:

```
43:     if ( my_node == 0)
44:         Create();
45:     else if ( my_node == (num_nodes - 1))
46:         Collapse();
47:     else
48:         Form();
```

- Click on the triangle next to the three lines so that a breakpoint is set for 44, 46, and 48.
- Pull down the Execute menu and select *Continue*.
- Shortly, three checkmarks should appear next to 44, 46, and 48. This means at least one process is stopped at each of these lines.
- Switch to the process page and verify that each process is indeed at a break point.
- Move cursor over each process to see message summary of process' state and location.

Diving into Source Code Contents

- Switch to the source page.
- Move the pointer over line 44's call to `Create()` and use mouse button number two to click on `Create()`.
- Source code to `Create()` should be displayed.
- Pull down the history menu and select `main.c at main`.
- Move the pointer over the variable `my_node` and click again with mouse button number two.
- Display should switch the the data page and automatically display the value of `my_node` for all the processes in context.

Forcing Message Blocking Situation

- Switch to the source page.
- Select the routine `Form` under file `form.c`

- Set a break point on line 20 by clicking next to the line. Line 19 and 20 should look like:

```
19:         {
20:         crecv( 0, &recvLeft, sizeof( TMessage));
```

- Switch to the process page.
- Select the *Continue* button.
- Note that processes 2, 4, and 6 (for eight process load) have hit the breakpoint while the other processes remain within the *Executing* state.
- Select the *Halt* button.
- The previously *Executing* processes should now be in the *Blocked* state. This means they are stopped within a routine that reflects a blocking receive is in progress.

Viewing the Stack

- Switch to the stack page.
- Select the *Update* button.
- A graphical representation of the call stack should appear.
- Colored nodes represent places where processes are stopped.
- The *Call Graph* button provides names for the call-graph elements.
- Clicking on a colored button presents a dialog with a listing of the processes at that point.
- *Tip*: If you want wider nodes in your stack frame, change the following line in your `.Xdefaults` or `XDebug` resource file:
  `XDebug*CallGraphButton.width: 72`
  to be use a value greater than 72.

Viewing the Message Queue

- Switch to the message queue page.
- Select the *Update* button.
- Information about receives in progress is displayed in a colored grid.
- Select radio button *Messages Pending Receipt*.
- Pending message information is now present, too.

- Moving cursor over a colored box displays process number.
- Clicking on a colored box displays queue information for process.

Exiting XDebug

- Choose *Exit* from the File menu.
- After a brief pause, xdebug's window will vanish.