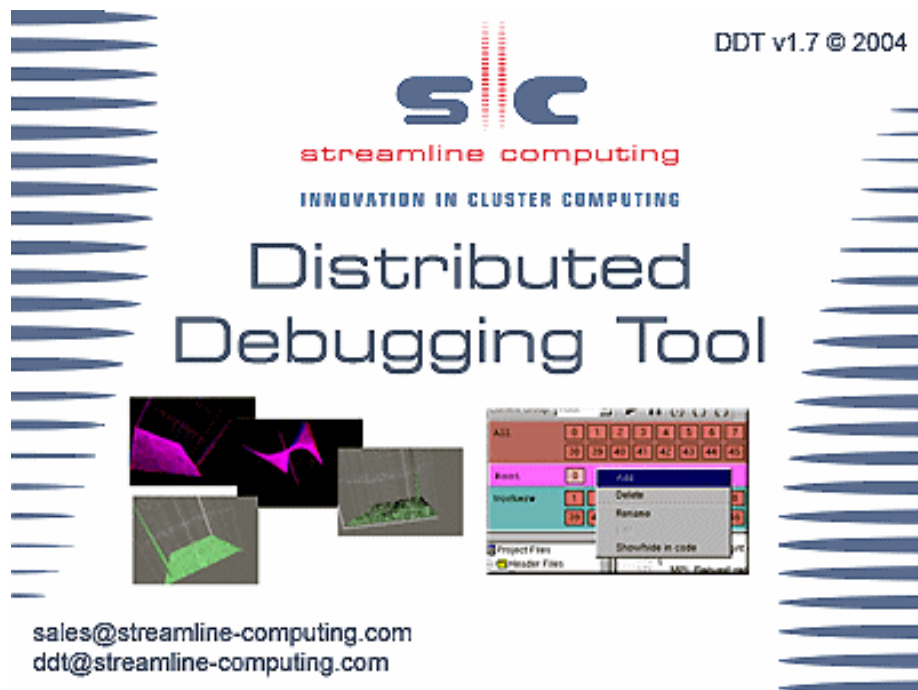


# The Distributed Debugging Tool



## Userguide



# Contents

Contents .....	3
1. Introduction.....	7
2. Installation.....	8
Licence Files.....	8
Floating Licences.....	9
Getting Help.....	10
3. Starting DDT .....	11
Debugging Multi-Process Programs .....	11
Debugging Single-Process Programs .....	13
Debugging A Core File .....	14
Attaching To Running Programs .....	14
Configuring DDT With Queuing Systems .....	17
Starting A Job In A Queue .....	19
Choosing The Right Debugger.....	19
4. DDT Overview .....	21
Saving And Loading Sessions.....	22
Source Code.....	22
Finding Lost Source Files .....	22
Dynamic Libraries.....	23
Finding Code Or Variables .....	23
Jump To Line.....	23
Editing Source Code .....	23
5. Controlling Program Execution.....	24
Process Control And Process Groups .....	24
Hotkeys.....	25
Starting, Stopping And Restarting A Program .....	26
Stepping Through A Program .....	26
Setting Breakpoints .....	26
Conditional Breakpoints .....	27
Suspending Breakpoints .....	27
Deleting A Breakpoint.....	27
Loading And Saving Breakpoints.....	28
Synchronizing Processes .....	28
Setting A Watch.....	28
Examining The Stack Frame.....	29
Align Stacks .....	29
Examining Threads.....	30
6. Variables And Data .....	31
Current Line .....	31

Local Variables .....	31
Arbitrary Expressions And Global Variables .....	32
Changing Data Values .....	32
Examining Pointers .....	32
Examining Multi-Dimensional Arrays.....	33
Visualizing Data .....	34
Cross-Process Comparison .....	35
Viewing Registers .....	36
Interacting Directly With The Debugger .....	36
7. Program Input And Output.....	38
Viewing Standard Output And Error .....	38
Displaying Selected Processes .....	38
Saving Output .....	39
Sending Standard Input (DDT-MP) .....	39
8. Message Queues .....	40
9. The Licence Server .....	42
Running The Server .....	42
Running DDT Clients .....	42
Logging.....	43
Troubleshooting.....	43
Adding A New Licence .....	43
Examples .....	43
Example Of Access Via A Firewall .....	44
Querying Current Licence Server Status .....	45
Licence Server Handling Of Lost DDT Clients .....	46
A. Supported Platforms .....	48
B. Troubleshooting DDT.....	49
Problems Starting DDT Frontend .....	49
Problems Starting Scalar Programs .....	49
Problems Starting Multi-Process Programs .....	50
C. FAQs.....	52
DDT will not load - what's wrong?.....	52
Why can't DDT find my hosts or the executable?.....	52
Why am I getting an error message about the debugger when starting my job? ..	52
The progress bar doesn't move and DDT 'times out'.....	52
The progress bar gets close to half the processes connecting and then stops and DDT 'times out' .....	53
My program runs but I can't see any variables or line number information, why not?.....	<b>Error! Bookmark not defined.</b>
I am using MPI_Get_processor_name() and I get "Process n has stopped with signal SIGTRAP" when I click play.....	53

My program doesn't start, and I can see a console error stating "QServerSocket: failed to bind or listen to the socket" .....	53
Why can't I see any output on stderr? .....	54
DDT complains about being unable to execute malloc.....	54
Why can't I use watches with the Intel IDB interface? .....	54
Why is my stack trace empty/incomplete? .....	54
Some features seem to be missing (e.g. watch points) – what's wrong? .....	54
My code does not appear when I start DDT.....	54
When I use step out my program hangs.....	55
When viewing messages queues after attaching to a process I get a “Cannot find Message Queue DLL” error .....	55
I get the error `The mpi execution environment exited with an error, details follow: Error code: 1 Error Messages: “mprun:mpmd_assemble_rsrcs: Not enough resources available”` when trying to start DDT.....	55
What do I do if I can't see my running processes in the attach window? .....	55
When trying to view my Message Queues using mpich I get no output but also see no errors .....	56
Obtaining Support.....	56
D. Debugging Fortran.....	57
How do I view the contents of an array which is given as a parameter to a subroutine and appears as "PTR TO ..." in the locals window? .....	57
Why is getting local variables so slow?.....	57
Why are there patched gdb's in the DDT distribution? .....	57
E. Notes On MPI Distributions .....	59
Bproc .....	59
HP MPI.....	59
LAM/MPI .....	59
MPICH And SMP Nodes .....	59
MPICH p4 .....	59
MPICH p4 mpd .....	60
MPICH-GM .....	60
IBM PE .....	60
NEC MPI .....	61
Quadratics MPI .....	61
SCore .....	61
Scyld .....	62
SGI Altix/Irix .....	62
F. Notes On Debuggers .....	63
Absoft .....	63
GNU .....	63
Intel Compilers.....	63

Portland Group Compilers .....	64
Solaris DBX.....	64
HP-UX And wdb .....	65
G. Architectures .....	66
AMD Opteron 64-Bit .....	66
SGI Altix 3000 .....	66
IBM AIX Systems .....	66
Index.....	67

# 1. Introduction

The Distributed Debugging Tool is an intuitive, scalable, graphical debugger. This document introduces DDT and explains how to use it to its full potential. If you just wish to get started with DDT, you will find that the examples directory of your DDT installation contains a quick-start example.

DDT can be used as a single-process (non MPI) or a multi-process program debugger. The availability of these capabilities will depend on the licence that you have – although multi-process licences are always capable of supporting single-process debugging.

Both modes of DDT are capable of debugging multiple threads, including OpenMP codes. DDT provides all the standard debugging features (stack trace, breakpoints, watches, view variables, threads etc.) for every thread running as part of your program, or for every process – even if these processes are distributed across a cluster using an MPI implementation.

Multi-process DDT encourages the construction of user-defined groups to manage and apply debugging actions to multiple processes. Once you are comfortable working with groups of processes, everything else becomes simple and intuitive. If you have used a visual debugger before you will find DDT's interface familiar. Using DDT to debug MPI code makes debugging parallel code as easy as debugging serial code.

C, C++, Fortran and Fortran90 are all supported by DDT, along with a large number of platforms, compilers and MPI libraries.

## 2. Installation

DDT is downloadable from the Streamline Computing website. The package is installed by untarring and running the install.sh script.

```
tar xvf ddt1.x-arch.tar
./install.sh
```

For the purposes of the rest of this document, the install directory that you choose will be referred to as \$DDTPATH

During this phase of installation, you will be given a choice of where to install DDT. DDT can be installed by a single user in the user's home or by an administrator in a common directory where file permissions permit.

A number of environment variables are required for DDT to function correctly. Users can set these within their own accounts, or system administrators can provide appropriate universal set up scripts.

It is important to follow the instructions in the README file that is contained in the tar file.

Due to the vast number of different site configurations and MPI distributions that are supported by DDT, it is inevitable that some users will need to take further steps to get DDT working. For example, it may be necessary to ensure that environment variables are propagated to remote nodes, and that DDT's libraries and executables are available on the remote nodes.

### Licence Files

Licence files should be stored as \$DDTPATH/Licence.

If this is inconvenient, the user can specify the location of a licence file using an environment variable, DDT\_LICENCE\_FILE. For example:

```
export DDT_LICENCE_FILE=$DDTPATH/SomeOtherLicence
```

The user also has the choice of using DDT\_LICENSE\_FILE as the environment variable.



The order of precedence when searching for licence files is:

- \$DDT\_LICENCE\_FILE
- \$DDT\_LICENSE\_FILE
- \$DDTPATH/Licence.

If you do not have a licence file, the DDT frontend will not start. A warning message will be presented.

For remote MPI processes, you will also require the licence to be installed on the nodes. If this licence is not present, the remote nodes will be unable to connect to the frontend.

Time-limited evaluation licences are available from the Streamline Computing website.

## Floating Licences

For users with floating licences, the licensing daemon must be started prior to running DDT.

```
$DDTPATH/bin/licenceserver &
```

This will start the daemon, it will serve all floating licences in the current working directory that match Licence\* or License\*.

The hostname, port and MAC address of the licence server will be agreed with you before issuing the licence, and you should ensure that the agreed host and port will be accessible by users.

DDT clients will use a separate client licence file which identifies the host, port, and licence number.

Log files can be generated for accounting purposes.

For more information on the Licence Server please see section 9 of this document.

## Getting Help

In the event of difficulties – in either installing or using DDT – please consult the appendix to this document or the support and software updates section of our website. This document is also available from within DDT by pressing F1.

Support is also available from the support team – [ddt@streamline-computing.com](mailto:ddt@streamline-computing.com)

### 3. Starting DDT

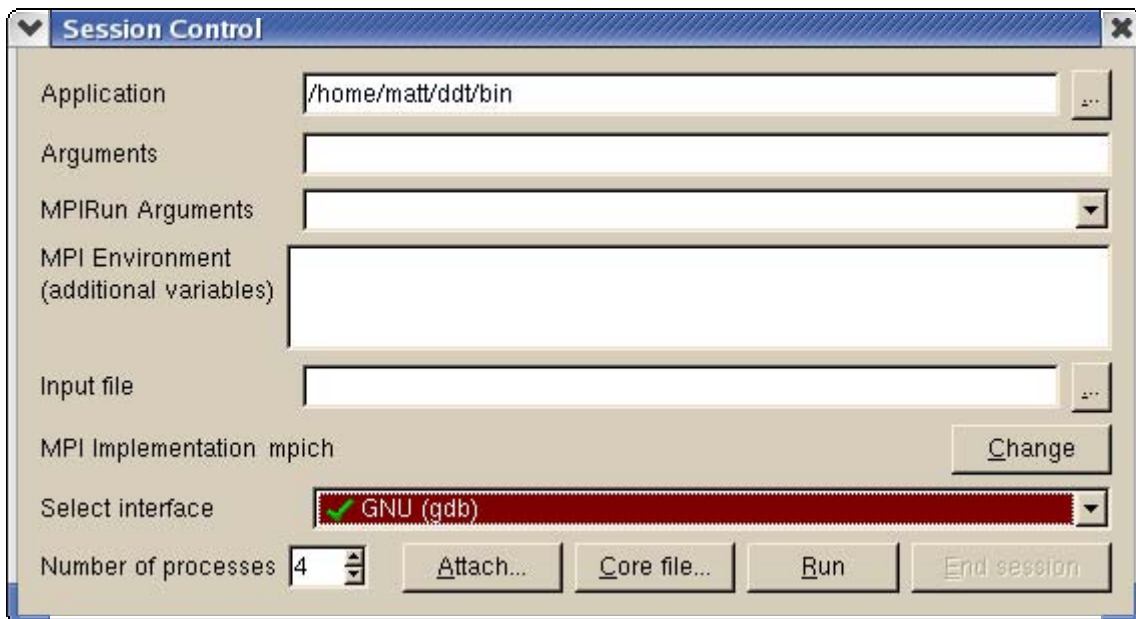
As always, when compiling the program that you wish to debug, you must add the debug flag to your compile command. For the most compilers this is `-g`. It is also advisable to turn off compiler optimisations as these can make debugging appear strange and unpredictable. If your program is already compiled without debug information you will need to remake the files that you are interested in again.

To start DDT simply type one of the following into a shell window:

```
ddt
ddt program_name
ddt program_name arguments
```

Once DDT has started it will display the `session Control` dialog used for configuring, starting and stopping debug sessions.

### Debugging Multi-Process Programs



*Fig.1 Session control dialog*

If your licence supports multi-process debugging, you will usually see the above dialog.

If your previous DDT session was debugging non-MPI codes, the view will be more compact than the above, and in this case you will be able to choose an MPI implementation and this will restore the full complement of parameter boxes.

In the application box, enter the full path to your application. If you specified one on the command line, this will already be filled in. You may alternatively select an application by clicking on the `...`.

The next box is for arguments. These are the arguments passed to your application, and will be automatically filled if you entered some on the command line.

The MPIRun arguments box is for arguments that are passed to `mpirun` or your equivalent (such as `scrun` on SCore, `mprun` on Solaris) – usually prior to your executable name in normal mpirun usage. You can place machine file arguments – if necessary – here. For most users this box can be left empty.

Please note that you should *not* enter the “-np” argument as DDT will do this for you.

The MPIRun environment should contain environment variables that should be passed to `mpirun` or its equivalent: some implementations allow you to set extra variables such as `MPI_MAX_CLUSTER_SIZE=1` on MPICH. These environment variables may also be passed to your program, depending on which MPI implementation your system uses. Most users will not need to use this box.

The choice of MPI implementation is critical to correctly starting DDT. Your system will normally use one particular MPI implementation. If you are unsure as to which to pick, try `generic`, consult your system administrator or Streamline. A list of settings for common implementations is provided in the appendix.

A debugger is then chosen to interact with each of your processes. In the select interface box, you should select from the appropriate debugger for your compiler and platform: See choosing the right debugger (page 15).

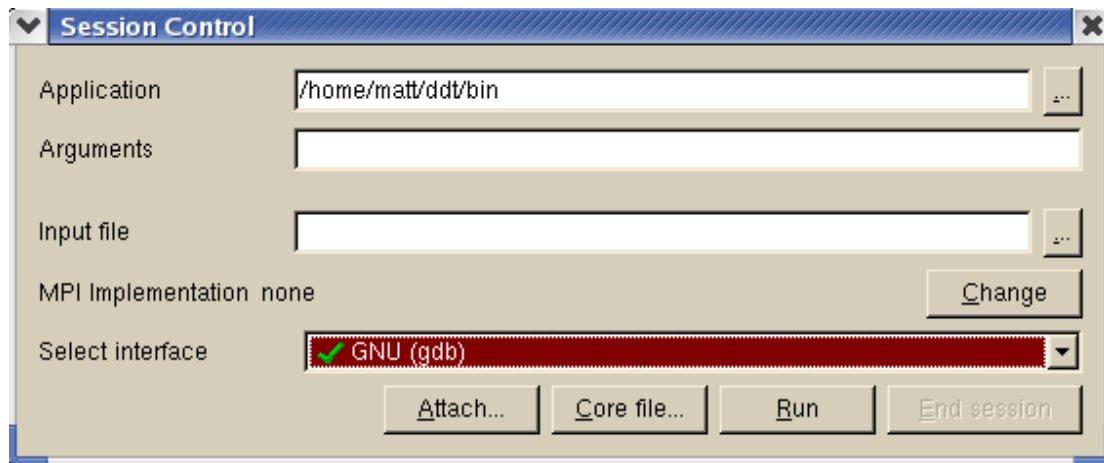
Finally you should enter the number of processes that you wish to run. DDT supports over 256 processes but this is limited by your licence.

Select run to start your program – or submit if working through a queue (see Configuring DDT with Queuing Systems). This will run your program through the debug interface you selected and will allow your MPI implementation to determine which nodes to start which processes on.

The kill button is provided to end the current session that you are running. This will close all processes and stop any running code. If any processes remain you may

have to clean them up manually using the `kill` command or a command provided with your MPI implementation such as `mpkill` on Solaris.

## Debugging Single-Process Programs



*Fig.2 Single-process session control dialog*

Users with single-process licences will immediately see the session control dialogue that is appropriate for single-process applications.

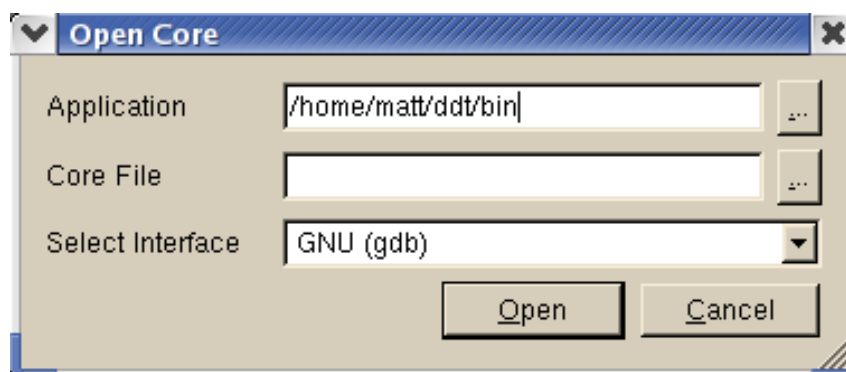
Users with multi-process licences should set the MPI Implementation to “none” – this will then place DDT into single-process mode and a dialog similar to the above will be shown. DDT can be placed into multi-process mode by changing the MPI implementation to anything except “none”.

Select the application – either by typing the file name in, or selecting using the browser by clicking the “...” button. Arguments can be typed into the supplied box.

Choose the correct debugger for the program. This will depend on the compiler which you have used: see the section on choosing the right debugger (page 15).

Finally press run to start your program.

## Debugging A Core File



*Fig.3 The open core dialog*

DDT allows you to debug a single core file generated by your application. Core files are not supported on all platforms and debug interfaces. The supported platforms and debug interfaces are:

- Irix (SGI dbx)
- HP-UX (gdb)
- Linux (Intel idb, GNU gdb and Absoft Fx2)
- Solaris (Sun dbx)

To debug using a core file, click on the `Open Core` button from the session control dialog. This switches to the `Open Core` dialog, which allows you to select an application, a core file and a debug interface. Clicking on `Open` will load the core file and start debugging it. While DDT is in this mode, you cannot play, pause or step because there is no process active – you are able to evaluate expressions and browse the variables and stack frames saved in the core file. Choosing to kill the session and restart will return DDT to its normal mode of operation.

## Attaching To Running Programs

DDT can attach to running processes on any machine you have access to, whether they are from MPI or scalar jobs, even if they have different executables and source paths. To make this possible you or your system administrator should provide a script called `remote-exec` in the \$DDTPATH/bin directory. It will be executed like this:

```
remote-exec HOSTNAME APPNAME [ARG1] [ARG2] ...
```

The script must start `APPNAME` on `HOSTNAME` with the arguments `ARG1 ARG2` and without further input (no password prompts). Standard output from `APPNAME` must appear on the standard output of `remote-exec`. On most systems the script can be implemented using `rsh`, as shown here:

```
#!/bin/sh
rsh $*
```

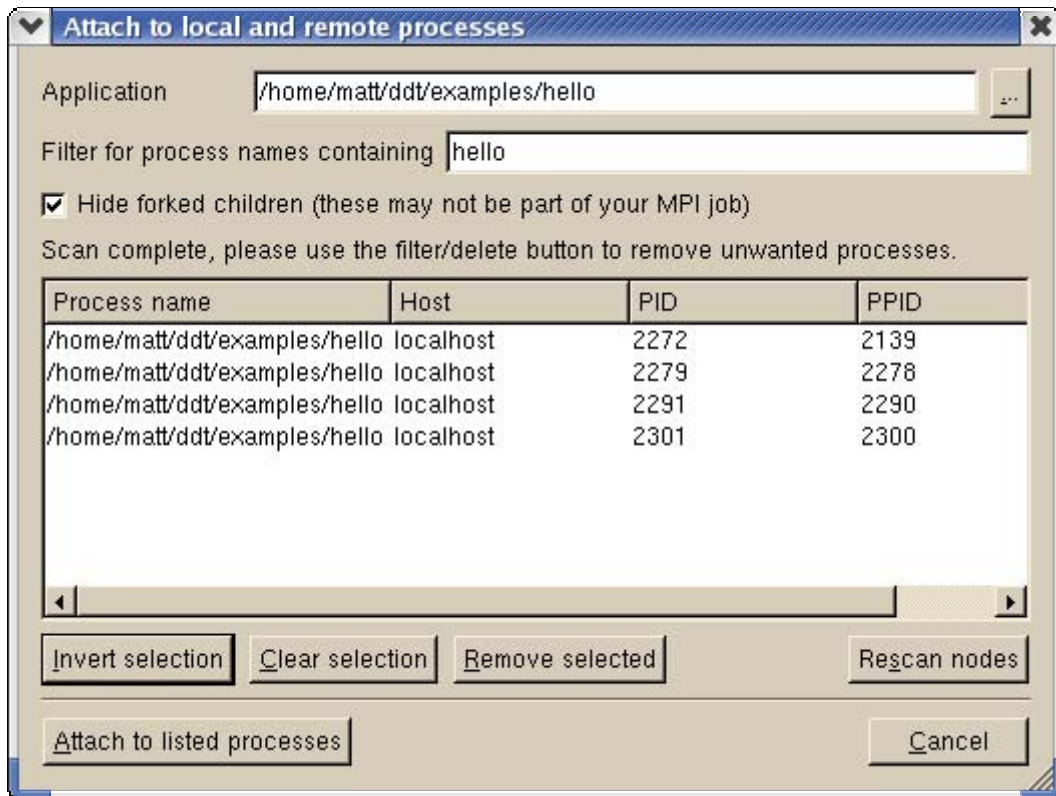
This particular implementation depends on having an appropriate `.rhosts` file in your home directory as explained in the `rsh` manpage. DDT comes with a default `remote-exec` file, set up as in the above example.

Once the script is set up (we advise testing it at the command line before using DDT) you must also provide a plain text file listing the nodes you want DDT to look for processes to attach to, e.g.

```
localhost
comp00
comp01
comp02
comp03
```

This file can be placed anywhere you have read access to, and may be provided by your system administrator. DDT must be given the location of this file - to do this just set the `Node list file` in the options window (`Session -> Configuration`). Each host name in this file will be sent to the `remote-exec` script as the `HOSTNAME` argument when DDT scans for and attaches to processes.

With the script and the node list configured, clicking on the `Attach` button will show DDT's Attach Dialog:



*Fig.4 Attach dialog*

Initially the list of processes will be blank while DDT scans the nodes provided in your node list file for running processes. When all the nodes have been scanned (or have timed out) the dialog will appear as shown. If you have not already selected an application executable to debug, you must do so here. Ensure that the list shows all the processes you wish to debug in your job, and no extra/unnecessary processes. You may modify the list by selecting and removing unwanted processes, or alternatively selecting the processes you wish to attach to and clicking on `Attach to selected processes`. If no processes are selected, DDT uses the whole visible list.

Some MPI implementations (such as MPICH) create forked (child) processes that are used for communication, but are not part of your job. To avoid displaying and attaching to these, make sure the `Hide forked children` box is ticked. DDT's definition of a forked child is a child processes that shares the parent's name. Some MPI implementations (such as the Scyld implementation) create your processes as children of each other. If you cannot see all the processes in your job, try clearing this checkbox and selecting specific processes from the list.

Once you click on the `Attach to selected/listed processes` button, DDT will use remote-exec to attach a debugger to each process you selected and will proceed to debug your application as if you had started it with DDT. When you end the debug



session, DDT will detach from the processes rather than terminating them – this will allow you to attach again later if you wish.

Because DDT is capable of attaching to several MPI and non-MPI programs in the same session, the rank numbers displayed may not be accurate – particularly if you have attached to a subset of the processes in your job. Streamline is working on methods to automatically determine process rank and/or display PID for processes with no rank. If this is an issue for you, please contact Streamline for information on and access to our feature development programme.

## Configuring DDT With Queuing Systems

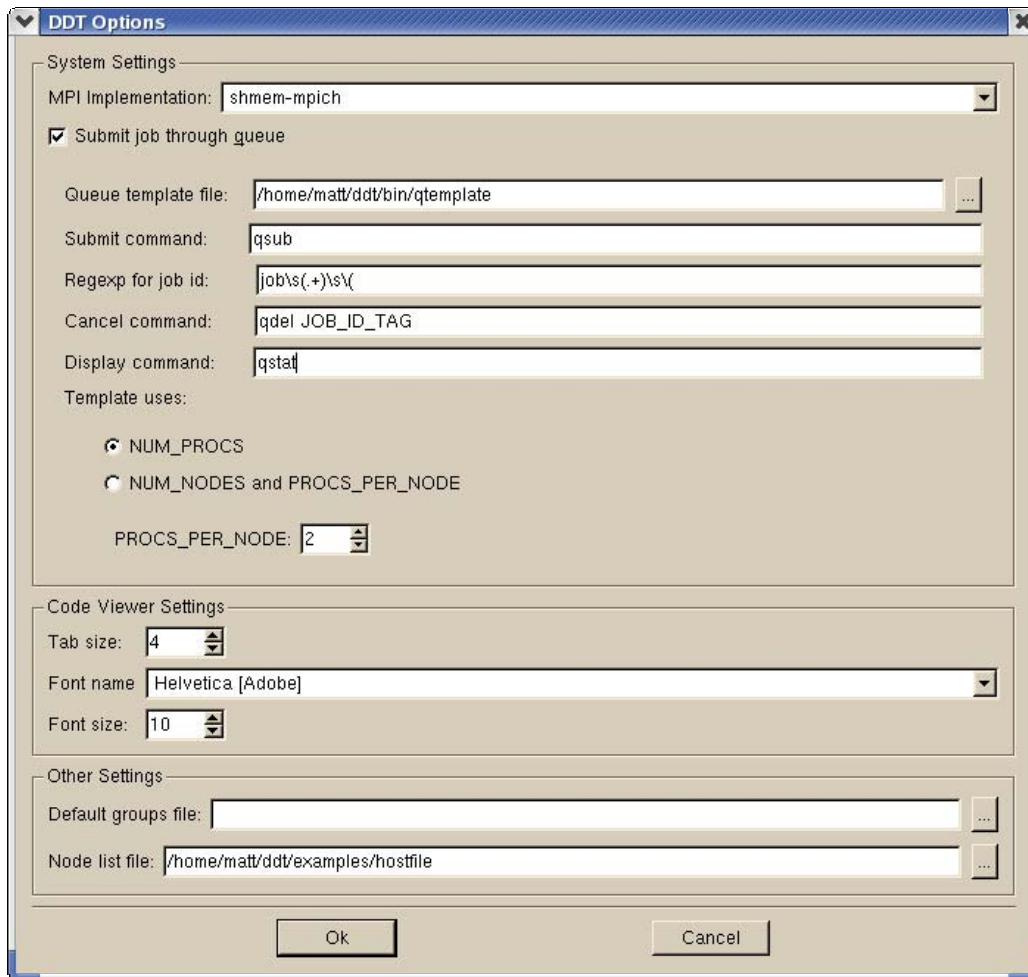
DDT can be configured to work with most job submission systems. In the configuration window, you should choose ``submit job through queue``. This displays extra options and switches DDT into queue submission mode.

Your system administrator may wish to provide a DDT config file containing the correct settings, removing the need for individual users to configure their own settings.

In this mode, DDT uses a template script to interact with your queue system. The “templates” subdirectory contains some example scripts that can be modified to meet your needs. `$DDTPATH/templates/sample.qtf`, demonstrates the process of creating a template file in some detail.

The template script is based on the file you would normally use to submit your job – typically a shell script that specifies the resources needed such as number of processes, output files, and executes ``mpirun``, ``vmirun``, ``poe`` or similar with your application. The most important difference is that job-specific variables, such as number of processes, number of nodes and program arguments, are replaced by capitalized keyword tags, such as `NUM_PROCS_TAG`. When DDT prepares your job, it replaces each of these keywords with its value and then submits the new file to your queue.

Once you have selected a queue template file, enter `submit`, `display` and `cancel` commands. When you start the debug session DDT will generate a submission file and append its filename to your `submit` command. For example, if you normally submit a job by typing ``job_submit -u myusername -f myfile`` then in DDT you should enter ``job_submit -u myusername -f`` as the `submit` command.



*Fig.5 Queuing systems*

To cancel a job, DDT will use a regular expression you provide to get a value for JOB\_ID\_TAG. This is substituted into the cancel command and executed to remove your job from the queue. The first bracketed expression in the regexp is used in the cancel command. The elements listed in the table are in addition to the conventional quantifiers, range and exclusion operators.

Element	Matches
C	A character represents itself
\t	A tab
.	Any character
\d	Any digit
\D	Any non-digit
\s	Whitespace
\S	Non-whitespace
\w	Letters or numbers (a word character)
\W	Non-word character

For example, your submit program might return the output “job id j1128 has been submitted” – one regular expression for getting at the job id is “id\s(.+)\shas”. If

you would normally remove the job from the queue by typing "job\_remove j1128" then you should enter "job\_remove JOB\_ID\_TAG" as DDT's cancel command.

Some queue systems allow you to specify the number of processes, others require you to select the number of nodes and the number of processes per node. DDT caters for both of these but it is important to know whether your template file and queue system expect to be told the number of processes (NUM\_PROCS\_TAG) or the number of nodes and processes per node (NUM\_NODES\_TAG and PROCS\_PER\_NODE\_TAG). If these terms seem strange, see 'sample.qtf' for an explanation of DDT's queue template system.

Please note that for DDT v1.5 onwards an extra environment variable must be set whilst working with most queue systems: DDT\_IGNORE\_MPI\_OUTPUT should be set to 1 prior to starting DDT.

## Starting A Job In A Queue

Clicking submit from the usual session control dialog will display the queue status until your job starts. DDT will execute the display command every second and show you the standard output. If your queue display is graphical or interactive then you cannot use it here.

If your job does not start or you decide not to run it, click on `Cancel Job`. If the regular expression you entered for getting the job id is invalid or if an error is reported then DDT will not be able to remove your job from the queue – it is strongly recommend you check the job has been removed before submitting another as it is possible for a forgotten job to execute on the cluster and either waste resources or interfere with other debug sessions.

Once your job is running, it will connect to DDT and you will be able to debug it.

## Choosing The Right Debugger

DDT uses standard command-line debuggers to control your program. The number of debuggers listed will depend on your platform and licence.

In general, the best debugger to choose is the one written for your compiler. The choice for Linux can be overwhelming, and this table summarizes the options.

Compiler	Recommended Debugger	Version	Notes
----------	----------------------	---------	-------

Absoft	FX2	Beta	Available from Absoft and Streamline
GNU	GDB	5.2.1 and above	Patched GDB included with DDT supports Fortran
Intel	IDB	Build date 2003-03-03 and above	More recent versions (2003-06-10+) are faster and have better memory usage
Portland	PGDBG	4.0.2 and above	May require MPICH to be compiled -g, except for most recent PGDBG A supplied GDB, patched to support Portland, can also be used for codes compiled - Mstabs

AMD Opteron users should note that there are 32 bit and 64 bit versions of GDB supplied, these should be used for 32 bit and 64 bit applications as appropriate.

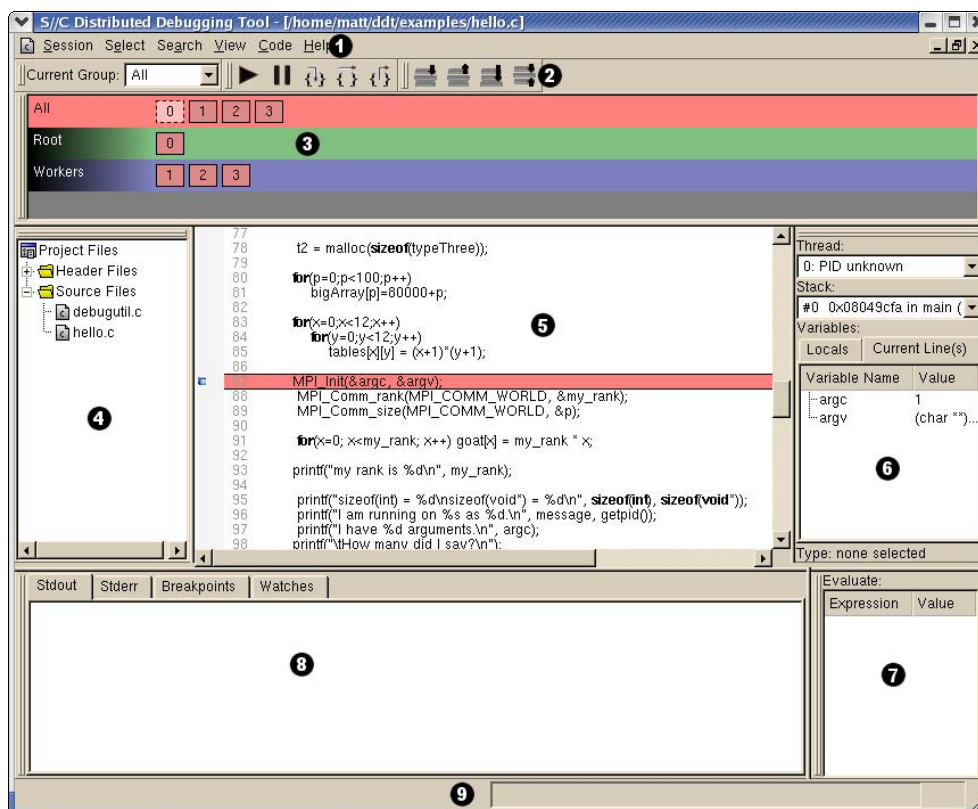
## 4. DDT Overview

DDT uses a multi-document interface which is common in many present day applications. This allows you to have many source files open, and to view one in the full workspace area, or to tile or cascade them. You may switch between these modes in the `Window` menu.

Each component of DDT (labeled and described in the key) is a dockable window, which may be dragged around by a handle (usually on the left hand edge). Components can also be dragged outside of DDT to form a new window.

You can hide or show each component using the `View` menu. The screen shot shows the default DDT layout.

Key
(1) Menu bar
(2) Process controls
(3) Process group window (DDT-MP)
(4) File window
(5) Code window
(6) Variable window
(7) Expression window
(8) Output window
(9) Status bar



*Fig.6 DDT main window*

Please note that on some platforms, the default screen size can be insufficient to display the status bar – if this occurs, you should expand the DDT window until DDT is completely visible.

## Saving And Loading Sessions

Most of the user–modified parameters and windows are saved by right clicking and selecting a save option in the corresponding window.

However, DDT also has the ability to load and save all these options concurrently to minimize the inconvenience in restarting sessions. Saving the session stores such things as process groups, the contents of the evaluate window and more. This ability makes it easy to debug code with the same parameters set time and time again.

To save a session simply use the `save Session` option from the `Session` menu. Enter a filename (or select an existing file) for the save file and click OK. To Load a session again simply choose the `Load Session` option from the `session` menu, choose the correct file and click OK.

## Source Code

When DDT begins a session, source code is automatically found from the information compiled in the executable.

Source and header files found in the executable are reconciled with the files present on the front–end server, and displayed in a simple tree view. Source files can be loaded for viewing by clicking on the filename.

Whenever a selected process is stopped, the source code browser will automatically leap to the correct file and line, if the source is available.

## Finding Lost Source Files

On some platforms, not all source files are found automatically. This can also occur, for example, if the executable or source files have been moved since compilation. Extra directories to search for source files can be added by right clicking whilst in the project files window, and selecting “Add source directories”. After adding the directories necessary, right click again and select “Scan for more sources”.

## Dynamic Libraries

If a library is loaded dynamically, its source file may not be found at the time the program starts. The source can be added by right clicking whilst in the project files window, and selecting “Scan for more sources”.

## Finding Code Or Variables

The `Find` and `Find in Files` dialogs are found from the `search` menu. The `Find` dialog will find occurrences of an expression in the currently visible source file. The `Find in Files` dialog searches all source and header files associated with your program and lists the matches in a result box. Click on a match to display the file in the main code window and highlight the matching line; this can be of particular use for setting a breakpoint at a function.

## Jump To Line

DDT has a jump to line function which enables the user to go directly to a line of code. This is found in the `search` menu. A dialog will appear in the center of your screen. Enter the line number you wish to see and click OK. This will take you to the correct line providing that you entered a line that exists. You can also use the hotkey CTRL-G to access this function quickly.

## Editing Source Code

If, prior to starting DDT, you set the environment variable “DDT\_EDITOR” to be the name of an editor/script that is an X application then on right clicking in the source code window, DDT will offer to launch your editor and bring your code up at the line selected by the mouse. For example:

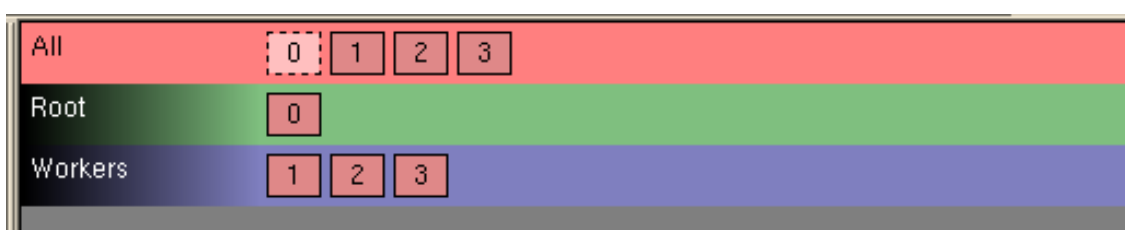
```
export DDT_EDITOR=emacs
export DDT_EDITOR=xterm -e vi
```

## 5. Controlling Program Execution

Whether debugging a multi-process or a single process code, the mechanisms for controlling program execution are very similar.

In multi-process mode, most of the features described in this section are applied using process groups, which we describe now. For single process mode, the commands and behaviors are identical, but apply to only a single process – freeing the user from concerns about process groups.

### Process Control And Process Groups



*Fig.7 The process group viewer*

MPI programs are designed to run as more than one process and can span many machines. DDT allows you to group these processes so that actions can be performed on more than one process at a time. The status of processes can be seen at a glance by looking at the process group viewer. The process group viewer is (by default) at the top of the screen with multi-coloured rows. Each row relates to a group of processes and operations can be performed on the currently highlighted group (e.g. playing, pausing and stepping) by clicking on the toolbar buttons. Switch between groups by clicking on them or their processes – the highlighted group is indicated by a lighter shade.

Each process is represented by a square containing its MPI rank (0 through n-1). The squares are colour-coded; red for a paused/stopped process and green for a running process. Any selected processes are highlighted with a lighter shade of their colour and the current process also has a dashed border. When a single process is selected the local variables are displayed in the variable viewer and displayed expressions are evaluated. You can make the code viewer jump to the file and line for the current stack frame (if available) by double-clicking on a process.

Groups can be created, deleted, or modified by the user at any time, with the exception of the `All` group, which cannot be modified. To copy processes from one group to another, simply click and drag the processes. To delete a process,



press the delete key. When modifying groups it is useful to select more than one process by holding down one or more of the following:

Key	Description
Control	Click to add/remove process from selection
Shift	Click to select a range of processes
Alt	Click to select an area of processes

Note: Some window managers (such as KDE) use Alt and drag to move a window – you must disable this feature in your window manager if you wish to use the DDT's box select.

Groups are added and deleted from a context-sensitive menu that appears when you right-click on the process group widget. This menu can also be used to rename groups, delete individual processes from a group and jump to the current position of a process in the code viewer. You can load and save the current groups to a file, but be careful when using this feature as the number of processes might have changed since the file was saved. If you are on a process group already when you right click, the further option of `add complement with` will be enabled. This allows a new group to be created using the set difference of the currently selected group and the one you choose from the submenu.

*Hint: To communicate with a single process, create a new group and drag that process into it.*

## Hotkeys

DDT comes with a pre-defined set of hotkeys to enable easy control of your debugging. All the features you see on the toolbar and several of the more popular functions from the menus have hotkeys assigned to them. Using the hotkeys will speed up day to day use of DDT and it is a good idea to try to memorize these.

Key	Function
F9	Play
F10	Pause
F5	Step into
F8	Step over
F6	Step out
CTRL-D	Down stack frame
CTRL-U	Up stack frame
CTRL-B	Bottom stack frame
CTRL-A	Align stack frames with current
CTRL-G	Goto
CTRL-F	Find

## Starting, Stopping And Restarting A Program

The Session Control dialog can be accessed at almost any time while DDT is running. If a program is running you can kill it and run it again or run another program. When DDT's startup process is complete your program should automatically stop either at the main function for non-MPI codes, or at the MPI\_Init function for MPI.

When a job has run to the end DDT will show a dialog box asking if you wish to restart the job. If you select yes then DDT will kill any remaining processes and clear up the temporary files and then restart the session from scratch with the same program settings.

When killing a job, DDT will attempt to ensure that all the processes are shutdown and clear up any temporary files. If this fails for any reason you may have to manually kill your processes using ``kill``, or a method provided by your MPI implementation such as ``lamclean`` for LAM/MPI.

## Stepping Through A Program

To start the program running click ``play`` and to stop it at any time click ``pause``. For multi-process DDT these start/stop all the processes in the current group (see Process Control and Process Groups).

Like many other debuggers there are three different types of step available. The first is ``step into`` that will move to the next line of source code unless there is a function call in which case it will step to the first line of that function. The second is ``step over`` that moves to the next line of source code in the bottom stack frame. Finally, ``step out`` will execute the rest of the function and then stop on the next line in the stack frame above.

When using step out be careful not to try and step out of the main function. Doing this will cause problems with the debugger most likely resulting in your program hanging. With some debuggers DDT will detect the defunct process and time out.

## Setting Breakpoints

First locate the position in your code that you want to place a breakpoint at. If you have a lot of source code and wish to search for a particular function you can use the ``Find` / `Find in files` dialog. Clicking the right mouse button in the code`

window displays a menu showing several options, including one to add or remove a breakpoint. In multi-process mode this will set the breakpoint for every member of the current group.

Every breakpoint is listed under the breakpoints tab towards the bottom of DDT's window.

## Conditional Breakpoints

Stdout   Stderr   Breakpoints   Watches					
	Group	Filename	Line	Condition	Full path
<input checked="" type="checkbox"/>	All	hello.c	93		/home/matt/ddt/examples/hello.c
<input checked="" type="checkbox"/>	Root	hello.c	101	x=3	/home/matt/ddt/examples/hello.c
<input type="checkbox"/>	Workers	hello.c	124		/home/matt/ddt/examples/hello.c
<input checked="" type="checkbox"/>	Workers	hello.c	126		/home/matt/ddt/examples/hello.c

*Fig.8 The breakpoints table*

Select the breakpoints tab to view all the breakpoints in your program. You may add a condition to any of them by clicking on the condition cell in the breakpoint table and entering an expression that evaluates to true or false. Each time a process (in the group the breakpoint is set for) passes this breakpoint it will evaluate the condition and break only if it returns true (typically any non-zero value). You can drag an expression from the evaluate window into the condition cell for the breakpoint and this will be set as the condition automatically.

## Suspending Breakpoints

A breakpoint can be temporarily deactivated and reactivated by checking/unchecking the activated column in the breakpoints panel.

## Deleting A Breakpoint

Breakpoints are deleted by either right-clicking on the breakpoint in the breakpoints panel, or by right clicking at the file/line of the breakpoint whilst in the correct process group and right clicking and selecting delete breakpoint.

## Loading And Saving Breakpoints

To load or save the breakpoints in a session right click in the breakpoint panel and select the load/save option. Breakpoints will also be loaded and saved as part of the load/save session.

## Synchronizing Processes

If the processes in a process group are stopped at different points in the code and you wish to re-synchronize them to a particular line of code this can be done by right clicking on the line at which you wish to synchronize them to and selecting synchronize group. This effectively sets all the processes in the selected group running and puts a break point at the line at which you choose to synchronize the processes at, ignoring any breakpoints that the processes may encounter before they have synchronized at the specified line.

If you choose to synchronize your code at a point where all processes do not reach then the processes that cannot get to this point will run to the end.

*Note: Though this ignores breakpoints while synchronizing the groups it will not actually remove the breakpoints.*

## Setting A Watch

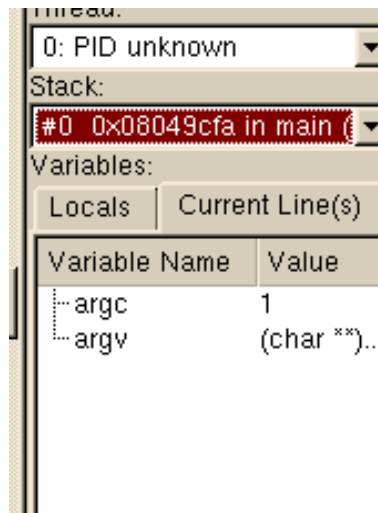
Stdout	Stderr	Breakpoints	Watches
Process	Expression	Stack frame	
0	my_rank	#0 0x08049cfa in main (argc=1, argv=0xbffd854, environ=0xbffd86c) at hello.c:87	
2	beingWatched	#0 0x08049cfa in main (argc=1, argv=0x8097be0, environ=0xbfffeaa8) at hello.c:87	

*Fig.9 The watches table*

A watchpoint is a type of breakpoint that monitors a variable's value and causes a break once the value is changed. Unlike breakpoints, it is not set on a line in the code window. Instead you must drag a variable from either the variables window or the evaluate window into the watches table. It is not generally useful to watch a variable that is allocated on the stack rather than the heap, and some debug interfaces (such as GDB) will remove a watchpoint when its variable goes out of scope. Variables on the heap do not go out of scope.

For multi-process debugging, watches can only be set on a single process and not a whole group.

## Examining The Stack Frame



*Fig.10 Selecting a stack frame*

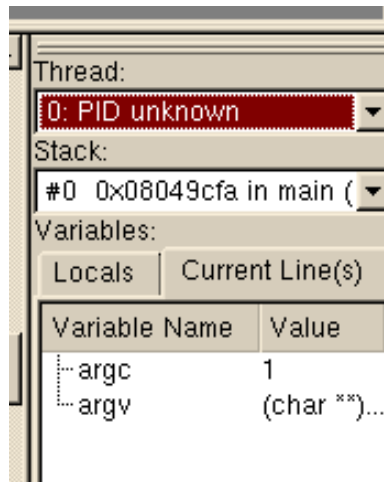
The stack frame (the current position in the stack) is displayed and changed using a drop-down list in the variable window. When you select a stack frame DDT will jump to that position in the code (if it is available) and will display the local variables for that frame. The toolbar can also be used to step up or down the stack, or jump straight to the bottom-most frame.

## Align Stacks

The align stacks button, or CTRL-A hotkey, sets the stack of the current thread on every process in a group to the same level - where possible - as the current process.

This feature is particularly useful where processes are interrupted - by the pause button - and are at different stages of computation. This enables tools such as the cross-process comparison window to compare equivalent local variables, and also simplifies casual browsing of values.

## Examining Threads

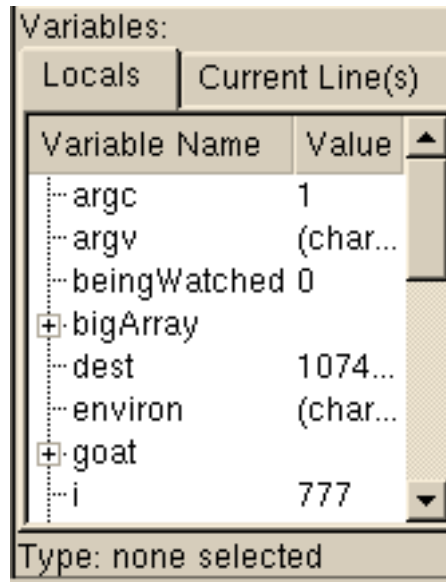


*Fig.11 Selecting a thread*

You can select a thread from the drop-down list in the variable window. Changing the thread will update the stack frame and local variables. Multi-process DDT users should note that many MPI implementations are not thread safe so you must be very careful when using threads. DDT supports both OpenMP and native threading libraries such as pthreads. If your program uses the pthreads library (either natively or via OpenMP) you may see an extra handler thread – this is not part of your program but is used by the operating system to manage multiple threads and may be present even when your program is only using one thread.

## 6. Variables And Data

The variable window contains two tabs that provide different ways to list your variables. The `Locals` tab contains all the variables for the current stack frame, while the `Current Line` tab displays all the variables referenced on the currently selected lines.



*Fig.12 Displaying variables*

### Current Line

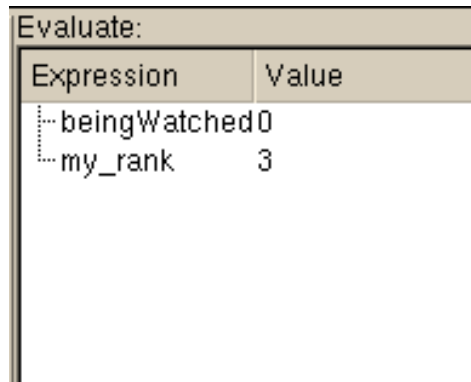
You can select a single line by clicking on it in the code viewer – or multiple lines by clicking and dragging. The variables are displayed in a tree view so that user-defined classes or structures can be expanded to view the variables contained within them. You can drag a variable from this window into the `Evaluate Expression` window; it will then be evaluated in whichever stack frame, thread or process you select.

### Local Variables

The locals tab contains local variables for the current process's currently active thread and stack frame.

For Fortran codes the amount of data classed as local can be substantial – as this can include many global or common block arrays. Should this prove problematic, it is best to conceal this tab underneath the current line tab as this will not then update after ever step.

## Arbitrary Expressions And Global Variables



Evaluate:	
Expression	Value
...beingWatched0	
...my_rank	3

*Fig.13 Evaluating expressions*

Since the global variables and arbitrary expressions do not get displayed with the local variables, you may wish to use the `Current Line` tab in the local variables and click on the line in the code window containing a reference to the global variable.

Alternatively, the Evaluate panel can be used to view the value of any arbitrary expression. Right click on the evaluate window, click on `Add Expression`, and type in the expression required in the current source file language. This value of the expression will be displayed for the current process and stack/thread, and is updated after every step.

## Changing Data Values

In the evaluate window, the value of an expression may be set by right clicking and selecting "edit value". This will change the value of the expression in the currently selected process.

*Note: This depends on the variable existing in the current stack frame and thread.*

## Examining Pointers

Pointer contents cannot normally be examined in the variables window but after dragging them into the evaluate window you can right click and select any of the following new options: view as vector, reference, or de-reference.

If a structure contains another pointer you must drag this pointer onto its own line in the evaluate window before you can start referencing/de-referencing it.

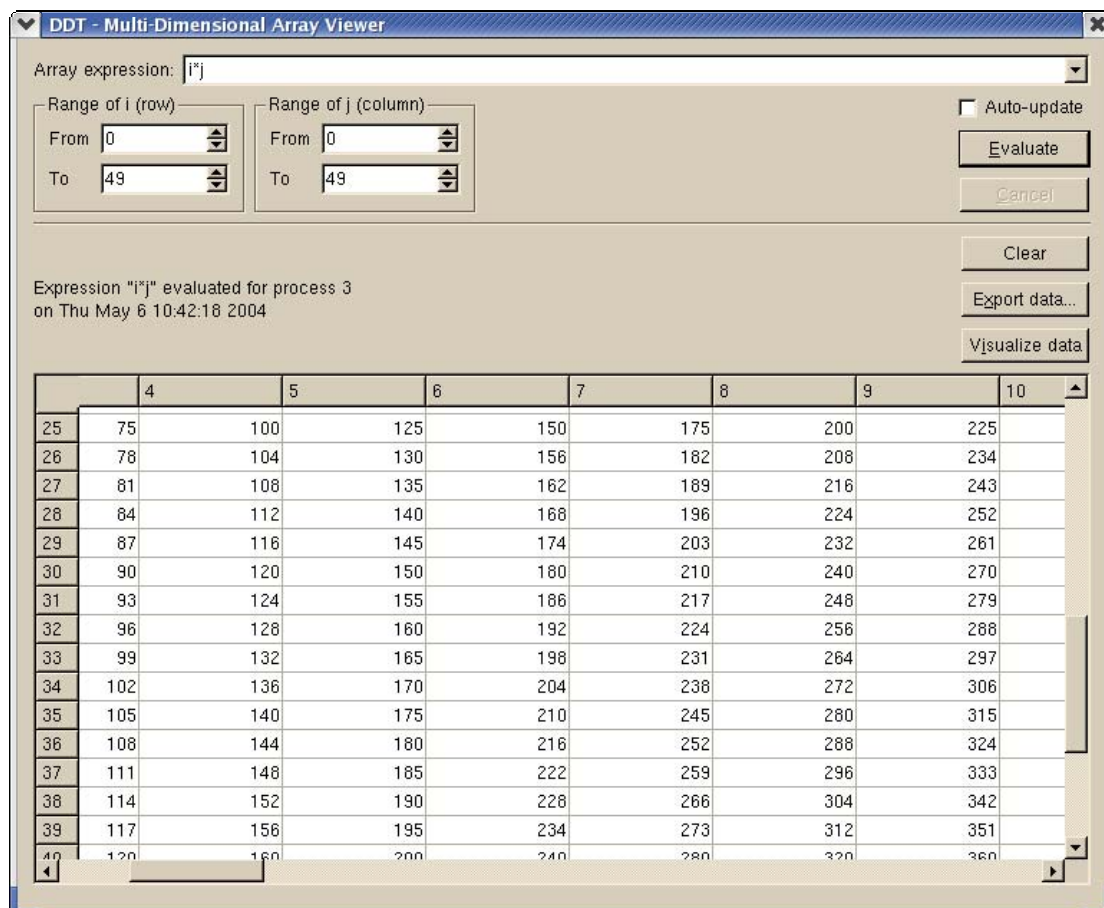


## Examining Multi-Dimensional Arrays

Large multi-dimensional arrays are not easy to view in a tree structure so DDT provides a multi-dimensional array viewer. This allows you to type in an expression optionally using up to two variables (i and j) and set the range for these variables.

Clicking `Evaluate` will fill a table with all evaluations of i and j. This data can be exported to a csv (comma-separated variables) file which can be plotted or analyzed in your favorite spreadsheet.

You can view any two-dimensional slice of your data by entering an expression based on i and j, such as  $A(i+j,j,1)$  in Fortran or `myArray[1][j][i+j]` in C/C++. Once the data is loaded into DDT's table you can visualize the data as a surface in 3-D space (see next section).



*Fig. 14 Multi-dimensional array viewer*

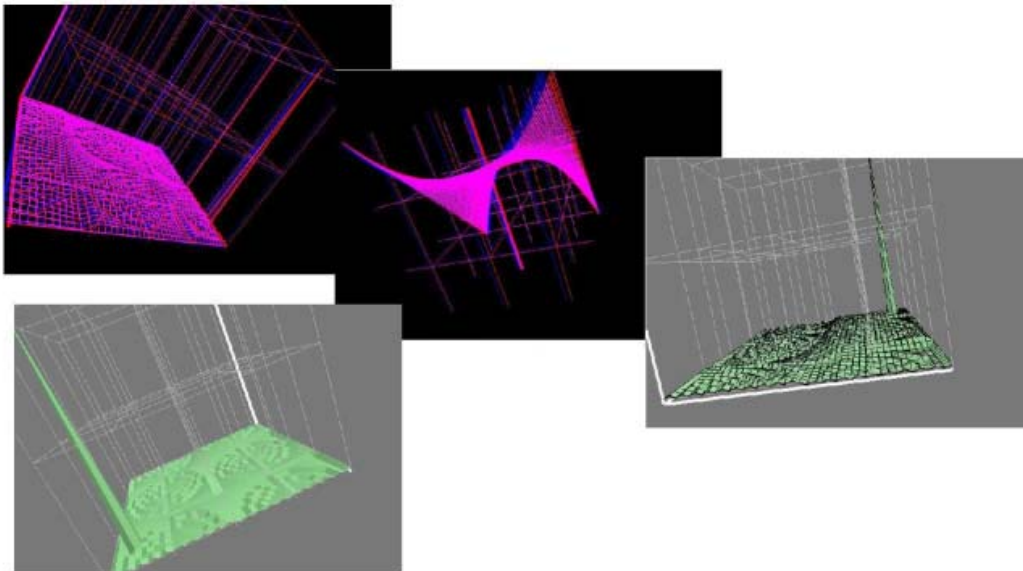
## Visualizing Data

A 2-D slice of an array, or table of expressions, may be displayed as a surface in 3-D space through the multi-dimensional array (MDA) viewer. After filling the table of the MDA viewer with values (see previous section), click `Visualize` to open a 3-D view of the surface. To display surfaces from two or more different processes on the same plot simply select another process in the main process group window and click evaluate in the MDA window, and when the values are ready, click `Visualize` again. The surfaces displayed on the graph may be hidden and shown using the checkboxes on the right-hand side of the window.

If OpenGL is enabled on your system, the graph may be moved and rotated using the mouse and a number of extra options are available from the window toolbar.

The mouse controls in OpenGL are:

- Hold down the left button and drag the mouse to rotate the graph.
- Hold down the right button to zoom – drag the mouse forwards to zoom in and backwards to zoom out.
- Hold the middle button and drag the mouse to move the graph.



*Fig.15 DDT visualization*

The toolbar and menu offer options to configure lighting and other effects, including the ability to save an image of the surface as it currently appears. There is even a stereo vision mode that works with red-blue glasses to give a convincing

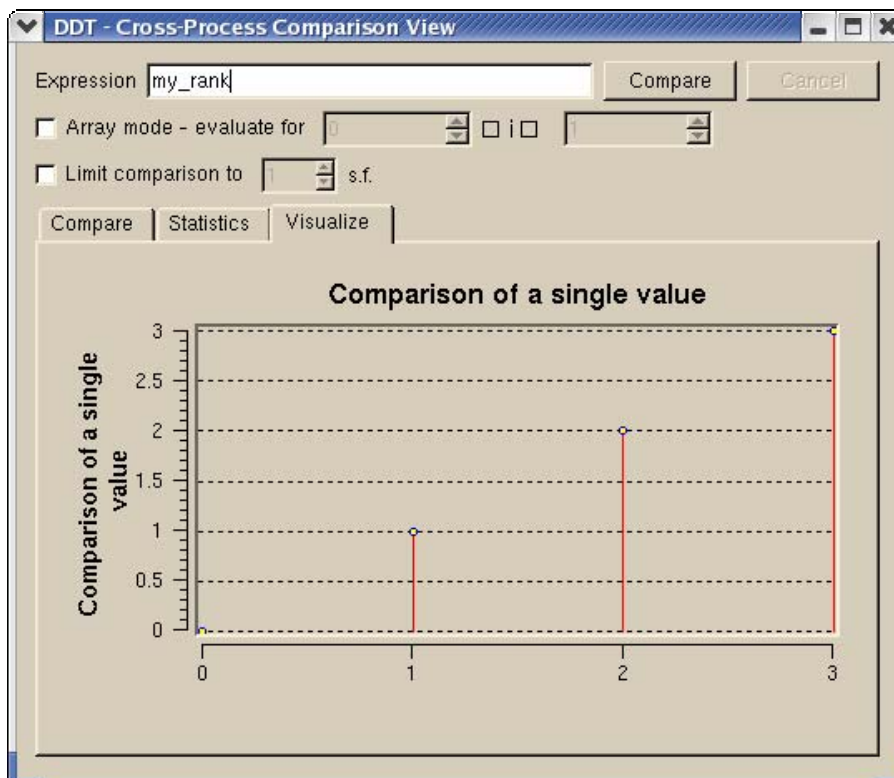
impression of depth and form. Contact Streamline if you need to get hold of some 3D glasses.

If OpenGL is not enabled, you will still be able to visualize your data, but further manipulation is not possible.

## Cross-Process Comparison

The cross-process comparison window can be used to analyze expressions calculated on each of the processes in the current process group. This window displays information in three ways:

- Grouped by expression value
- Statistically – maximum, minimum, variance and similar with a box-and-whisker plot which displays max, min and interquartile range graphically
- A plot of values. In the case of a 1-D array expression the plot of values will display a line graph of values for all processes and these can be turned on and off individually by clicking the appropriate checkbox.



*Fig.16 Cross process comparison*

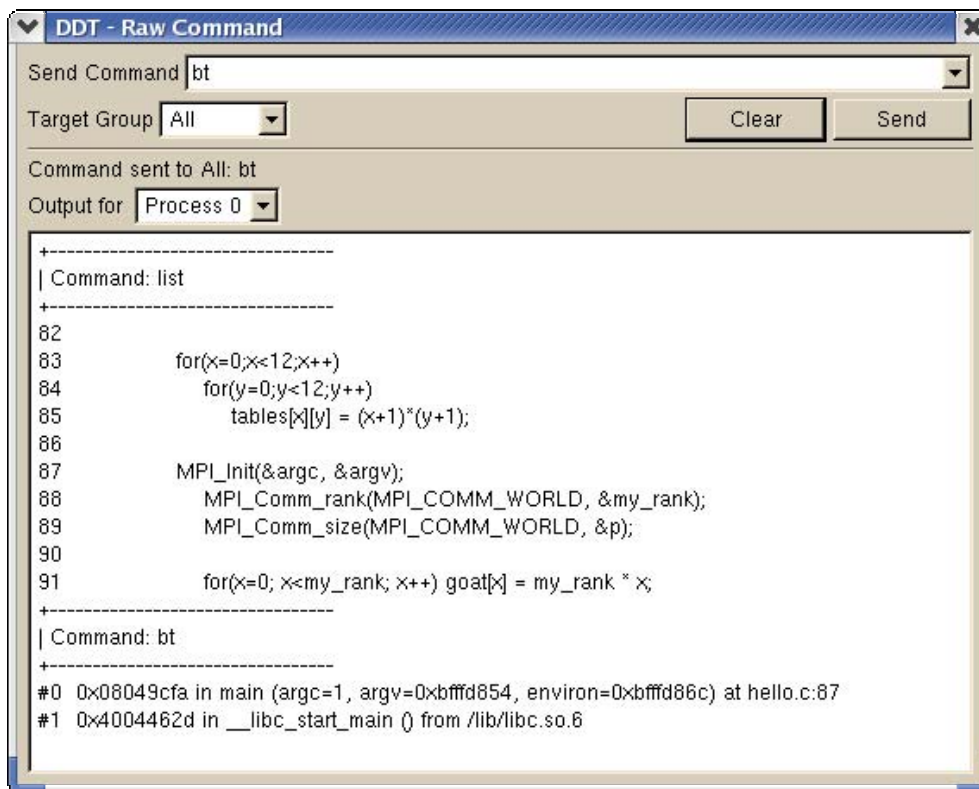
To use this window, select the cross-process comparison window from the view menu. Type the expression that you wish to analyze. If you wish to compare an

array, select the `use variable i` button and type in the bounds that you require. Click the `compare` button, and the current values will be evaluated on all the processes in the current group. If a process is still running in the current group, its value will not be found; press cancel and pause all the processes before trying again.

## Viewing Registers

To view the values of machine registers on the currently selected process, select the registers window from the windows pull-down menu. These values will be updated after each instruction, change in thread or change in stack frame.

## Interacting Directly With The Debugger



*Fig.17 Debugger interaction window*

DDT provides a raw command dialog that will allow you to send commands directly to the debugger interface. This dialog bypasses DDT and its book-keeping – if you set a breakpoint here, DDT will not list this in the breakpoint list, for example.

Be careful with this dialog; we recommend you only use it where the graphical interface does not provide the information or control you require. Sending

commands such as `quit` or `kill` may cause the interface to stop responding to DDT.

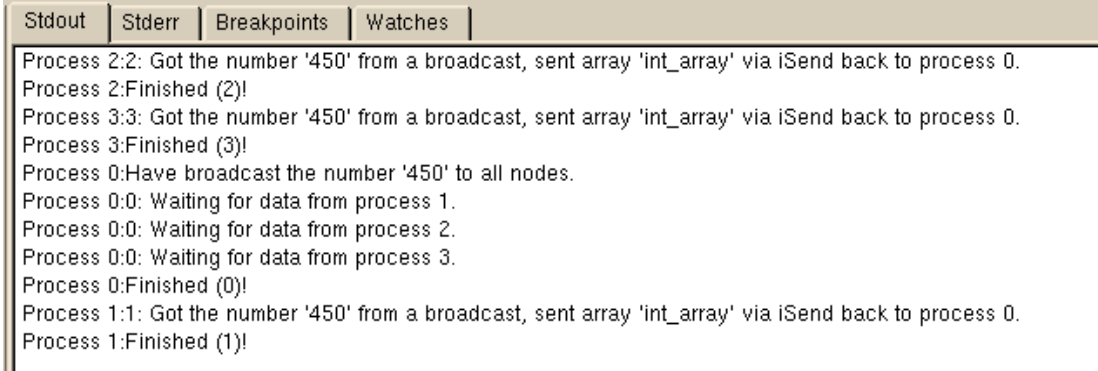
Each command is sent to a group of processes (selected from within the dialog box – not necessarily the current group). To communicate with a single process, create a new group and drag that process into it.

The raw process command window will not work with running processes and requires all processes in the chosen group to be paused.

## 7. Program Input And Output

DDT collects and displays output from all processes, placing output and error streams in separate panels that are controllable and allow output to be filtered by origin. Both standard output and error are handled identically, although on most MPI implementations, error is not buffered but output is and consequently can be delayed.

### Viewing Standard Output And Error



```

Stdout | Stderr | Breakpoints | Watches
-----|-----|-----|-----
Process 2:2: Got the number '450' from a broadcast, sent array 'int_array' via iSend back to process 0.
Process 2:Finished (2)!
Process 3:3: Got the number '450' from a broadcast, sent array 'int_array' via iSend back to process 0.
Process 3:Finished (3)!
Process 0:Have broadcast the number '450' to all nodes.
Process 0:0: Waiting for data from process 1.
Process 0:0: Waiting for data from process 2.
Process 0:0: Waiting for data from process 3.
Process 0:Finished (0)!
Process 1:1: Got the number '450' from a broadcast, sent array 'int_array' via iSend back to process 0.
Process 1:Finished (1)!

```

*Fig.18 Standard output window*

At the bottom of the screen (by default) there are tabs for displaying standard output and standard error.

The contents of these panels can be cut and pasted into the X-clipboard.

### Displaying Selected Processes

By right clicking the mouse you can choose whether to view output for the current process, the current process group if no process is selected, or for all processes. You also have the option to copy a selection to the clipboard.

MPI users should note that most MPI implementations place their own restrictions on program output. Some buffer it all until MPI\_Finalize is called, others may ignore it or send it all through to one process. If your program needs to emit output as it runs, Streamline suggest writing to a file.

## Saving Output

By right-clicking in an output window, it is possible to save the contents of the window to a file.

## Sending Standard Input (DDT-MP)

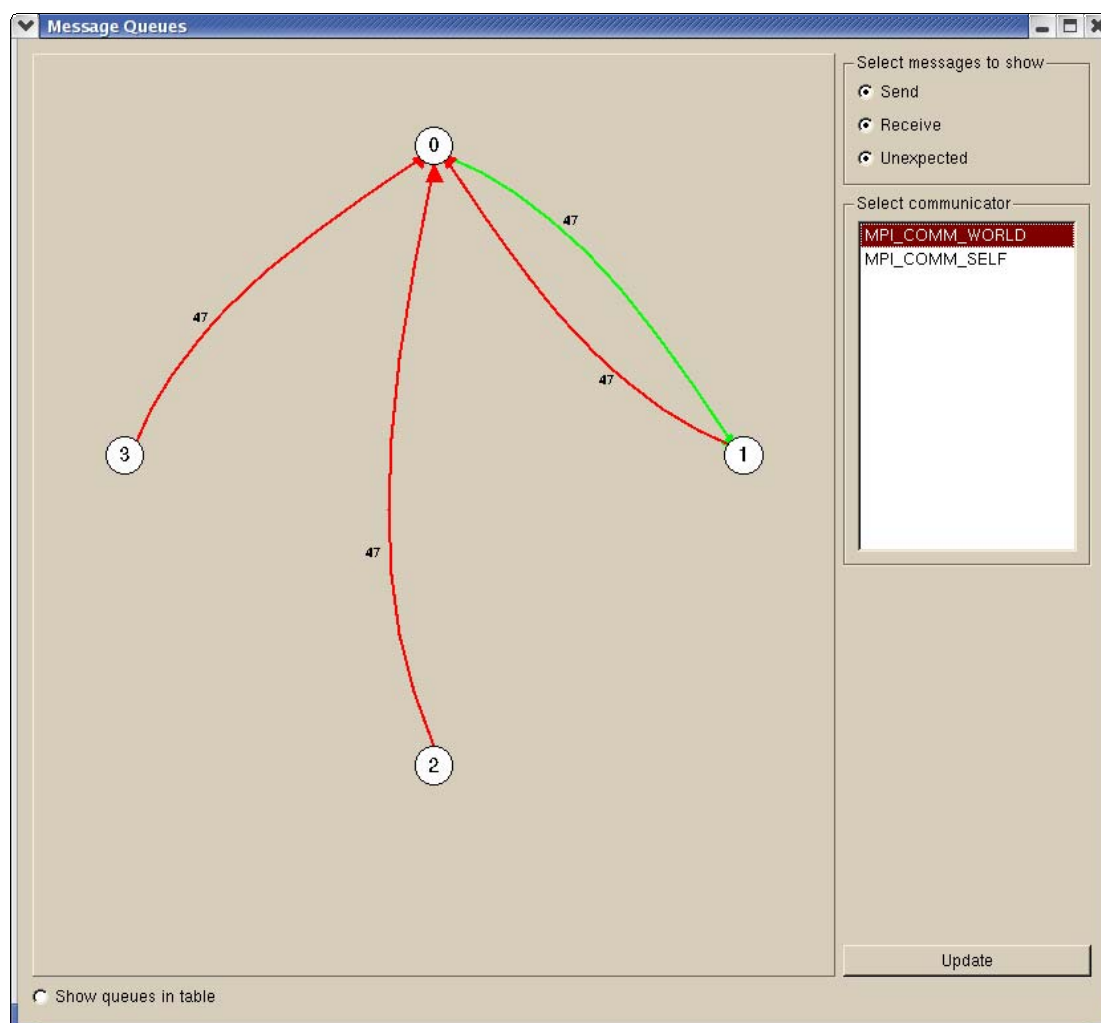
DDT provides an `Input File` option in the session control window. Using this window you may select the file you wish to use as the input file. DDT will automatically insert the correct arguments to your MPI implementation.

Alternatively in DDT you may enter the arguments directly in the `MPI Arguments` box. Typically you add an option to the mpirun command line, such as `-stdin filename`. You may add the same options to the `MPI Run Arguments` box when starting your DDT session.

*Note: If DDT is running on a fork-based system such as Scyld, or a `-comm=shared` compiled MPICH, your program may not receive an EOF correctly from the input file. If your program seems to hang while waiting for the last line or byte of input, this is likely to be the problem. See the FAQ or contact Streamline for a list of possible fixes.*

## 8. Message Queues

Open the Message Queue view by selecting 'Message Queues' from the 'View' menu.



*Fig. 19 Message queue window*

When the window appears you can click 'Update' to get the current queue information. Please note that this will stop all running processes. While DDT is gathering the data a dialog box will be displayed and you can cancel the request at any time.

You must select a communicator to see the messages in that group. The ranks displayed in the diagram are the ranks within the communicator (not MPI\_COMM\_WORLD). Different colours are used to display messages from each type of queue.

DDT use the message queue debug interface to gather queue information. Within this interface each communicator has three distinct message queues:



<b>Label</b>	<b>Description</b>
Send Queue	Represents all the outstanding send operations
Receive Queue	Represents all the outstanding receive operations
Unexpected Message Queue	Represents messages that have been sent to this process but have not been received

In order to take advantage of the message queue view within DDT you need to compile the debug interface for your MPI implementation. In MPICH this is done by using '--enable-debug' when running configure. LAM automatically compiles the library.

DDT will try to load the default library for the MPI implementation (provided one exists) but for this to happen it must be in the LD\_LIBRARY\_PATH. If this is not convenient you can set the environment variable, DDT\_QUEUE\_DLL, which can be the absolute path (or just in the LD\_LIBRARY\_PATH).

If you experience problems connecting to the message queue library when attaching to a process see the FAQ for possible solutions.

## 9. The Licence Server

The licence server supplied with DDT is capable of serving clients for several different licences, enabling one common server to serve all Streamline software in an organization.

### Running The Server

For security, the licence server should be run as an unprivileged user (eg. nobody). If run without arguments, the server will use licences in the current directory (files matching Licence\* and License\*). An optional argument specifies the path to be used instead of the current.

System administrators will normally wish to add scripts to start the server automatically during booting.

### Running DDT Clients

DDT will, as is also the case for fixed licences, use a licence file either specified via environment variables (DDT\_LICENCE\_FILE or DDT\_LICENSE\_FILE) or from the default location of \$DDTPATH/Licence.

In the case of floating licences this file is unverified and in plain-text, it can therefore be changed by the user if settings need to be amended.

The fields are:

Name	Required	Description
hostname	Yes	The hostname, or IP address of the licence server
ports	No	A comma separated list of ports to be tried locally for frontend-backend communication in DDT, Defaults to 4242,4243,4244,4244,4245
serial_number	Yes	The serial number of the server licence to be used
serverport	Yes	The port the server listens on
type	Yes	Must have value 2 – this identifies the licence as needing a server to run properly

*Note: The serial number of the server licence is specified as this enables a user to be tied to a particular licence.*

## Logging

Set the environment variable `DDT_LICENCE_LOGFILE` to the file that you wish to append log information to. Set `DDT_LICENCE_LOGLEVEL` to set the amount of information required. These steps must be done prior to starting the server.

Level 0: no logging.

Level 1: client licences issued are shown, served licences are listed.

Level 2: stale licences are shown when removed, licences still being served are listed if there is no spare licence.

Level 3: full request strings received are displayed

Level 6 is the maximum.

In level 1 and above, the MAC address, username, process ID, and IP address of the clients are logged.

## Troubleshooting

Licences are plain-text which enables the user to see the parameters that are set; a checksum verifies the validity. If problems arise, the first step should be to check the parameters are consistent with the machine that is being used (MAC and IP address), and that, for example, the number of users is as expected.

## Adding A New Licence

To add a new licence to be served, copy the file to the directory where the existing licences are served and restart the server. Existing clients should not experience disruption, if the restart is completed within a minute or two.

## Examples

In this example, a dedicated licence server machine exists but uses the same filesystem as the client machines, and DDT is installed at `/opt/software/ddt`

To run the `licenceserver` as `nobody`, serving all licences in `/opt/software/ddt`, and logging most events to the `/tmp/licence.ddt.log`.

```
% su - nobody
```

```
Password:
```

```
% export DDT_LICENCE_LOGFILE=/tmp/licence.ddt.log
```

```
% export DDT_LICENCE_LOGLEVEL=2
% cd /opt/software/ddt
% ./bin/licenceserver /opt/software/ddt/ &
% exit
```

Serving the floating licences from the same directory as a normal DDT installation is possible as the licence server will ignore licences that are not server licences.

If the server licence is file “/opt/software/Licence.server.physics” and is served by the machine server.physics.acme.edu, at port 4252, the licence would look like:

```
type=3
serial_number=1014
max_processes=48
expires=2004-04-01 00:00:00
support_expires=2004-04-01 00:00:00
mac=00:E0:81:03:6C:DB
interface=eth0
debuggers=gdb
serverport=4252
max_users=2
beat=60
retry_limit=4
hash=P5I: ?L,FS=[CCTB<IW4
hash2=c18101680ae9f8863266d4aa7544de58562ea858
```

Then the client licence could be stored at “/opt/software/Licence” and contain:

```
type=2
serial_number=1014
hostname=server.physics.acme.edu
serverport=4252
```

## Example Of Access Via A Firewall

SSH forwarding can be used to reach machines that are beyond a firewall, for example the remote user would start:

```
ssh -C -L 4252:server.physics.acme.edu:4242 login.physics.acme.edu
```

And a local licence file should be created:

```
type=2
serial_number=1014
hostname=localhost
serverport=4252
```

## Querying Current Licence Server Status

The licence server provides a simple HTML interface to allow for querying of the current state of the licences being served. Point your favorite web browser at a URL of the form:

```
http://<hostname>:<serverport>/status.html
```

For example, using the values from the licence file examples, above:

```
http://server.physics.acme.edu:4252/status.html
```

Initially, no licences will be being served, and the output in your browser window should look something like:

```
[Licences start]
  [Licence Serial Number: 1014]
    [No licences allocated - 2 available]
[Licences end]
```

As licences are served and released, this information will change. To update the licence server status display, simply refresh your web browser window. For example, after one DDT has been started:

```
[Licences start]
  [Licence Serial Number: 1014]
    [1 licences available]
      [Client 1]
        [mac=00:04:23:99:79:65; uname=gwh; pid=14007;
licence=1014]
        [Latest heartbeat: 2004-04-13 11:59:15]
[Licences end]
```

Then, after another DDT is started and the web browser window is refreshed (notice the value for number of licences available):

```
[Licences start]
  [Licence Serial Number: 1014]
    [0 licences available]
      [Client 1]
        [mac=00:04:23:99:79:65; uname=gwh; pid=14007;
licence=1014]
          [Latest heartbeat: 2004-04-13 12:04:15]
        [Client 2]
          [mac=00:40:F4:6C:4A:71; uname=graham; pid=3700;
licence=1014]
            [Latest heartbeat: 2004-04-13 12:04:59]
[Licences end]
```

Finally, after the first DDT finishes:

```
[Licences start]
  [Licence Serial Number: 1014]
    [1 licences available]
      [Client 1]
        [mac=00:40:F4:6C:4A:71; uname=graham; pid=3700;
licence=1014]
          [Latest heartbeat: 2004-04-13 12:07:59]
[Licences end]
```

## Licence Server Handling Of Lost DDT Clients

Should the licence server lose communication with a particular instance of a DDT client, the licence allocated to that particular DDT client will be made unavailable for new DDT clients until a certain timeout period has expired. The length of this timeout period can be calculated from the licence server file values for `beat` and `retry_limit`:

$$\text{lost\_client\_timeout\_period} = (\text{beat seconds}) * (\text{retry\_limit} + 1)$$

So, for the example licence files above, the timeout period would be:

$$60 * (4 + 1) = 300 \text{ seconds}$$

During this timeout period, details of the “lost” DDT client will continue to be output by the licence server status display. As long as additional licences are available, new DDT clients can be started. However, once all of these additional licences have been allocated, new DDT clients will be refused a licence while this timeout period is active.

After this timeout period has expired, the licence server status will continue to display details of the “lost” DDT client until another DDT client is started. The licence server will grant a licence to the new DDT client and the licence server status display will then reflect the details of the new DDT client.

## A. Supported Platforms

A full list of supported platforms and configurations is maintained on the Streamline Computing website. It is likely that MPI distributions supported on one platform will work immediately on other platforms.

<b>Platform</b>	<b>Operating Systems</b>	<b>MPI</b>	<b>Compilers</b>
Intel/AMD x86 AMD Opteron (32+64) Intel Itanium 2	Redhat 7 and above, SuSE, Debian, and similar	SGI Altix, Bproc, LAM-MPI, MPICH, Myricom MPICH-GM, Quadrics MPI, Scali MPI Connect, SCore, Scyld	GNU, Absoft, Intel and Portland
HP Itanium and PA- RISC	HP-UX 11.22 and 11.11	MP-MPI	Native
IBM Power	AIX 5.1 and above	IBM PE	Native
SGI MIPS	IRIX	SGI MP Toolkit	Native
Sun Ultrasparc	Solaris 8 and above	Sun Clustertools 4 and above	Native



## B. Troubleshooting DDT

If you should encounter any difficulties in using DDT, you should, in the first instance, view the FAQ (Appendix C).

You may find a problem with your DDT that has already been fixed by Streamline, please check the support pages of the Streamline Computing website for updates. If your problem persists, contact Streamline directly by emailing [ddt@streamline-computing.com](mailto:ddt@streamline-computing.com).

### Problems Starting DDT Frontend

If DDT is unable to start, this is usually one of three reasons:

- DDT is not in the PATH – the shell reports that DDT is not found. Change your path to include the \$DDTPATH/bin directory
- The licence is invalid – in this case DDT will issue an error message. You should verify that you have a licence file, that it is stored as \$DDTPATH/Licence, and check manually that it is valid by inspection. If DDT still refuses to start, please contact Streamline

### Problems Starting Scalar Programs

Please note that v1.5 onwards may erroneously report a problem with MPI when encountering a problem, instead of suggesting that your program cannot start.

There are a number of possible sources for problems. The most common is – for users with a multi-process licence – that the MPI implementation has not been set to “none”.

Other potential problems are

- A previous DDT session is still running, or has not released resources required for the new session. Usually this can be resolved by killing stale processes. The most obvious symptom of this is a delay of approximately 60 seconds and a message stating that not all processes connected. You may also see, in the terminal, a QServerSocket message
- The target program does not exist or is not executable
- The selected debugger cannot be found
- The selected debugger has crashed

- DDT's backend daemon – ddt-debugger – is missing from \$DDTPATH/bin – in this case you should check your installation, and contact Streamline for further assistance.

## Problems Starting Multi-Process Programs

If you encounter problems whilst starting an MPI program with DDT, the first step is to establish that it is possible to run a single-process (non-MPI) program such as a trivial “hello world” – and resolve such issues that may arise. After this, attempt to run a multi-process job – and the symptoms will often allow a reasonable diagnosis to be made.

In the first instance, verify that MPI is installed correctly by running a job outside of DDT, such as the example in \$DDTPATH/examples.

```
mpirun -np 8 ./a.out
```

Verify that mpirun is in the PATH, or the environment variable DDTMPIRUN is set to the full pathname of mpirun.

If the progress bar does not report that at least process 0 has connected, then the remote ddt-debugger daemons cannot be started or cannot connect to the frontend.

The majority of such problems are caused by environment variables not propagating to the remote nodes whilst starting a job. To a large extent, the solution to these problems depend on the MPI implementation that is being used. In the simplest case, for rsh based systems such as a default MPICH installation, correct configuration can be verified by rsh-ing to a node and examining the environment. It is worthwhile rsh-ing with the env command to the node as this will not see any environment variables set inside the .profile command. For example if your nodes use a .profile instead of a .bashrc for each user then you may well see a different output when running “rsh node env” than when you run “rsh node” and then run “env” inside the new shell.

If only one, or very few, processes connect, it may be because you have not chosen the correct MPI implementation. Please examine the list and look carefully at the options. Should no other suitable MPI be found, please contact Streamline for advice.

If a large number of processes are reported by the status bar to have connected, then it is possible that some have failed to start due to resource exhaustion, timing out, or, unusually, an unexplained crash. You should verify again that MPI is still working, as some MPI distributions do not release all semaphore resources correctly (for example MPICH on Redhat with SMP support built in).

To check for time-out problems, set the DDT\_NO\_TIMEOUT environment variable to 1 before launching the frontend and see if further progress is made. This is not a solution, but aids the diagnosis. If all processes now start, please contact Streamline for further long-term advice

## C. FAQs

### DDT will not load – what's wrong?

Check that you have the environment variables set up (see the installation README file) and that the DDT executable (\$DDT) exists.

If DDT will not load then you may be missing some libraries, or they may be incompatible with your Linux/Unix version – view the output of "ldd \$DDT" and look for the missing libraries. Contact Streamline for further assistance.

### Why can't DDT find my hosts or the executable?

Ensure that the hostname(s) given are reachable using ping, if not try using the IP addresses. If DDT fails to find the executables, ensure the executable is available on every machine, and – for the common MPICH distribution – that you can "rsh" to each host without the password prompt.

### Why am I getting an error message about the debugger when starting my job?

Your MPI distribution most likely is not correctly compiled to include a small amount of debugging information. An alternative cause is that the version of GDB you are using is not compatible; GDB version 5 and above has been tested, our favorites are 5.2 upwards, and the DDT distribution comes with a patched GDB 5.3 to improve Fortran array support. Ensure a supported version of GDB is in your path on all nodes.

### The progress bar doesn't move and DDT 'times out'

It's possible that the program 'ddt-debugger' hasn't been started by mpirun or has aborted. You can log onto your nodes and confirm this by looking at the process list BEFORE clicking 'Ok' when DDT times out. Ensure ddt-debugger has all the libraries it needs and that it can run successfully on the nodes using mpirun.

Alternatively, there may be one or more processes ('ddt-debugger', 'mpirun', 'rsh') which could not be terminated. This can happen if DDT is killed during its startup or due to MPI implementation issues. You will have to kill the processes manually, using 'ps x' to get the process ids and then 'kill' or 'kill -9' to terminate them.

This issue can also arise for mpich-p4mpd, and the solution is explained in Appendix E.

If your intended mpirun command is not in your path, you may either add it to your path or set the environment variable DDTMPIRUN to contain the full path of the correct mpirun.

## **The progress bar gets close to half the processes connecting and then stops and DDT 'times out'**

This is likely to be caused by dual processor configuration for your MPI distribution. Make sure you have selected 'smp-mpich' or 'scyld' as your MPI implementation in DDT's configuration window. If this doesn't help, see Appendix E for a workaround and contact us for further assistance.

## **I am using MPI\_Get\_processor\_name() and I get "Process n has stopped with signal SIGTRAP" when I click play**

Gdb has a known problem with fork() and gethostbyname() which causes it to crash your program. Many MPI implementations such as Scyld use fork() during MPI\_Init – any subsequent calls to MPI\_Getprocessor\_name() or gethostbyname() will not work under gdb. You can:

- Replace your MPI with a non-fork based one, e.g. MPICH (not -comm=shared)
- Use a different debugger, such as Absoft's Fx2
- Avoid using MPI\_Get\_processor\_name()/gethostbyname()

## **My program doesn't start, and I can see a console error stating "QServerSocket: failed to bind or listen to the socket"**

Ordinarily this message is not a sign of a problem – it is emitted when another DDT session, is running and consequently the DDT master uses another socket instead. However, if you know this not to be the case and your program is not starting, it's likely that a previous run of DDT has been unable to terminate and release resources completely. This is known to occur occasionally for MPICH-GM. If this happens, run /usr/bin/killall -9 ddt-debugger on your nodes – you can actually use mpirun to do this for you.

## **Why can't I see any output on stderr?**

DDT automatically captures anything written to stdout/stderr and displays it. Some shells (such as csh) and debuggers (such as dbx on Solaris) do not support this feature in which case you may see your stderr mixed with stdout, or you may not see it at all. In any case we strongly recommend writing program output to files instead, since the MPI specification does not cover stdout/stderr behaviour.

## **DDT complains about being unable to execute malloc**

Should this error message occur, often due to backend-debugger failure, it is possible to bypass this step. Set the environment variable DDT\_DONT\_GET\_RANK to any non-empty value on the nodes and this will force DDT to guess ranks, which may resolve the problem.

## **Why can't I use watches with the Intel IDB interface?**

This is a known issue with Intel's IDB debugger on several systems, please contact Streamline for more information/assistance.

## **Why is my stack trace empty/incomplete?**

This is a known issue with Intel's IDB on several systems and GDB on the Opteron architecture, contact Streamline for more information/assistance.

## **Some features seem to be missing (e.g. watch points) – what's wrong?**

This is because not all debugger's support every feature that DDT does and so they are disabled by removing the window/tab by from DDT's interface. For example if you are using Intel's IDB debugger then the watches tab has been removed as this debugger doesn't support watches.

## **My code does not appear when I start DDT**

This is probably due to the currently selected font not being correctly installed on your system. Go into the Session – Configuration window and choose a font such as Times or Century Schoolbook and you should now be able to see the code.

## When I use step out my program hangs

You cannot use the step out feature from the Main function of your program. This will cause the debugger to hang. With some debuggers DDT will return a time out error. Make sure you only use step out inside loops and functions.

## When viewing messages queues after attaching to a process I get a “Cannot find Message Queue DLL” error

This is due to the fact that you have started your MPI process outside of DDT. The message queue process cannot then find which DLL to use for the version of MPI that you have started. The way to fix this is to set the variable DDT\_QUEUE\_DLL explicitly before you start DDT.

Example: “export DDT\_QUEUE\_DLL=/usr/local/mpich/lib/libtvmppich.so”

The files needed for LAM and MPICH are listed here:

- Lam – liblam\_totalview.so
- MPICH – libtvmppich.so

## I get the error `The mpi execution environment exited with an error, details follow: Error code: 1 Error Messages: “mprun:mpmd\_assemble\_rsrcs: Not enough resources available”` when trying to start DDT

This error occurs when running DDT on a Solaris machine. If you select more processes than you have processors in your machine then mprun is not able to allocate the resources needed. To fix this simply add the argument `-W` to the `MPI Arguments` box, this will tell mprun to wrap the processes and will enable you to start your desired number of processes in DDT.

## What do I do if I can't see my running processes in the attach window?

This is usually a problem with either your remote-exec script or your node list file. First check that the entry in your node list file corresponds with either localhost (if you're running on your local machine) or with the output of `hostname` on the desired machine.

Secondly try running remote-exec manually ie. `remote-exec ls` and check the output of this. If this fails then there is a problem with your remote-exec script. If `rsh` is still being used in your script check that you can rsh to the desired machine. Otherwise check that you can attach to your machine in the way specified in the `remote-exec` script. If you still experience problems with your script then contact Streamline for assistance.

## When trying to view my Message Queues using mpich I get no output but also see no errors

This is a known problem on the Opteron system with 64/32bit compatibility. The 64bit library that is built with mpich on the Opteron does not return any data that DDT can interpret. Try copying in a library built on a compatible system such as Redhat 9 and setting the DDT\_QUEUE\_DLL argument to point to this library.

If you need a copy of this library but cannot build one please contact Streamline.

## Obtaining Support

If you are unable to resolve your problem, the most effective way to get support is to email Streamline with a detailed report of your problem. If possible, you should obtain a log file for the problem and email this to Streamline.

Generating a log file is simple. Firstly the DDTLOG environment variable must be given the name of a file to write to. Secondly, start DDT with a -debug flag. For example, bash users would type:

```
export DDTLOG=$HOME/ddt.log
ddt -debug
```

The user should then attempt to demonstrate the problem, and quit or kill DDT.



## D. Debugging Fortran

How do I view the contents of an array which is given as a parameter to a subroutine and appears as "PTR TO ..." in the locals window?

*Note: This only occurs with gdb and the g77 compiler, as far as we know.*

Use the Multi-dimensional array viewer BUT you must ensure the expression you type dereferences the array before you use it.

Example:

If your array is A and it appears as, say, PTR TO REAL\*4(4,4,-1), in the locals window, the expression you type in must first of all dereference A – otherwise gdb will produce a segmentation fault

To view the contents of A(1,2,3) you should type in

```
(*A) (1,2,3)
```

and click "evaluate"

### Why is getting local variables so slow?

Firstly please check that the patched gdb supplied with this distribution is being used in preference to the default gdb.

DDT parses the response to gdb's "info locals" command, and this can frequently produce massive responses for Fortran codes. If your local variables are still slow, try viewing only the current line and putting the variables that you are interested in in the evaluate window instead.

### Why are there patched gdb's in the DDT distribution?

A disagreement between the data description format of the Intel compilers and the g77 compiler for multi-dimensional arrays has led to the situation where the format descriptions are the reverse of each other.

Unfortunately, standard gdb produces the correct result for g77 for some of the command set, and the correct result for Intel for the remainder. We therefore provide one version that produces the correct result for g77 all the time, and recommend that idb is used for Intel compiled Fortran codes. You can verify the problem on your existing gdb by viewing a REAL(5\*4) array 'x' and examining the output of the three commands:

```
print x
print x(1,2)
whatis x
```

under the Intel compiler and then the g77 compiler.

Our patches also improve Portland support (see B.4), and fix a commonly occurring segmentation fault when viewing the parameters of a Fortran subroutine.

## E. Notes On MPI Distributions

This appendix has brief notes on many of the MPI distributions supported by DDT. Advice on settings and problems particular to a distribution are given here.

### Bproc

By default, the p4 interface will be chosen. If you wish to use GM (Myrinet), place `-gm` in the MPIrun arguments, and this will be used instead. Select Generic as the MPI implementation.

### HP MPI

Select HP MPI as the MPI implementation.

### LAM/MPI

No reported issues with this distribution. Select LAM-MPI as the MPI implementation.

### MPICH And SMP Nodes

This issue affects some distributions where shared memory is used to communicate between processors on a dual processor machine. For `mpich-p4` this will only affect you if your configuration of `mpich` specified `-comm=shared`.

Under these circumstances the dual CPUs use a different starting mechanism for `mpirun`. We recommend selecting the ``smp-mpich`` or ``scyld`` implementation from DDT's configuration window as appropriate. If this does not solve your problem, or you are using an unsupported MPI implementation then you can try setting `MPI_MAX_CLUSTER_SIZE=1`. This will still allow you to use a large cluster, but it will fool MPI into using `rsh/ssh` instead of `fork` to start jobs. It will still use all available cpus, for MPICH you could do this by specifying a dual processor machine TWICE in the `machines.LINUX` file instead of specifying `hostname:2`.

### MPICH p4

No reported issues with this distribution, choose MPICH as the MPI implementation.

## MPICH p4 mpd

This daemon based distribution passes a limited set of arguments and environments to the job programs. If the daemons do not start with the correct environment for DDT to start, then the environment passed to the ddt-debugger backend daemons will be insufficient to start.

It should be possible to avoid these problems if .bashrc or .tcshrc/.cshrc are correct. However, if unable to resolve these problems, you can pass DDTPATH, HOME and LD\_LIBRARY\_PATH, plus any other environment variables that you need, such as LM\_LICENSE\_FILE if you're using the Portland debugger, manually. This is achieved by adding `-MPDENV- DDTPATH={insert ddtpath here} HOME={homedir} LD_LIBRARY_PATH={ld-library-path}` to the "program arguments" area of the run dialog. Alternatively from the command line you may simply write:

```
$DDT {program-name} -MPDENV- HOME=$HOME DDTPATH=$DDTPATH
LD_LIBRARY_PATH=$LD_LIBRARY_PATH
```

and your shell will fill in these values for you.

Choose MPICH as the MPI implementation.

## MPICH-GM

No reported issues with this distribution. Select Generic as the MPI implementation.

## IBM PE

If you are able to use poe outside of a queuing system, set the environment variable DDTMPIRUN to the full pathname of poe. If your poe does not take the standard mpirun arguments (eg. `-np xx`), it is advisable to write a wrapper script called mpirun which will invoke poe with the arguments you want.

In the present release of DDT, it is necessary to set the DDT\_DONT\_GET\_RANK variable to 1 for MPI debugging. Without this, processes will not be able to start.

A sample Loadleveller script, which starts debugging jobs on IBM AIX (POE) systems is included in the \$DDTPATH/templates directory. When working with Loadleveller, it is necessary to set the environment variable DDT\_IGNORE\_MPI\_OUTPUT to 1.

In order to view source files it is important to have bash and gdb in your path. GDB is provided with the DDT distribution. Bash can be installed locally if not on your system, and is available from [ftp.gnu.org](http://ftp.gnu.org).

Select IBM PE as the MPI implementation.

## NEC MPI

Select Generic as the MPI implementation.

## Quadratics MPI

Select Generic as the MPI implementation.

## SCore

DDT is supported by SCore versions 5.6.0 and above, a patch is available for SCore 5.40. DDT can be launched either within a scout session, or using a queue. Presently an omission in SCore prevents arguments being passed to programs; we expect a patch to be issued imminently – contact Streamline if this issue affects you.

There are several methods to start DDT on an SCore system and your administrator should recommend one for use with your cluster. Streamline recommend using a Sun GridEngine and provide a queue template file for this system. However, we have found the following method to work on single-user mode clusters:

- 1) Make sure your home directory is mounted on each cluster node
- 2) Create a host file containing a list of the computer nodes your intend to run the job on
- 3) Start a scout session: `scout -F host.list`
- 4) Start DDT at the prompt: `ddt`
- 5) Make sure DDT is configured for SCore mode, with the correct number of processes. Use the MPI Argument ``nodes=MxN`` to specify the number of processes per node and number of nodes, as documented for `scrunch`. Make sure to multiply these numbers when selecting the number of processes for DDT! Both must be specified for single-user mode Score systems
- 6) Click on ``Start``

Note that the first release of Score 5.6.0 shipped with a flaw in `scrunch.exe` – this prevents DDT shutting down a job correctly. The scout session must be closed and

reopened between DDT sessions on these systems. This only affects single-user mode Score 5.6.0 installs.

If environment variables are not being propagated to remote nodes, we suggest moving `$DDTPATH/bin/ddt-debugger` to `$DDTPATH/bin/ddt-debugger.bin`, and creating a replacement executable shell script which sets the correct environment variables before running `ddt-debugger.bin` - for example:

```
#!/bin/sh
. ./bashrc
$DDTPATH/bin/ddt-debugger.bin $*
```

Choose SCore as the MPI implementation.

## Scyld

When running under Scyld, DDT starts all its `ddt-debugger` processes on the local machine instead of on the nodes. This is because Scyld represents the cluster as a single system image. For all but the largest clusters this should not be a problem. If this is an issue for you (insufficient file handles etc.) then contact Streamline for additional assistance.

The process details window will not show any hostnames when running under Scyld. This should not matter because Scyld represents a cluster as a single system image.

Choose Scyld as the MPI implementation.

## SGI Altix/Irix

Some versions SGI's MP Toolkit can cause GDB to crash due to a library problem. This is easily resolved if it occurs. Compile your application with the extra linking flag `"-lrt"` and try DDT again. SGI MP Toolkit should be chosen from the MPI implementations list.

## F. Notes On Debuggers

Always compile with a minimal amount of, or no, optimization – many compilers reorder instruction execution and omit debug information when compiled with optimization turned on.

Some MPI implementations such as MPICH 1.2.5, require you to compile your code with the same compiler family as the implementation was compiled with. For example, if your copy of MPICH was compiled up using the Intel compilers, you should also compile your programs using the Intel compilers. Debugging with a mix of compilers may be possible in some cases but this is not supported or recommended.

### Absoft

DDT supports the new Absoft fx2 debugger. Fx2 debugger does not support multiple threads at this time. DDT has been tested with version 7 and 8 of the Absoft FORTRAN 77/90/95 compilers.

### GNU

Choose GNU (GDB) interface. Note that fortran arrays within subroutines may appear as `PTR TO`, see section 3. DDT has been tested with gcc/g77 and comes with a patched gdb version 5.3/6 – which corrects some bugs relating to fortran.

### Intel Compilers

Choose the Intel (IDB) interface. It is important to have a recent idb, with a build date of 2003-03-03 or greater. Versions of 2003-06-10 and beyond have improved performance and fewer bugs than the 2003-03-03 version. Users of Intel compilers should be able to obtain a recent IDB from Intel easily, and the binary can be installed by the user in a local path.

DDT has been tested with icc/ifc version 7.1 and above. Some versions of IDB handle watch and stack information incorrectly, which DDT cannot always correct for. At times it may be impossible to view the full stack trace. Streamline do not recommend using watches with the IDB interface until this is resolved as they can cause your debug session to become unusable.

Some earlier releases of IDB are unable to limit printing arrays and will print the whole array instead of a sensible sized prefix. For huge arrays this may be a problem. By keeping the `local variables` panel concealed, this will limit the number of times this issue arises.

On some systems IDB takes an unusually long time to return a list of source files to DDT (several minutes) – particularly for older versions of IDB. It is possible to work around the problem by setting the environment variable DDT\_IDB\_MANUAL\_FILES to 1 – this will prevent IDB from building file lists automatically but when your job starts there will be no files in the file tree. You can add these manually by right-clicking on the file tree and choosing to add source directories. You may need to refresh DDT by double-clicking on the current process before the source file appears with correctly-highlighted lines of code. We expect future versions of IDB to resolve this.

## Portland Group Compilers

To use the Portland compilers with GDB, recompile your code with `-g -Mstabs` options for compatibility. Owners of the Portland Cluster Development Kit (CDK) may also use PGDBG to debug a cluster.

DDT has been tested with Portland Tools 4 and 5.1-3.

When using PGDBG as the backend debugger it is important to ensure that the -Munroll and -fast options are **not** used to compile your MPI distribution (this is a known problem with PGDBG and the 1.2.x series of MPICH) as PGDBG is unable to execute a step-out which is necessary during MPI startup when this optimization is turned on.

## Solaris DBX

A bug in some versions of DBX prevents redirection of stdout/stderr, which means that DDT cannot display the program's output. If your version of DBX is affected then Streamline recommend writing output to a file instead.

Watches are not supported by this debugger in multi-threaded codes – which includes all parallel codes using Sun Clustertools – consequently DDT cannot support watches when using DBX.

DDT has been tested on Solaris 5.8 and 9 with the Forte Developer 7 tools suite.



## HP-UX And wdb

DDT has been tested to work with the gdb that is supplied with wdb v4.0. This is available from <http://www.hp.com/go/wdb>. Ensure that this gdb is in your path and select gdb as the interface.

## G. Architectures

### AMD Opteron 64-Bit

This architecture requires a very recent GDB, including some patches (from SUSE) that have yet to be included at the main archive sites.

Without these recent GDBs, you may experience a loss of stack trace information in the `select()` I/O function and consequently DDT may not run. DDT will use the 64-bit GDBs included with DDT (filenames ending with `.64`).

### SGI Altix 3000

Some versions SGI's MP Toolkit can cause GDB to crash due to a library problem. This is easily resolved. Compile your application with the extra linking flag `"-lrt"` and try DDT again.

### IBM AIX Systems

In the present release of DDT, it is necessary to set the `DDT_DONT_GET_RANK` variable to 1 for MPI debugging. Without this, processes will not be able to start.

A sample Loadleveller script, which starts debugging jobs on IBM AIX (POE) systems is included in the `$DDTPATH/templates` directory.

In order to view source files it is important to have `bash` and `gdb` in your path. GDB is provided with the DDT distribution.

## Index

- Absoft, 55
- AIX, 41, 52, 57
- Align Stacks, 24
- Altix, 41, 54, 57
- Bproc, 51
- Breakpoints, 22
  - Conditional Breakpoints, 22
  - Deleting A Breakpoint, 23
  - Loading, 23
  - Saving, 23
  - Suspending Breakpoints, 23
- Core File, 10
- Cross-Process Comparison, 29
- Data
  - Changing, 27
- Debug flag, 47
- Debugger, 15
- Dynamic Libraries, 18
- FAQ, 44
- Finding Code Or Variables, 18
- Help, 7
- Hotkeys, 21
- HP MPI, 51
- HP-UX, 56
- IBM PE, 52
- Input, 32
- Installation, 6
- Intel Compilers, 55
- Irix, 54
- Jobs
  - Cancelling, 15
  - Regular Expression, 14, 15
  - Starting, 15
- Jump To Line, 19
  - Double Clicking, 20
- LAM/MPI, 51
- Licence Server, 36
- Log File, 47
- Message Queues, 34
- MPI rank, 20
- MPICH
  - GM, 52
  - p4, 51
  - p4 mpd, 51
  - SMP, 51
- Multi-Dimensional Arrays, 27
- NEC MPI, 52
- OpenMP, 5, 25
- Opteron, 16, 41, 46, 47, 57
- Pointers, 27
- Portland Group, 56
- Process
  - Control, 20
  - Groups, 20
- Process Group
  - Adding A Process, 20
  - Creating, 20
  - Deleting, 20
- Pthreads, 25
- Quadratics MPI, 52
- Queue
  - Configuring, 13
- Queuing, 13
- Raw Command, 31
- Registers
  - Viewing, 30
- Restarting, 21
- SCore, 9, 41, 52, 53
- Scyld, 53
- Session
  - Loading, 18
  - Saving, 18
- SGI, 54
- Single-Process
  - Multi Process Licence, 10
  - Single Process Licence, 10
- Solaris, 9, 11, 41, 45, 47, 56
- Solaris DBX, 56

Source Code, 18	Stepping Through A Program, 22
Source Files	Stopping, 21
Find Missing, 18	Sun Clustertools, 41, 56
Stack Frame, 24	Support, 7
Standard Error, 32	Synchronizing Processes, 23
Standard Output, 32	Threads
Starting, 21	Examining, 25
Starting DDT, 8	Visualizing Data, 28