FUJITSU

FUJITSU Software
Technical Computing Suite V2.0

Debugger User's Guide (PRIMEHPC FX100)

J2UL-1897-02ENZ0(00)
November 2015

Preface

Purpose

This guide describes the features of the interactive Debugger (referred to as "the Debugger" in this guide) and explains how to use it for Technical Computing Suite (referred to as "this system" in this guide).

The Debugger is provided with the FUJITSU Software Development Tools (FSDT), which is a GUI development environment. Refer to the "fdb manual", displayed by executing the man command, for information on using fdb commands on the command line.

This guide applies to the Debugger used on the Linux operating system.

Intended Readers

This guide is intended for those who want to debug programs using the Debugger. It is assumed that readers of this guide have working knowledge of program development operations and related command operations in Linux.

Organization of This Guide

This manual is organized as follows:

Chapter 1 Overview of the Debugger

Provides an overview of the functionality of the Debugger

Chapter 2 Debugger Features

Describes the Debugger features that are commonly used while debugging

Chapter 3 Debugger Windows and Menus

Describes the elements in Debugger windows

Chapter 4 List of Debugger Operations

Lists the various debugging operations with their respective operating instruction

Appendix A Considerations for Using the Debugger

Describes the key points to consider when using the Debugger

Appendix B Notes on Migration from FX10 System to FX100 System

Describes notes on migrating from FX10 system to FX100 system

Appendix C Compatibility Information (FX10 System)

Describes compatibility information as notes on migrating

Appendix D Compatibility Information (FX100 System)

Describes compatibility information as notes on migrating

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Trademarks

- Linux is a registered trademark or trademark of Linus Torvalds in the United States and other countries.
- OpenMP is a trademark of OpenMP Architecture Review Board.
- Other trademarks and registered trademarks are trademarks or registered trademarks of their respective owners.
- Trademark symbols (TM, (R)) are not necessarily added to system name or product name, etc. published in this material.

Date of Publication and Version

Version	Manual code
November 2015, 2nd Version	J2UL-1897-02ENZ0(00)
February 2015, Version 1.1	J2UL-1897-01ENZ0(01)
October 2014, 1st Version	J2UL-1897-01ENZ0(00)

Copyright

Update History

Changes	Location	Version
The article "Debugging COARRAY feature" is added.	Appendix A	2nd Version
The article "Debugging in environment with effective job swap function" is added.	Appendix A	
Fixed the error in writing.	-	
The article "Using the tool runtime daemon for the Debugger" is changed.	Appendix A	Version 1.1
The article "Display the source file on source view panel" is changed.	Appendix A	
The article "The how of debugging by fdb command" is added.	Appendix A	
The article "Start condition of Debugger" is added.	Appendix A	
"Notes on migrating from FX10 system to FX100 system" is added.	Appendix B	
Compatibility information is added.	Appendix C Appendix D	

Contents

Chapter 1 Overview of the Debugger

This chapter provides an overview of the Debugger, explaining its functionality.

The Debugger enables debugging of Fortran, C/C++, MPI, and XPFortran programs for which a job is submitted to this system.

For normal debug and corefile debug of MPI and XPFortran programs, up to 192 parallel processes of MPI programs can be debugged.

If the number of parallel processes of programs to be debugged is more than 192, limit the count to 192 or less, and then debug it.

1.1 Debugger Features

The Debugger provides the following three debugging modes.

- Normal debug

  Normal debug submits a job using the debugging feature of the FUJITSU Software Development Tools (FSDT) with the user terminal, executing from the start of a program and debugging it. It displays the expressions and variables of a program, controls the execution, and also sets the execution stop position while debugging.

- Corefile debug

  Corefile debug statically verifies the abnormal termination state by using the corefile that is output when a job terminates abnormally. The Debugger can handle only one corefile at a time.

- Job ID attach debug

  This captures all the processes of a job by specifying the job ID.

  Job ID attach debug is useful if you want to verify the state of a program when the active job specified by the job ID does not exit.

Additionally, using the Debugger GUI, you can perform the following operations for Fortran, C/C++, MPI, and XPFortran programs.

- Control a program's execution

  It is possible to restart a program and run it till the next stop position, or execute it line-by-line.

- Set the execution stop position for a program

  Breakpoints, watchpoints, and barrierpoints can be set.

- Display the expressions and variable values

  The expressions, variable values, and local variables can be displayed.

- Display and select the stack trace

  By displaying the stack trace and selecting the stack position, a function call can be traced.

The command that displays the debug information of the debugging engine (fdb) can also be executed by using the command line interface from the GUI.

1.2 Operating Environment of the Debugger

Debugging processes using the Debugger involves compiling programs using the Fujitsu compiler and executing jobs using the job operation software on this system.

Therefore, a correct environment for compilation and job execution is necessary.

Note that the Debugger is started from FSDT, which is the GUI support environment.

This guide explains how to use the Debugger after it has been started from FSDT. Refer to the "*Programming Workbench User's Guide*" for information on FSDT operations.

Additionally, refer to the chapter of the launcher feature in the "*Programming Workbench User's Guide*" for information on how to start the Debugger using the launcher feature of FSDT.

This guide relates to the following manuals, and so you may also reference these manuals.

- *Fortran Language Reference*

- *Fortran User's Guide*
- *Fortran Compiler Messages*
- *C User's Guide*
- *C++ User's Guide*
- *C/C++ Compiler Optimization Messages*
- *MPI User's Guide*
- *XPFortran User's Guide*
- *Programming Workbench User's Guide*
- *Job Operation Software First Step Guide*
- *Job Operation Software End User's Guide*

1.3 Program Compilation

Program compilation

Compile using the Fortran, C, or C++ processing system, and specify the -g option on compilation.

The -g option generates the debug information. If this option is not specified, the variable values cannot be referenced as there is no correspondence with the source program while debugging it.

Refer to the "*Fortran User's Guide*", the "*C User's Guide*", the "*C++ User's Guide*", and the "*MPI User's Guide*" for information on this option and program compilation.

Debugging XPFortran programs

The Debugger cannot debug an XPFortran program as it is.

An XPFortran program is compiled to an MPI program by the XPFortran compile and linkage command (xpfrtpx), and then executed by the MPI program execution command (mpiexec).

A user debugs an XPFortran program as if debugging an MPI program while referring to the Debug/Tuning Support Data generated by the XPFortran compile and linkage command.

Refer to the "*XPFortran User's Guide*" for information on how to compile an XPFortran program, and how to generate the Debug/Tuning Support Data.

1.4 Job Submission

The Debugger debugs a job submitted on the job operation software of this system.

For the job submission command, pjsub, specified by normal debug, the Debugger adds the -X option.

When this option is specified, all environment variables, including the environment variable necessary for the Debugger to start, are forwarded with the batch request.

Chapter 2 Debugger Features

This chapter explains the Debugger features that are commonly used while debugging.

Refer to "Chapter 3 Debugger Windows and Menus" and "Chapter 4 List of Debugger Operations" for information on the windows and operation menus in the Debugger.

The Debugger provides three debugging modes that resolve the following problems:

- Incorrect results of a program
- Abnormal termination of a program
- Active program does not terminate (for example, enters an endless loop or a deadlock)

Normal debug (Incorrect results of a program)

The normal debug mode submits a job from the user terminal of the Debugger, executes a program from the top, displays the expressions and variables of the program, controls the program's execution, and sets the execution stop position.

As this mode debugs a program by executing it from the top, it is ideal for verifying the logic of a program with incorrect execution results. The user executes the source code lines to confirm a program's behavior. The program is executed line-by-line or till the stop position set for a program, enabling you to verify the logic and detect the reason for incorrect execution results.

Corefile debug (Abnormal termination of a program)

The corefile debug mode statically verifies the state at program termination by using the corefile that is output when a job abnormally terminates.

The user submits a job after setting up the program to output the corefile in case of abnormal termination. If the job terminates abnormally, the corefile is generated, and then forwarded to the login node for collection. Refer to the description of job staging in the "*Job Operation Software End User's Guide*" for information on corefile collection. The staging function appends the rank number of a process to the corefile name and forwards that corefile. Using this rank number, you can identify the process that generated the corefile. After the corefile is collected, the Debugger is started, the corefile, the program, and working directory are specified on the **Corefile debug** tab, and the abnormal termination state is analyzed. Two or more corefiles cannot be specified at the same time. The analyzed content contains the variable values, the process ID, the generated signal, the part (line/instruction) that caused abnormal termination, and the register (including the expanded register) and the space map information of the process.

Job ID attach debug (Active program does not terminate)

This captures all the processes of a job by specifying the job ID.

Job ID attach debug is useful if you want to verify the state of a program when the active job specified by the job ID does not exit.

When debugging starts, the program stops at a source code line as soon as the Debugger detects the program to be debugged. You may then start debugging the program by controlling the program's execution by using stop positions or executing the program line-by-line, and verify the program's behavior. Note that when the Debugger detects the program to be debugged, you can verify the values of variables and also check if there is a deadlock.

After debugging, you can either forcibly end or continue running the job.

2.1 Setting of Stop Positions

The Debugger can stop the program execution at any point in a program. You can stop a program, and perform various operations, such as display variable values, to confirm whether the program is operating as intended. By default, the stop position setting is enabled during debugging, but you may disable it if you do not want to stop a program.

The following stop positions are available.

Breakpoints

A breakpoint determines where to temporarily stop in a program while debugging.

If a breakpoint is set for all processes in a program, the process execution temporarily stops in an MPI program when an individual process reaches the breakpoint.

Additionally, if a breakpoint is set for a process with threads and if a thread reaches a breakpoint, each thread in the process stops executing temporarily.

The Debugger also provides the "Temporary breakpoint" that works only once.

Barrierpoints

A barrierpoint is the stop position that only applies to processes with threads.

When all threads in a process reach the barrierpoint, the execution is temporarily stopped. The Debugger cannot be operated for the process till all threads reach the barrierpoint.

Watchpoints

A watchpoint focuses on a specific variable and temporarily stops the program execution when the variable is accessed (referenced, changed, or referenced and changed). Note that though watchpoints are a powerful feature for monitoring variable access, they compromise the performance because they also monitor memory access.

2.2 Execution Control

2.2.1 Execution control for the temporary stop state

The methods for executing a program that has temporarily stopped are described below.

Continue

The program execution is resumed. The program continues execution till it reaches a stop position, or it terminates.

Step

The current line is executed. For a function call or a procedure call, the execution stops at the first line of the called function or procedure.

Next

The current line is executed. For a function call or a procedure call, the function or the procedure is executed as one sentence, and then the execution stops at the next line. The execution does not stop at a stop position, such as a breakpoint, till it returns from the function/ procedure call, even if there are breakpoints within the called function/procedure.

Finish

The program is executed till the return of a current function or procedure. After the current function or procedure returns, the program execution stops at the first line immediately after the function/procedure call. The execution does not stop at a stop position, such as a breakpoint, till it returns from the function/procedure, even if there are breakpoints within the function/procedure.

Rerun

The program is re-executed from the top. The information regarding breakpoints set for the group is not changed.

2.2.2 Execution control for active programs

The methods for stopping an active program are described below.

Stop

The program execution is temporarily interrupted. Debug operations, such as display of variables and restarting program execution, can be continued.

Kill

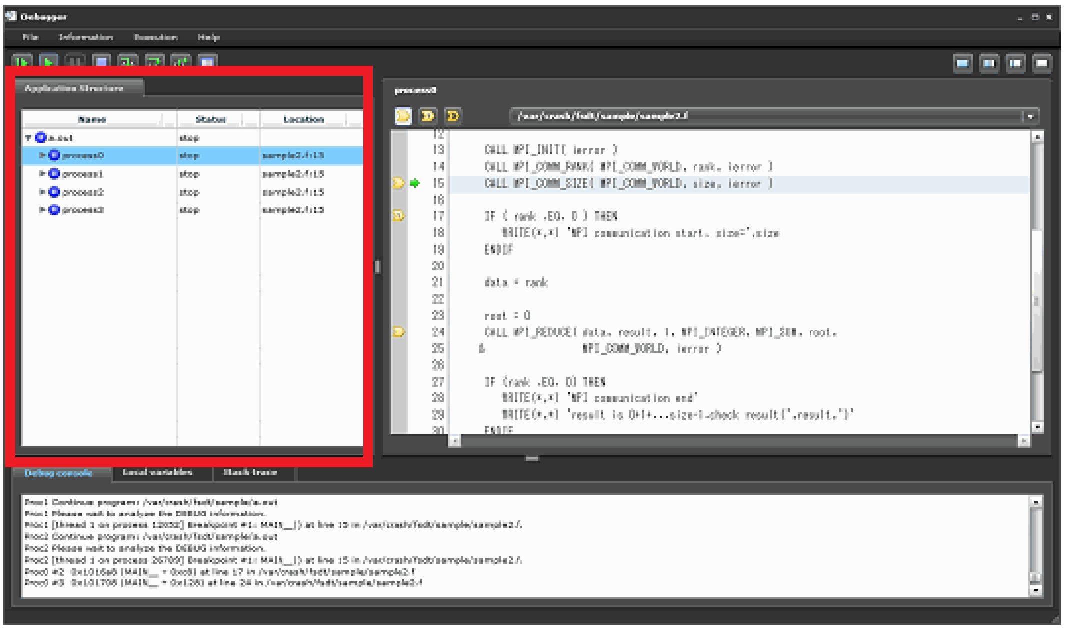
The program execution is interrupted, and the program terminates. The program execution cannot be controlled in this case. Use "Rerun" to resume debugging.

2.2.3 Specifying the debugging object

Using the Debugger, you can debug an entire program, a process, or a thread.

The program structure is displayed in the **Application structure** panel in the main window.

Figure 2.1 Main window (Application structure panel)



Name	Status	Location
a.out	stop	
process0	stop	sample2.f:15
process1	stop	sample2.f:15
process2	stop	sample2.f:15
process3	stop	sample2.f:15

```
      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
         WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
         WRITE(*,*) 'MPI communication end'
         WRITE(*,*) 'result is 0+1+...size-1.check result(',result,')'
      ENDIF
```

```
Proc1 Continue program: /var/crash/fxdt/sample/a.out
Proc1 Please wait to analyze the DEBUG information.
Proc1 [thread 1 on process 12032] Breakpoint #1: MAIN__() at line 15 in /var/crash/fxdt/sample/sample2.f.
Proc2 Continue program: /var/crash/fxdt/sample/a.out
Proc2 Please wait to analyze the DEBUG information.
Proc2 [thread 1 on process 26709] Breakpoint #1: MAIN__() at line 15 in /var/crash/fxdt/sample/sample2.f.
Proc0 #2  0x101016a8 (MAIN__ + 0xc8) at line 17 in /var/crash/fxdt/sample/sample2.f
Proc0 #3  0x10101708 (MAIN__ + 0x128) at line 24 in /var/crash/fxdt/sample/sample2.f
```

The item selected in the **Application structure** panel is called the active entity, and it is the object for debugging.

Note that the menus that can be used differ depending on the active entity object (an entire program, a process, or a thread).

2.3 Display Feature

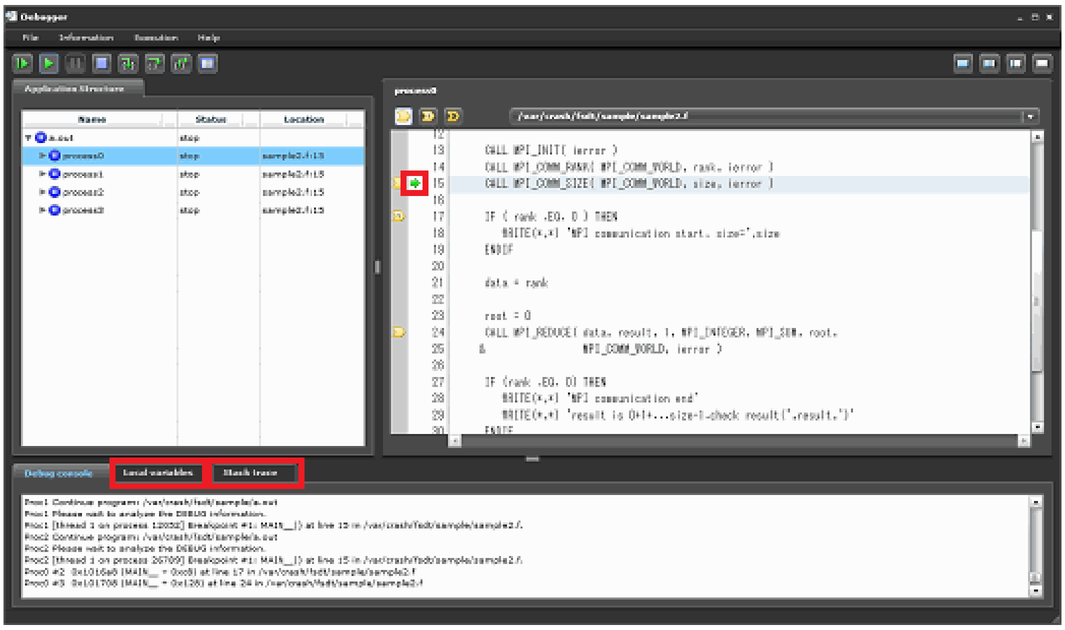
When a program's execution temporarily stops, the Debugger displays the following information:

- Position where the program stopped (a single process or a single thread is displayed)
- Stack trace information (trace back)
- Local variables (values at the time of a stop are displayed)

In the main window where debugging is performed, information on the active entity object is displayed.

Use the **Entity information** window (that displays information on processes and threads) to view the stack trace information and the local variable information for a process or a thread that is not currently active.

Figure 2.2 Main window (stop position, stack trace information, and local variables)

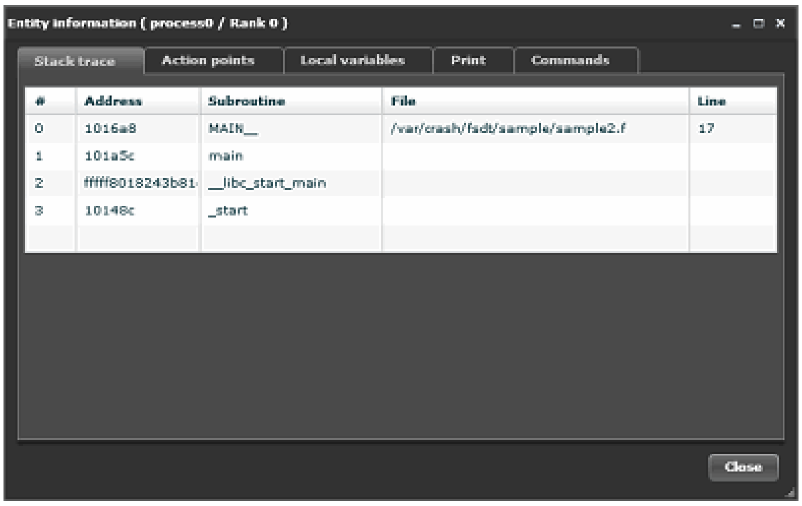


Name	Status	Location
a.out	stop	
process0	stop	sample2.f:15
process1	stop	sample2.f:15
process2	stop	sample2.f:15
process3	stop	sample2.f:15

```
      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
         WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
         WRITE(*,*) 'MPI communication end'
         WRITE(*,*) 'result is 0+1+...size-1-check result(',result,')'
      ENDIF
```

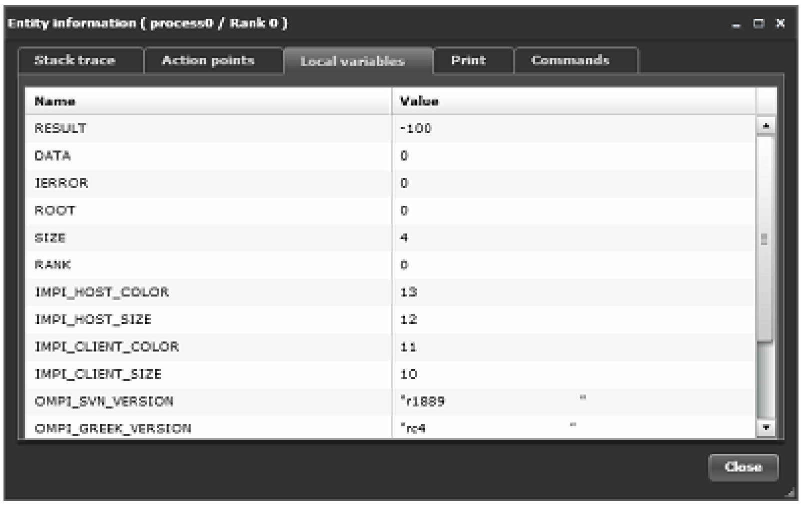
```
Proc1 Continue program: /var/crash/fsdt/sample/a.out
Proc1 Please wait to analyze the DEBUG information.
Proc1 [thread 1 on process 12032] Breakpoint #1: MAIN__() at line 15 in /var/crash/fsdt/sample/sample2.f.
Proc2 Continue program: /var/crash/fsdt/sample/a.out
Proc2 Please wait to analyze the DEBUG information.
Proc2 [thread 1 on process 26709] Breakpoint #1: MAIN__() at line 15 in /var/crash/fsdt/sample/sample2.f.
Proc0 #2  0x1016a8 (MAIN__ + 0xc8) at line 17 in /var/crash/fsdt/sample/sample2.f
Proc0 #3  0x101708 (MAIN__ + 0x128) at line 24 in /var/crash/fsdt/sample/sample2.f
```

Figure 2.3 Entity information window (stack trace information)



#	Address	Subroutine	File	Line
0	1016a8	MAIN__	/var/crash/fsdt/sample/sample2.f	17
1	101a5c	main		
2	fffff8018243b81	__libc_start_main		
3	10148c	_start		

Figure 2.4 Entity information window (local variables)

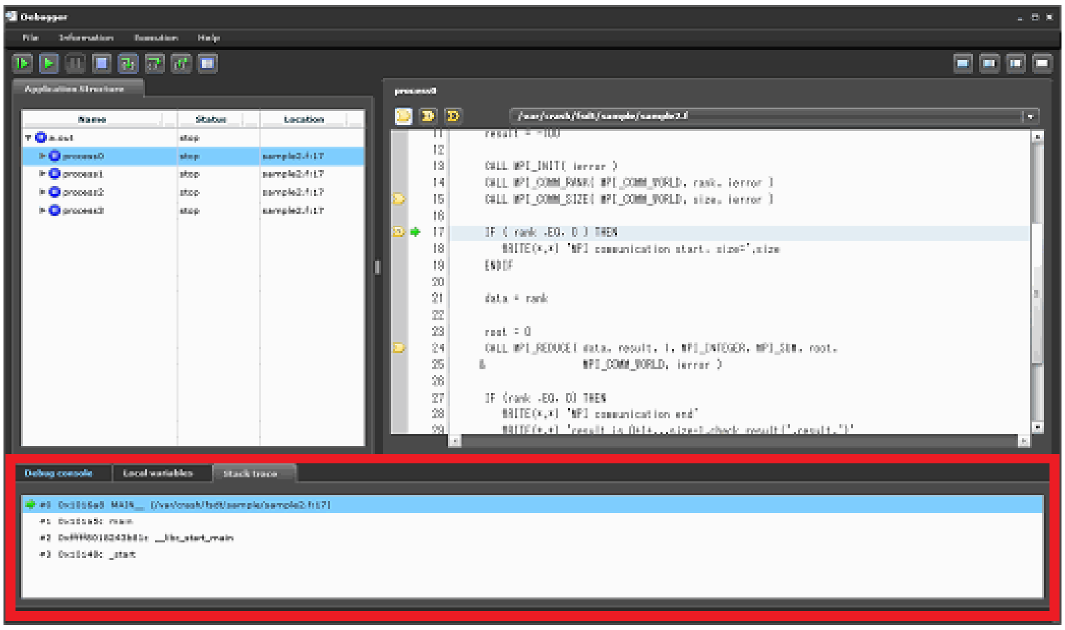


Name	Value
RESULT	-100
DATA	0
IERROR	0
ROOT	0
SIZE	4
RANK	0
IMPI_HOST_COLOR	13
IMPI_HOST_SIZE	12
IMPI_CLIENT_COLOR	11
IMPI_CLIENT_SIZE	10
OMPI_SVN_VERSION	"r1889 "
OMPI_GREEK_VERSION	"rc4 "

2.3.1 Stack trace information

The Debugger allows you to check the values of arguments and the values of local variables in functions by tracing the stack trace (by changing the frame). In case an argument contains an incorrect value, use the stack trace information to find which part of the stack trace path introduced the incorrect argument.

Figure 2.5 Main window (stack trace information)



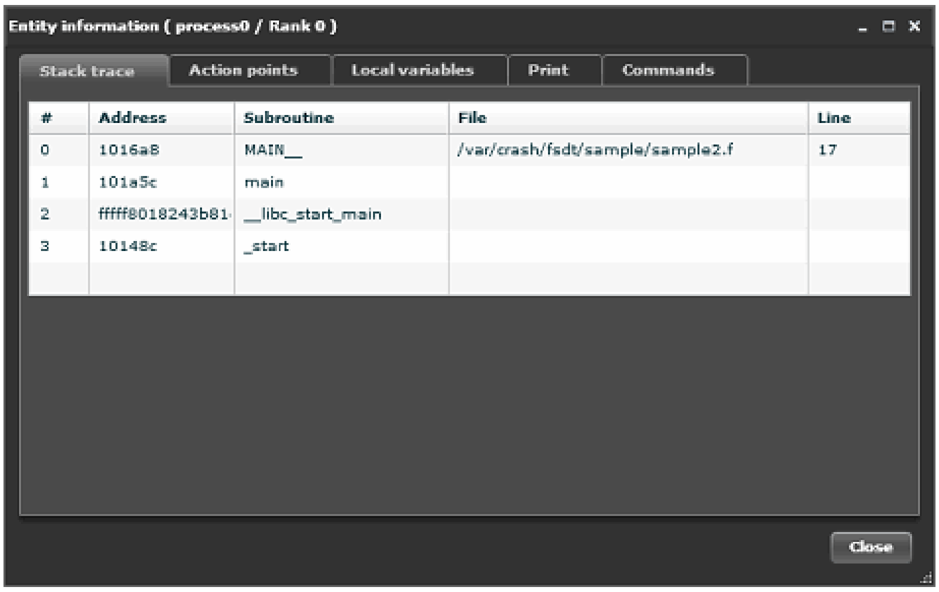
Name	Status	Location
a.out	stop	
process0	stop	sample2.f:17
process1	stop	sample2.f:17
process2	stop	sample2.f:17
process3	stop	sample2.f:17

/var/crash/fsdt/sample/sample2.f

```
      result = -100

      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
         WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
         WRITE(*,*) 'MPI communication end'
         WRITE(*,*) 'result is 0+1+...+size-1.check result(',result,')'
```

```
#0 0x1016a8 MAIN__ (/var/crash/fsdt/sample/sample2.f:17)
#1 0x101a5c main
#2 0xffff8018243b81c __libc_start_main
#3 0x10148c _start
```

Figure 2.6 Entity information window (stack trace information)



#	Address	Subroutine	File	Line
0	1016a8	MAIN__	/var/crash/fsdt/sample/sample2.f	17
1	101a5c	main		
2	ffff8018243b81	__libc_start_main		
3	10148c	_start		

Refer to the stack trace information in the main window or the **Entity information** window, and select the call position (frame) for which you want to see detailed information. The information regarding local variables for the selected call position is displayed.

If you select a call position in the stack trace in the main window while the **Entity information window** for the active entity is displayed, the position of the stack trace is automatically updated.

2.3.2 Display feature

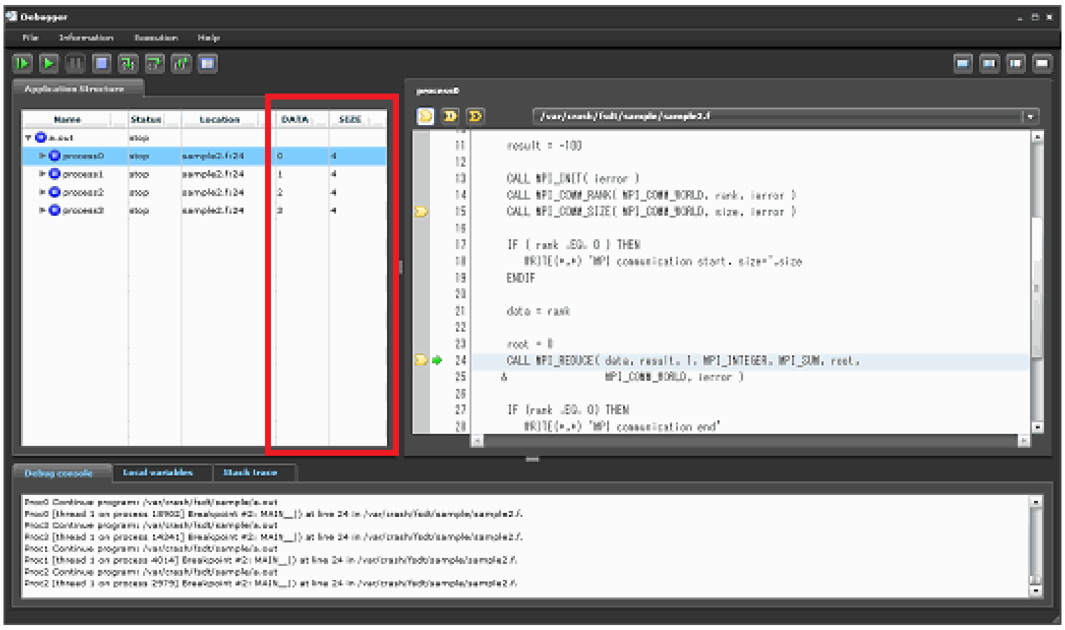
The Debugger also provides the feature to display the values of variables when a program's execution temporarily stops.

Up to 10 scalar variables can be displayed.

The variables are displayed in the **Application structure** panel.

Note that the value of a local variable, which is out of scope, is not displayed.

Figure 2.7 Main window (values of variables)



Name	Status	Location	DATA	SIZE
a.out	stop			
process0	stop	sample2.f:24	0	4
process1	stop	sample2.f:24	1	4
process2	stop	sample2.f:24	2	4
process3	stop	sample2.f:24	3	4

/var/crash/fodt/sample/sample2.f

```
      result = -100

      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
         WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
         WRITE(*,*) 'MPI communication end'
```

```
Proc0 Continue program: /var/crash/fodt/sample/a.out
Proc0 [thread 1 on process 18902] Breakpoint #2: MAIN__() at line 24 in /var/crash/fodt/sample/sample2.f.
Proc3 Continue program: /var/crash/fodt/sample/a.out
Proc3 [thread 1 on process 14041] Breakpoint #2: MAIN__() at line 24 in /var/crash/fodt/sample/sample2.f.
Proc1 Continue program: /var/crash/fodt/sample/a.out
Proc1 [thread 1 on process 40141] Breakpoint #2: MAIN__() at line 24 in /var/crash/fodt/sample/sample2.f.
Proc2 Continue program: /var/crash/fodt/sample/a.out
Proc2 [thread 1 on process 29791] Breakpoint #2: MAIN__() at line 24 in /var/crash/fodt/sample/sample2.f.
```

Chapter 3 Debugger Windows and Menus

This chapter describes the windows in the Debugger.

The Debugger comprises three windows:

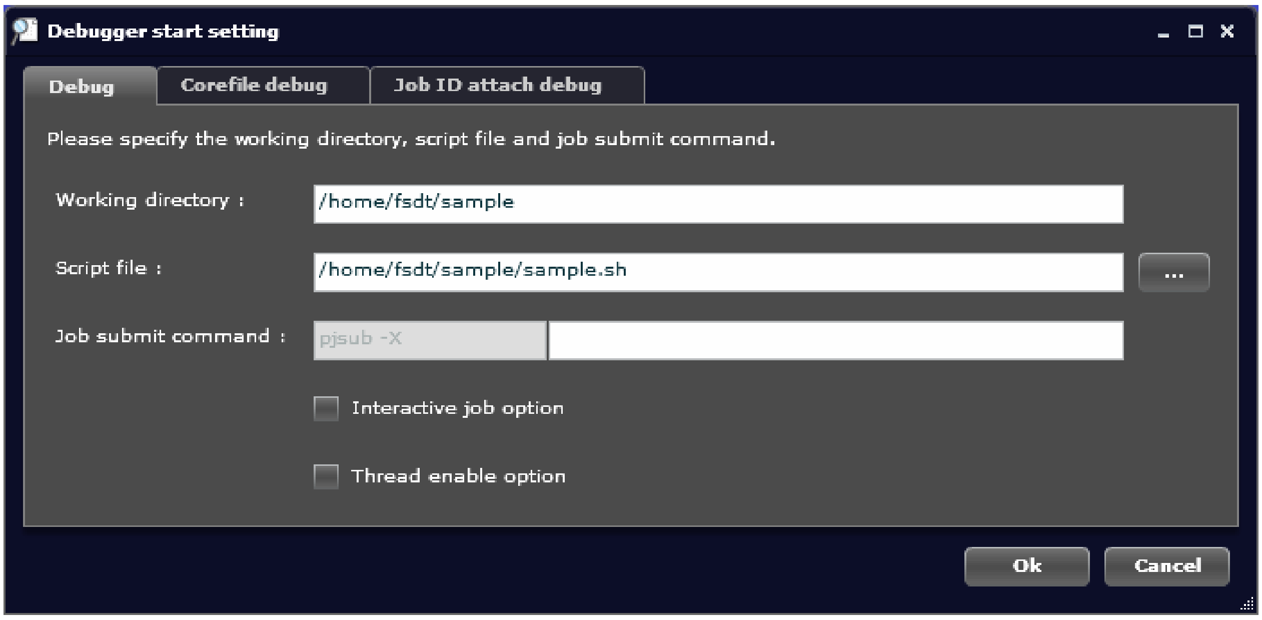
- **Debugger start setting** window: Starts the Debugger
- Main window: Operates the Debugger
- **Entity information** window: Displays the debugging information, for example, stop positions

3.1 Debugger start setting Window

This is the window that starts the debugging.

When the Debugger starts from the FUJITSU Software Development Tools, the **Debugger start setting** window is displayed.

Figure 3.1 Debugger start setting window



Please specify the working directory, script file and job submit command.

Working directory : /home/fsdt/sample

Script file : /home/fsdt/sample/sample.sh

Job submit command : pjsub -X

Interactive job option

Thread enable option

The following table describes the elements in the **Debugger start setting** window.

Table 3.1 Debugger start setting window elements

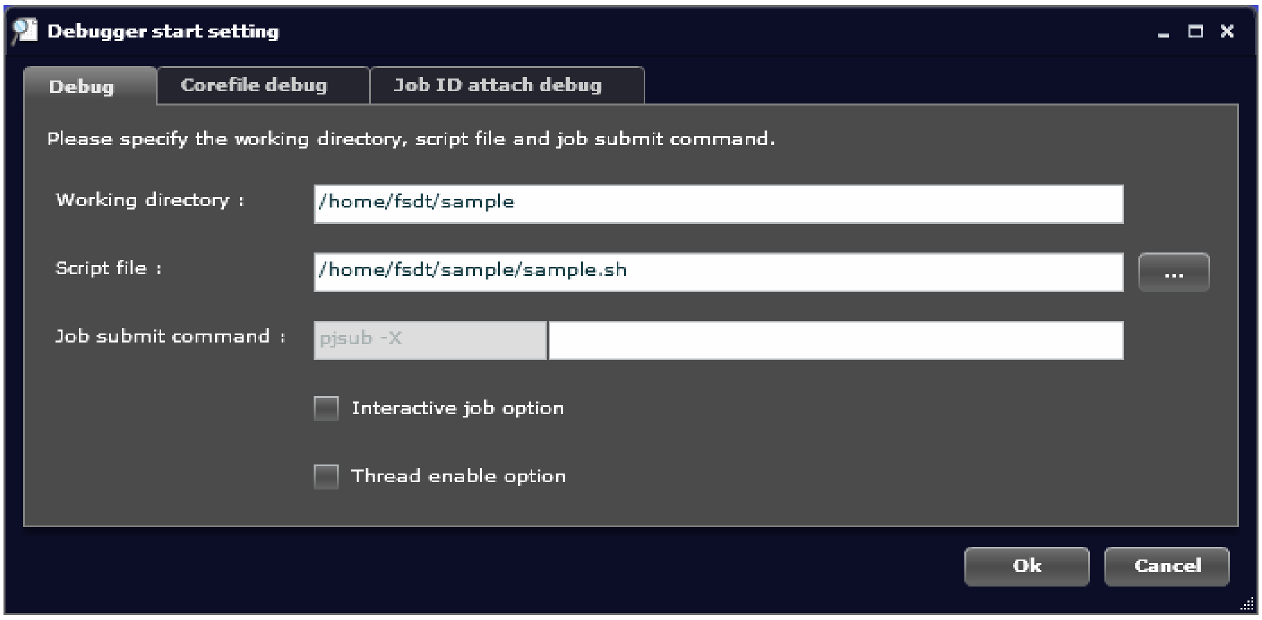
Element	Description
Debug	Allows debugging from the top of a program The working directory, the script file to be debugged, and the job submission options must be specified.
Corefile debug	Allows debugging the corefile that is output when a program abnormally terminates The working directory, the executable file, and the core image file must be specified.
Job ID attach debug	Allows debugging a program, which is already running. The job identifier must be specified to start the debugging. Use the pjstat command to confirm the job identifier of an active program.
Ok	Starts the debugging by using the information specified on the selected tab The job is submitted, and the main window is displayed.
Cancel	Closes the **Debugger start setting** window

3.1.1 Debug tab

Starts the debugging from the top of a program

The job is submitted in the specified directory.

Figure 3.2 Debugger start setting window (Debug tab)



Please specify the working directory, script file and job submit command.

Working directory : /home/fsdt/sample

Script file : /home/fsdt/sample/sample.sh

Job submit command : pjsub -X

Interactive job option

Thread enable option

The following table describes the elements on the **Debug** tab.

Table 3.2 Debug tab elements

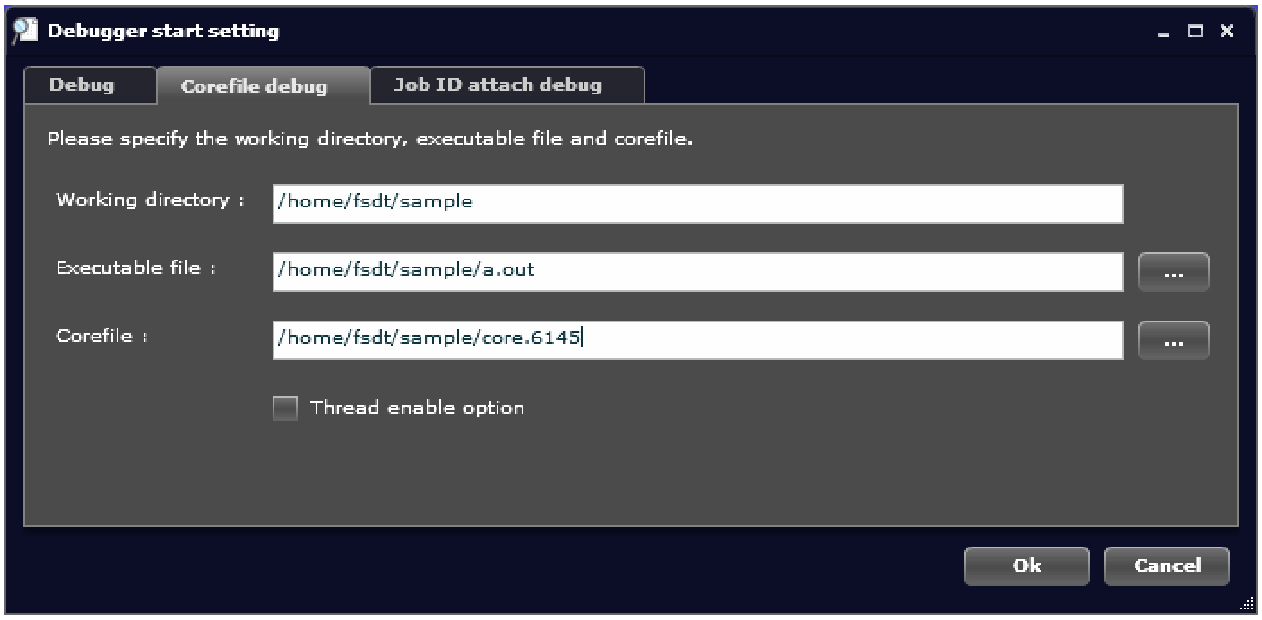
Element	Description
Working directory	The directory which the Debugger submits the job at is specified. This is not optional and must be specified.
Script file	Specify a script file that starts the program to be debugged. This is not optional and must be specified. Click the [**...**] button to select the script file from the displayed file selection dialog box.
Job submit command	Provide the command to submit the specified job. If this element appears gray, it means it is disabled and cannot be edited.
Interactive job option	Select this check box to submit an interactive job.
Thread enable option	Select this check box to display the thread information in the Debugger, in case of a multi-thread program.

3.1.2 Corefile debug tab

Allows debugging the corefile that is output when a program abnormally terminates

The following table describes the elements on the **Corefile debug** tab.

Figure 3.3 Debugger start setting window (Corefile debug tab)



Please specify the working directory, executable file and corefile.

Working directory : /home/fsdt/sample

Executable file : /home/fsdt/sample/a.out

Corefile : /home/fsdt/sample/core.6145

☐ Thread enable option

Table 3.3 Corefile debug tab elements

Element	Description
Working directory	Specify a directory that is used as the current directory while performing a corefile debug session. This is not optional and must be specified.
Executable file	Specify the executable file. This is not optional and must be specified. Click the [**...**] button to select the executable file from the displayed file selection dialog box.
Corefile	Specify the core image file to be output when the file specified in **Executable file** abnormally terminates. This is not optional and must be specified. Click the [**...**] button to select the core image file from the displayed file selection dialog box.
Thread enable option	Select this check box to display the thread information in the Debugger, in case of a multi-thread program.

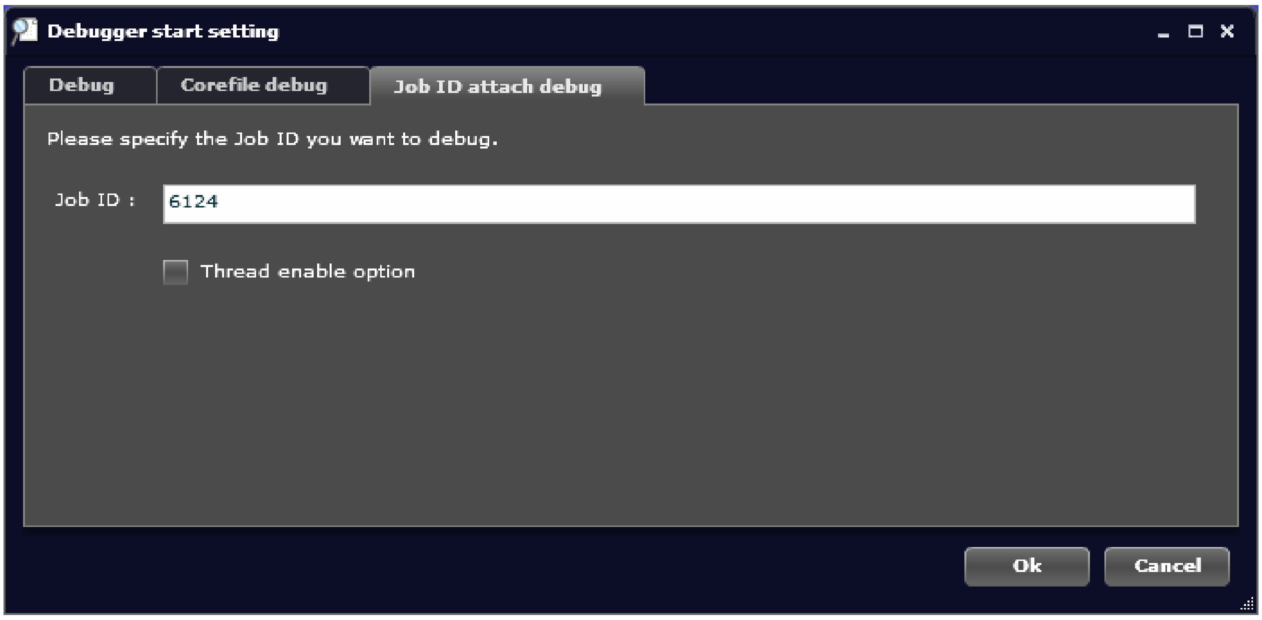
3.1.3 Job ID attach debug tab

Allows debugging a program, which is already running.

You cannot perform job ID attach debug for a program for which you do not have appropriate access rights.

The following table describes the elements on the **Job ID attach debug** tab.

Figure 3.4 Debugger start setting window (Job ID attach debug tab)



Please specify the Job ID you want to debug.

Job ID : 6124

☐ Thread enable option

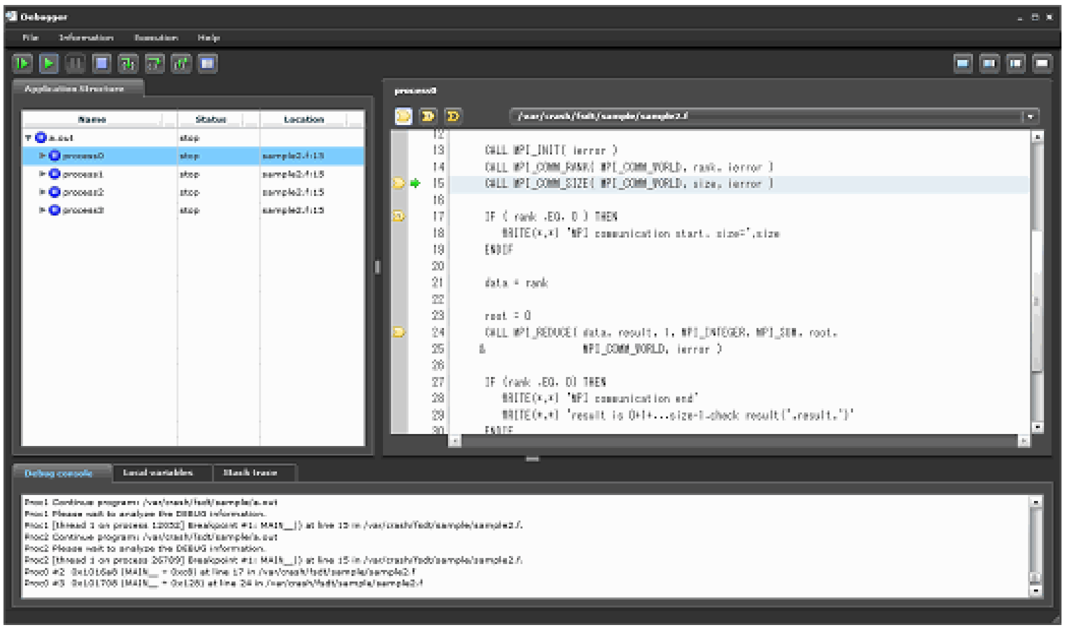
Table 3.4 Job ID attach debug tab option

Element	Description
Job ID	Specify the job identifier of the job to be debugged. This is not optional and must be specified.
Thread enable option	Select this check box to display the thread information in the Debugger, in case of a multi-thread program.

3.2 Main Window

When the debugging starts, the main window is displayed.

Figure 3.5 Main window



Name	Status	Location
a.out	stop	
process0	stop	sample2.f:15
process1	stop	sample2.f:15
process2	stop	sample2.f:15
process3	stop	sample2.f:15

/var/crash/fsdt/sample/sample2.f

```
      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
         WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
         WRITE(*,*) 'MPI communication end'
         WRITE(*,*) 'result is 0+1+...size-1.check result(',result,')'
      ENDIF
```

```
Proc1 Continue program: /var/crash/fsdt/sample/a.out
Proc1 Please wait to analyze the DEBUG information.
Proc1 [thread 1 on process 12032] Breakpoint #1: MAIN__() at line 15 in /var/crash/fsdt/sample/sample2.f.
Proc2 Continue program: /var/crash/fsdt/sample/a.out
Proc2 Please wait to analyze the DEBUG information.
Proc2 [thread 1 on process 26709] Breakpoint #1: MAIN__() at line 15 in /var/crash/fsdt/sample/sample2.f.
Proc0 #2  0x1016a8 (MAIN__ + 0xc8) at line 17 in /var/crash/fsdt/sample/sample2.f
Proc0 #3  0x101708 (MAIN__ + 0x128) at line 24 in /var/crash/fsdt/sample/sample2.f
```

The following table describes the elements in the main window.

Table 3.5 Main window elements

Element	Description
Menu bar	Provides options to enable debugging operations
Toolbar	Contains buttons for the most frequently used debugging operations, such as execution control and setting stop positions
Application structure panel	Displays the processes and the threads of the program being debugged in a tree structure; the entity that is selected becomes the active entity
Source view panel	Displays the source and stop position related information, such as the current stop line
Debug console tab	Displays messages output from the Debugger
Local variables tab	Displays the local variables and their values
Stack trace tab	Displays the stack trace information

3.2.1 Menu bar

The menu bar enables debugging operations.

Table 3.6 Debugging operations menu

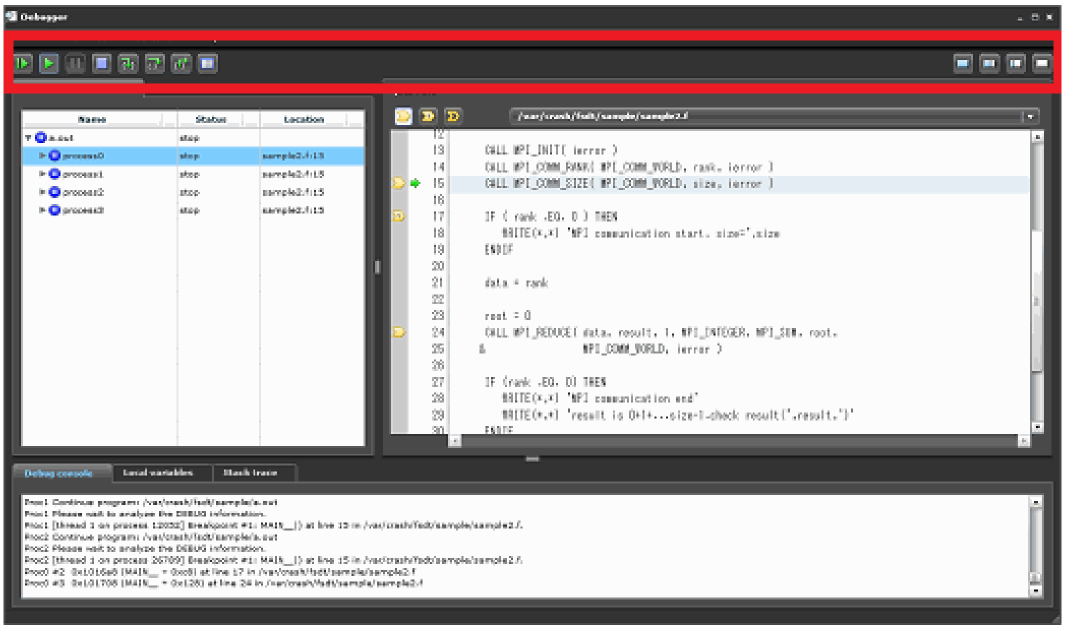
Menu	Submenu/Command		Description
File	**Exit**		Exits the Debugger
Information	**Stack trace ...**		Displays the **Entity information** window that contains information regarding the active entity Refer to "3.3 Entity Information Window" for details.
	Action points		
		Breakpoints ...	
		Watchpoints ...	
		Barrierpoints	
	Local variables ...		
	Print ...		
	Commands ...		
Execution	**Rerun**		Re-executes a program from the top The breakpoint-related information set for the group is carried over.
	Continue		Restarts the program execution The program executes till a stop position, or till it terminates.
	Stop		Temporarily interrupts the program execution However, debug operations, for example, the display of program variables and restarting program execution can be continued.
	Kill		Interrupts the program execution and terminates the program In this case, you cannot control the program execution. To resume debugging, use **Rerun**.
	Step		Executes the current line For a function call or a procedure call, it stops at the first line of the called function or procedure.
	Next		Executes the current line For a function call or a procedure call, the function or the procedure call is executed as one sentence and the execution stops at the next line. The execution does not stop at a stop position, such as a breakpoint, till it returns from the previous called function.

Menu	Submenu/Command	Description
	Finish	Executes till the return of the current function or procedure After the current function or procedure returns, it stops at the line that will be executed first. The execution does not stop at a stop position, such as a breakpoint, till it returns from the previous called function.
Help	**About Debugger**	Displays information about the Debugger

3.2.2 Toolbar

The toolbar contains icons for the most frequently used debugging operations, such as execution control.

Figure 3.6 Toolbar



Name	Status	Location
a.out	stop	
process0	stop	sample2.f:15
process1	stop	sample2.f:15
process2	stop	sample2.f:15
process3	stop	sample2.f:15

```
      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
         WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
         WRITE(*,*) 'MPI communication end'
         WRITE(*,*) 'result is 0+1+...size-1-check result(',result,')'
      ENDIF
```

```
Proc1 Continue program: /var/crash/fsdt/sample/a.out
Proc1 Please wait to analyze the DEBUG information.
Proc1 [thread 1 on process 12032] Breakpoint #1: MAIN__() at line 15 in /var/crash/fsdt/sample/sample2.f.
Proc2 Continue program: /var/crash/fsdt/sample/a.out
Proc2 Please wait to analyze the DEBUG information.
Proc2 [thread 1 on process 26709] Breakpoint #1: MAIN__() at line 15 in /var/crash/fsdt/sample/sample2.f.
Proc0 #2  0x1016a8 (MAIN__ + 0xc8) at line 17 in /var/crash/fsdt/sample/sample2.f
Proc0 #3  0x101708 (MAIN__ + 0x128) at line 24 in /var/crash/fsdt/sample/sample2.f
```

The following table describes the toolbar icons.

Table 3.7 Toolbar icons with their description

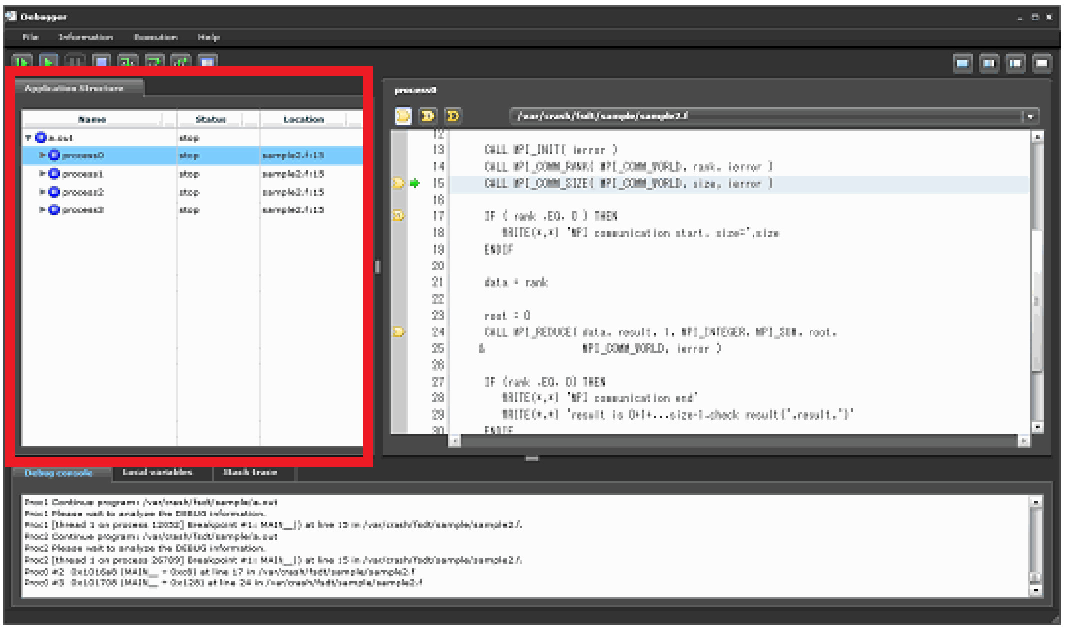
Name	Icon	Function
Return	▶	Returns from the head of the program. Information on the breakpoint set to the group is succeeded.
Continue	▶	The program execution is resumed. The program continues executing until reaching to the stop position or terminating.
Stop	⏸	The program execution is temporarily interrupted. The debug operations of the display of the variable and the restart of the program execution, etc. can be continued.
Kill	■	The program execution is interrupted, and it cancels. The program execution control cannot be done at the following. When the debug operation is continued, it is necessary to di Return.
Step	↴	The current line is executed. In the line of the function or the procedure call, it stops by the first line of the call function or procedure.
Next	↷	The current line is executed. In the line of the function or the procedure call, the function or the procedure call is executed as one sentence, and it stops by the next line. Execution does not stop at the stop position when there is a stop position like the breakpoint etc. before it returns from a call function.

Name	Icon	Function
Finish		Execute until the return of a current function or procedure. After a current function or procedure returns, it stops by the line executed first. The execution stop is not done at the stop position when there is a stop position like the breakpoint etc. before it returns from the function.
Entity information window		The entity information window is displayed.
10:0 ratio display		The ratio of the displays of the application structure panel and the source view panel is made 10:0.
7:3 ratio display		The ratio of the displays of the application structure panel and the source view panel is made 7:3.
3:7 ratio display		The ratio of the displays of the application structure panel and the source view panel is made 3:7.
0:10 ratio display		The ratio of the displays of the application structure panel and the source view panel is made 0:10.

3.2.3 Application structure panel

Displays the processes and threads of a program (It is called the application structure at the following) in a tree structure.

Figure 3.7 Application structure panel



Name	Status	Location
a.out	stop	
process0	stop	sample2.f:15
process1	stop	sample2.f:15
process2	stop	sample2.f:15
process3	stop	sample2.f:15

```
      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
         WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
         WRITE(*,*) 'MPI communication end'
         WRITE(*,*) 'result is 0+1+...size-1.check result(',result,')'
      ENDIF
```

```
Proc1 Continue program: /var/crash/fadt/sample/a.out
Proc1 Please wait to analyze the DEBUG information.
Proc1 [thread 1 on process 12032] Breakpoint #1: MAIN__() at line 15 in /var/crash/fadt/sample/sample2.f.
Proc2 Continue program: /var/crash/fadt/sample/a.out
Proc2 Please wait to analyze the DEBUG information.
Proc2 [thread 1 on process 26709] Breakpoint #1: MAIN__() at line 15 in /var/crash/fadt/sample/sample2.f.
Proc0 #2  0x1016a8 (MAIN__ + 0xc8) at line 17 in /var/crash/fadt/sample/sample2.f
Proc0 #3  0x101708 (MAIN__ + 0x128) at line 24 in /var/crash/fadt/sample/sample2.f
```

The following table describes the types of nodes (referred to as "entity") in the tree structure.

Table 3.8 Entity types

Entity name	Description
Group entity	This is the root component of the tree. The entire program that is executed is displayed. The executed program's name is displayed as an entity name.
Process entity	This is the first level of the tree. The processes in the executed program are displayed, and the number allocated in the process as entity name is displayed.
Thread entity	This is the second level of the tree. The thread in each process is displayed, and the number allocated in the thread as entity name is displayed.

Note that the icon changes depending on the run state (status) of each entity.

Table 3.9 Entity status icons

Status	Icon
running	▶
stop (suspend)	⏸
finished	⏹

In the **Application structure** panel, right-click an entity to display a right-click menu that allows you to perform various debugging operations.

Table 3.10 Right-click menu in the Application structure panel

Command	Description
Display ...	Adds a variable to the display feature Use this option to display a dialog box where you can specify the variable to be added to the display feature.
Delete display	Deletes the variable set for the display feature Use this option to display a list of variables set for the display feature, and then delete the variables individually.
Change current	Sets the entity that is currently selected as the current entity of the parent entity For example, if a process entity is selected, it is set as the current entity of the group entity, which is the parent entity. Similarly, if a thread entity is selected, it is set as the current entity of the process entity, which is the parent entity.
Show rank number	Changes the display of the **Name** column in the **Application structure** panel to the rank number
Show process number	Changes the display of the **Name** column in the **Application structure** panel to the process number
Entity information ...	Displays the **Entity information** window

The entity that is selected in the **Application structure** panel for debugging is called the active entity.

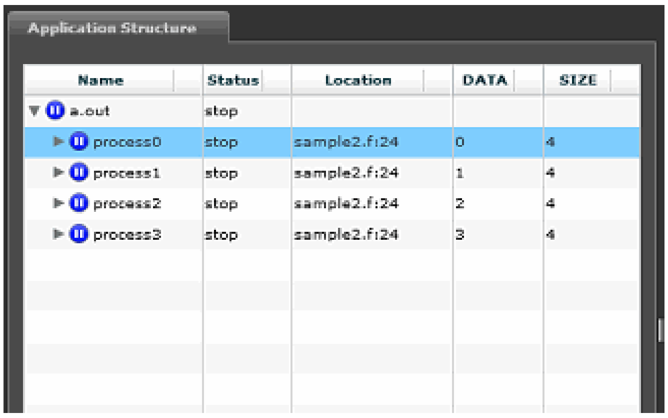
The contents of the source view panel, the **Local variables** tab, and the **Stack trace** tab are updated according to the active entity that is selected.

Display feature

The display feature updates the values of displayed variables whenever the situation changes, for example, when the execution stops at a function. As a result, it enables monitoring of variables values.

The variables for each process entity are displayed to the right of the table in the **Application structure** panel.

Figure 3.8 Application structure panel (display feature)



Application Structure

Name	Status	Location	DATA	SIZE
a.out	stop			
process0	stop	sample2.f:24	0	4
process1	stop	sample2.f:24	1	4
process2	stop	sample2.f:24	2	4
process3	stop	sample2.f:24	3	4

Up to 10 variables can be displayed using the display feature.

Use any of the following methods to add a variable to monitor it.

- Use the right-click menu in the **Application structure** panel.
- Use the right-click menu on the **Local variables** tab.
- Use the right-click menu in the source view panel.
- Use the right-click menu on the **Local variables** tab in the **Entity information** window.

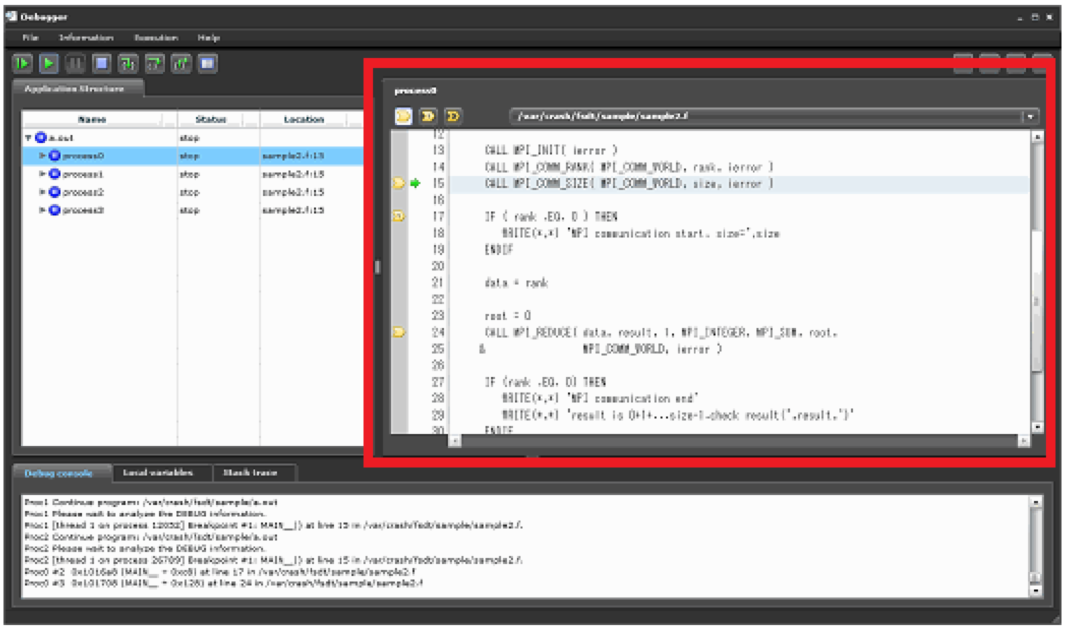
You can delete a displayed variable by using the right-click menu in the **Application structure** panel.

3.2.4 Source view panel

Displays the source and the execution stop positions, such as breakpoints

When the Debugger starts, the source file with the stop positions included is displayed in the source view panel.

Figure 3.9 Source view panel



Name	Status	Location
a.out	stop	
process0	stop	sample2.f:15
process1	stop	sample2.f:15
process2	stop	sample2.f:15
process3	stop	sample2.f:15

```
      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
        WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
        WRITE(*,*) 'MPI communication end'
        WRITE(*,*) 'result is 0+1+...size-1-check result(',result,')'
      ENDIF
```

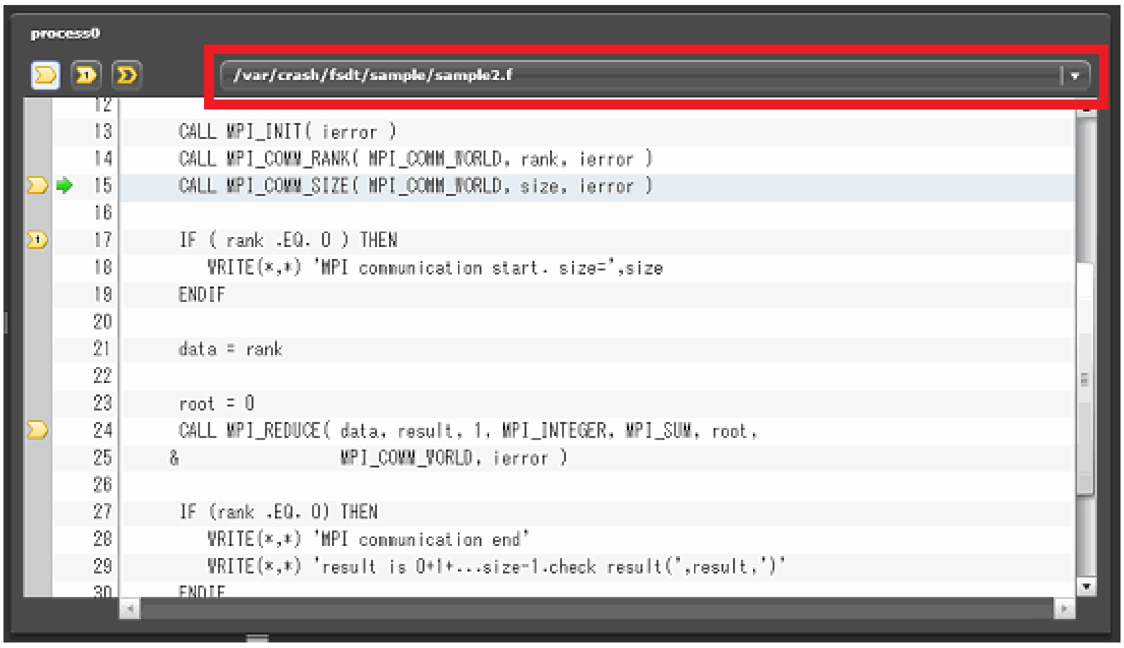
```
Proc1 Continue program: /var/crash/fsdt/sample/a.out
Proc1 Please wait to analyze the DEBUG information.
Proc1 [thread 1 on process 12032] Breakpoint #1: MAIN__() at line 15 in /var/crash/fsdt/sample/sample2.f.
Proc2 Continue program: /var/crash/fsdt/sample/a.out
Proc2 Please wait to analyze the DEBUG information.
Proc2 [thread 1 on process 26709] Breakpoint #1: MAIN__() at line 15 in /var/crash/fsdt/sample/sample2.f.
Proc0 #2  0x1016a8 (MAIN__ + 0xc8) at line 17 in /var/crash/fsdt/sample/sample2.f
Proc0 #3  0x101708 (MAIN__ + 0x128) at line 24 in /var/crash/fsdt/sample/sample2.f
```

Source file selection box

Use the source file selection box to select the source file to be displayed.

This allows you to open the other program source file, so that you can browse the source code and set a new breakpoint.

Figure 3.10 Source file selection box



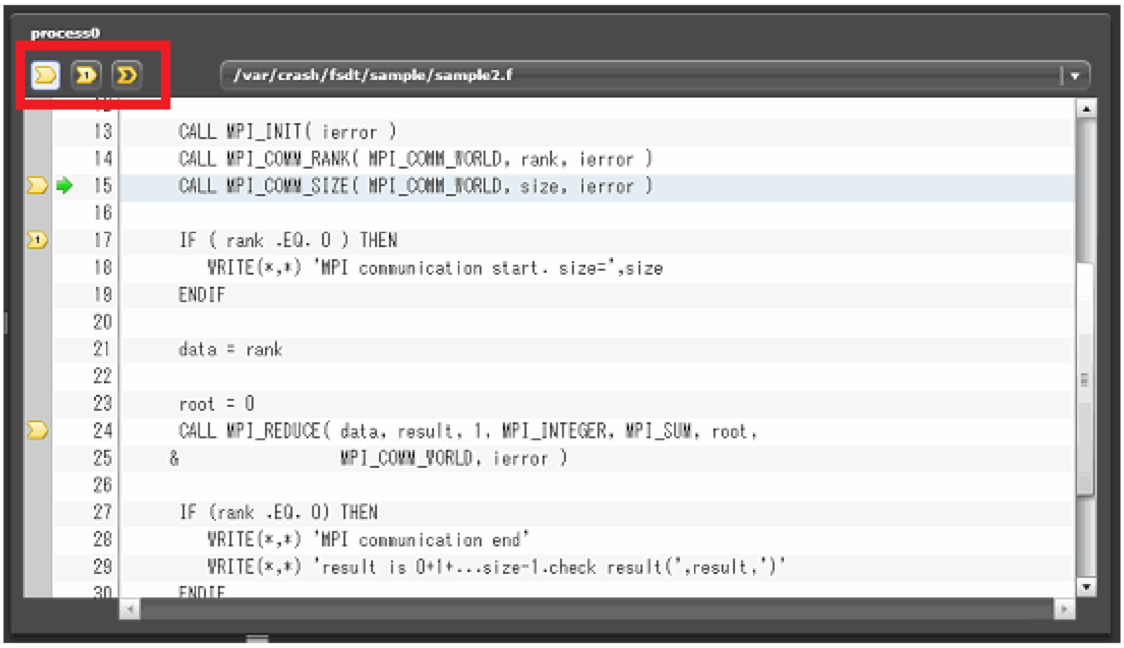
```
      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
         WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
         WRITE(*,*) 'MPI communication end'
         WRITE(*,*) 'result is 0+1+...size-1.check result(',result,')'
      ENDIF
```

Action point selection icons

In the Debugger, breakpoints, temporary breakpoints, barrierpoints, and watchpoints are referred to as action points.

Click the margin area in the source view panel (as described below) to set an action point (breakpoint, temporary breakpoint, or barrierpoint) for a source line.

Figure 3.11 Action point selection icons



```
      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
         WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
         WRITE(*,*) 'MPI communication end'
         WRITE(*,*) 'result is 0+1+...size-1.check result(',result,')'
      ENDIF
```

Table 3.11 Action point selection icons

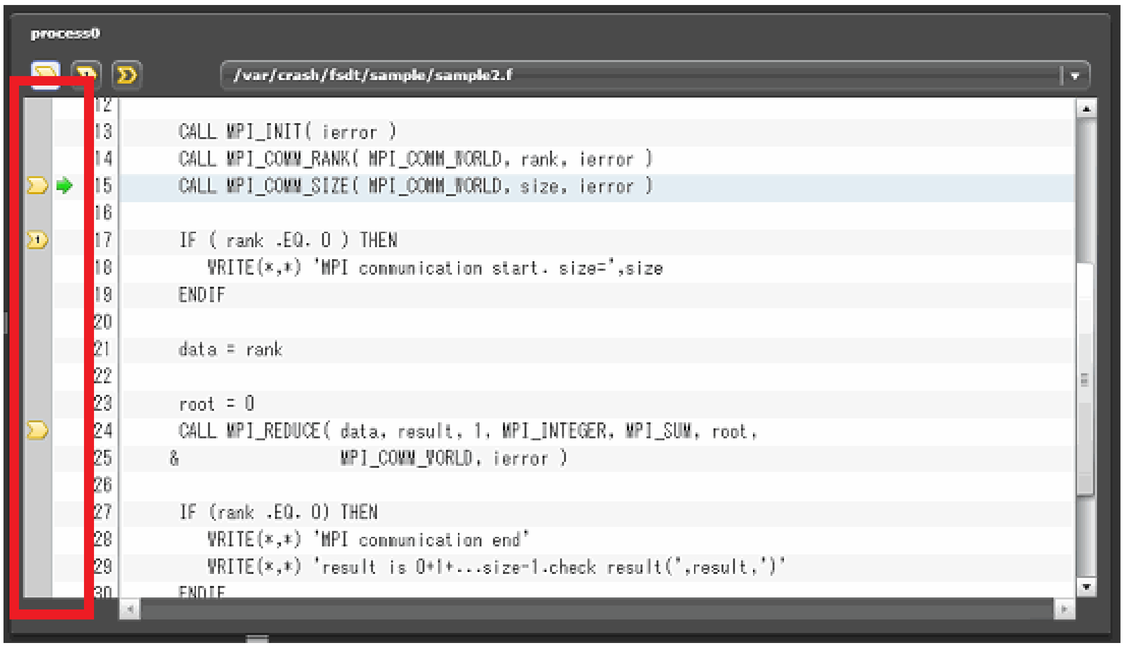
Name	Icon	Function
Breakpoint	▷	The breakpoint can be set by clicking the margin area.
Temporary breakpoint	▷1	The temporary breakpoint (cleared automatically when the program stops) can be set by clicking the margin area.

Name	Icon	Function
		When an active entity is a group and a thread, it becomes deactivated and cannot to use.
Barrier point	⪢	The barrier point can be set by clicking the margin area. When an active entity is a group and a thread, it becomes deactivated and cannot to use.

Margin area

The area (left of the source row number) where the stop position is displayed is called the margin area. When the margin area is clicked, the position where the cursor stops is set as the action point. The set action point is identified by the action point icon (described in the table below). The action point can be released by clicking the action point icon.

Figure 3.12 Margin area



```
      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
         WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
         WRITE(*,*) 'MPI communication end'
         WRITE(*,*) 'result is 0+1+...size-1.check result(',result,')'
      ENDIF
```

The following table shows the icons displayed in the margin area.

Table 3.12 Margin area icons

Display item	Status	Icon
Application stop position	Always	➡
Breakpoint	Enabled	▷
	Disabled	▷
Temporary breakpoint	Enabled	▷
	Disabled	▷
Barrier point	Enabled	⪢
	Disabled	⪢

When you right-click a breakpoint, a temporary breakpoint, or a barrierpoint displayed in the margin area, a right-click menu is displayed.

Table 3.13 Right-click menu in the margin area

Command	Description
Delete	The breakpoint or the barrierpoint is deleted.
Disable	The breakpoint or the barrierpoint is disabled.
Enable	The breakpoint or the barrierpoint is enabled.

Debug operations in the source view panel

When you right-click a source, a right-click menu is displayed. The action corresponding to the selected menu option will be performed for the string selected in the source view window.

Table 3.14 Right-click menu in the source view panel

Command	Description
Print	Displays the values of program variables; the result is displayed in the **Entity information** window. When the **entity information** window of an active entity is not displayed, it is newly displayed.
Set ...	Sets the value for a variable; a dialog box is displayed where you can set the value for the variable
Display	Adds a variable to the display feature Refer to "3.2.3 Application structure panel" for details.
Watch	Sets a watchpoint (for updating) The program execution stops at the point where the specified variable was changed.
Refer	Sets a watchpoint (for referencing) The program execution stops at the point where the specified variable was referred.
Referwatch	Sets a watchpoint (for updating or referencing) The program execution stops at the point where the specified variable was changed or was referred.

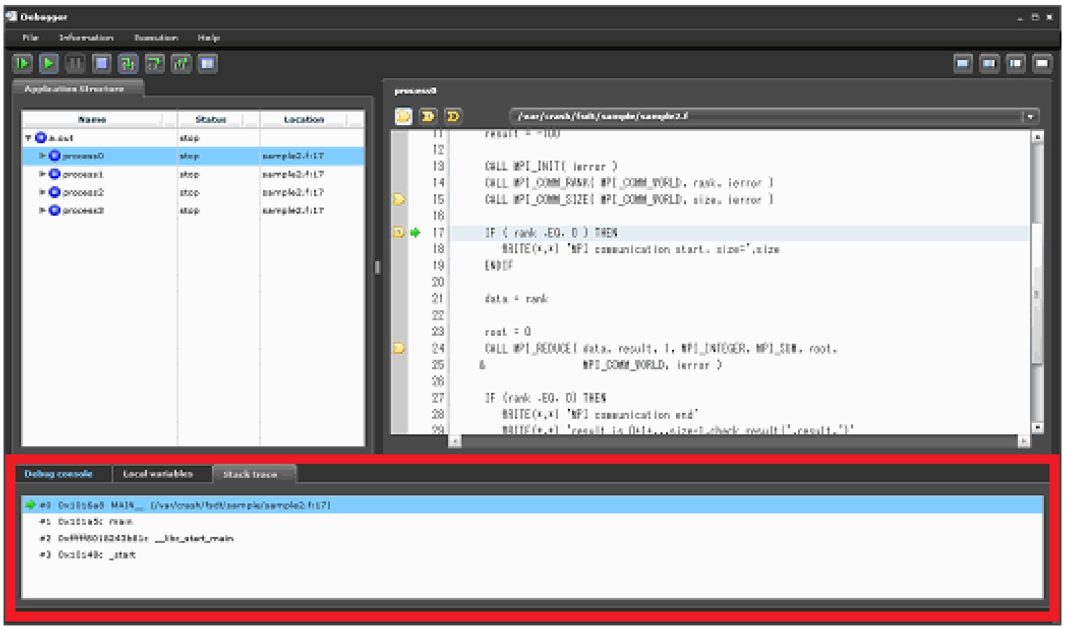
3.2.5 Stack trace tab

Displays the list of stack frames for a program

The frame number, the address, the function identifier, the source line, and the source file name are displayed in each line.

Clicking a frame in the Stack trace tab will change the current frame. Note that changing the frame is only possible if the active entity is a process entity.

Figure 3.13 Stack trace tab



```
      result = -100

      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
        WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
        WRITE(*,*) 'MPI communication end'
```

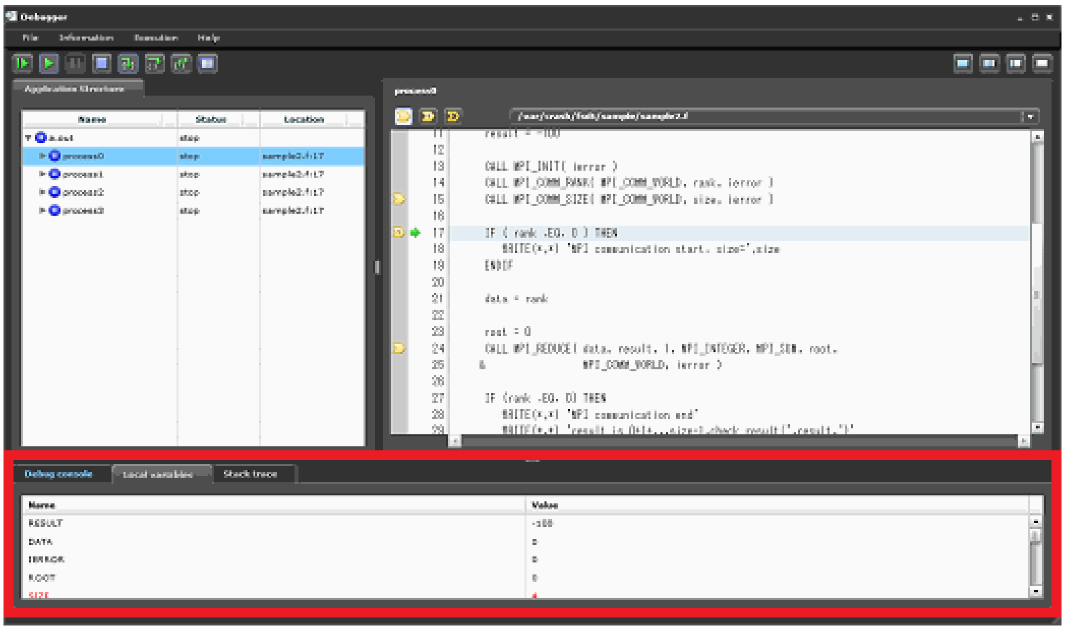
Name	Status	Location
a.out	stop	
process0	stop	sample2.f:17
process1	stop	sample2.f:17
process2	stop	sample2.f:17
process3	stop	sample2.f:17

```
#0 0x4016a8 MAIN__ [/var/crash/fsdt/sample/sample2.f:17]
#1 0x401a3c main
#2 0xffff80018243b81c __libc_start_main
#3 0x40148c _start
```

3.2.6 Local variables tab

Displays the values of local variables of the current frame.

Figure 3.14 Local variables tab



Name	Value
RESULT	-100
DATA	0
IERROR	0
ROOT	0
SIZE	4

When you right-click a local variable, a right-click menu is displayed.

Table 3.15 Right-click menu on the Local variables tab

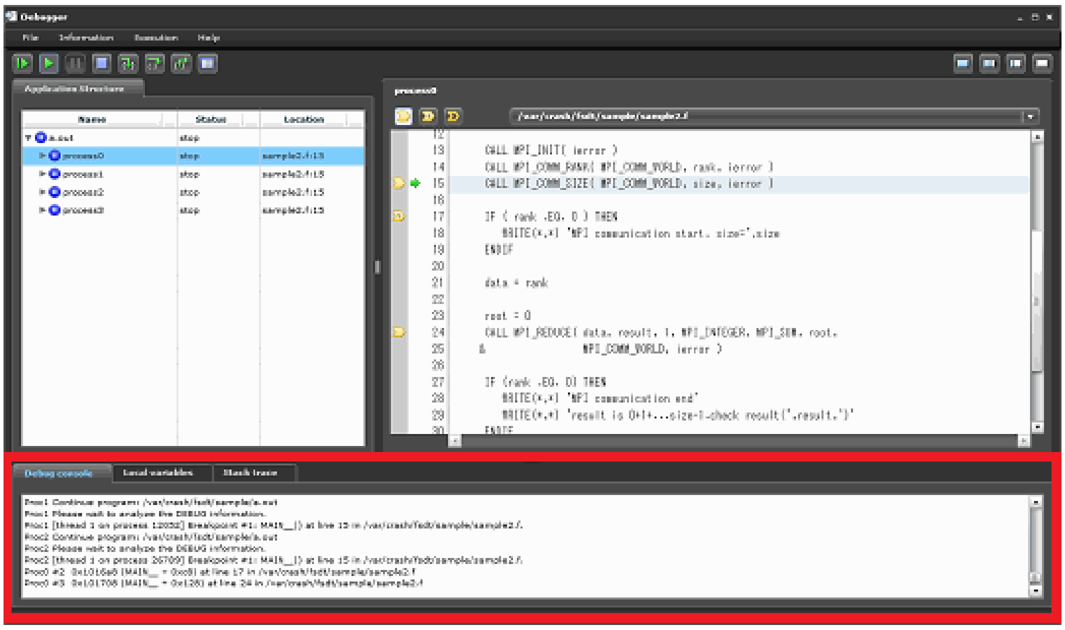
Command	Description
Print	Displays the values of program variables; the result is displayed in the **Entity information** window. Refer to the Print tab in the Entity information window for the display.
Print(Pointer)	If the program variable is a pointer, the destination of the pointer is displayed. The result is displayed on the **Print** tab in the **Entity Information** window. Refer to the Print tab in the Entity information window for the display.

Command	Description
Set ...	Sets the value for a variable; a dialog box is displayed where you can set the value for the variable
Display	Adds a variable to the display feature Refer to "3.2.3 Application structure panel".
Watch	Sets a watchpoint (for updating) The program execution stops at the point where the specified variable was changed.
Refer	Sets a watchpoint(for referencing) The program execution stops at the point where the specified variable was referred.
Referwatch	Sets a watchpoint(for updating or referencing) The program execution stops at the point where the specified variable was changed or was referred.

3.2.7 Debugging console tab

Displays the execution log of the Debugger.

Figure 3.15 Debugging log display area



Name	Status	Location
a.out	stop	
process0	stop	sample2.f:15
process1	stop	sample2.f:15
process2	stop	sample2.f:15
process3	stop	sample2.f:15

/var/crash/fsdt/sample/sample2.f

```
      CALL MPI_INIT( ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierror )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierror )

      IF ( rank .EQ. 0 ) THEN
         WRITE(*,*) 'MPI communication start. size=',size
      ENDIF

      data = rank

      root = 0
      CALL MPI_REDUCE( data, result, 1, MPI_INTEGER, MPI_SUM, root,
     &                 MPI_COMM_WORLD, ierror )

      IF (rank .EQ. 0) THEN
         WRITE(*,*) 'MPI communication end'
         WRITE(*,*) 'result is 0+1+...size-1.check result(',result,')'
      ENDIF
```

```
Proc1 Continue program: /var/crash/fsdt/sample/a.out
Proc1 Please wait to analyze the DEBUG information.
Proc1 [thread 1 on process 12032] Breakpoint #1: MAIN__() at line 15 in /var/crash/fsdt/sample/sample2.f.
Proc2 Continue program: /var/crash/fsdt/sample/a.out
Proc2 Please wait to analyze the DEBUG information.
Proc2 [thread 1 on process 26709] Breakpoint #1: MAIN__() at line 15 in /var/crash/fsdt/sample/sample2.f.
Proc0 #2  0x1016a0 (MAIN__ + 0xe0) at line 17 in /var/crash/fsdt/sample/sample2.f
Proc0 #3  0x101708 (MAIN__ + 0x128) at line 24 in /var/crash/fsdt/sample/sample2.f
```

3.3 Entity Information Window

For each entity, you can configure the following display/settings using the **Entity information** window.

When the situation of a program changes by the debugging operations in the main window, the contents of the displayed **Entity information** window are automatically updated to the latest information.

Table 3.16 Entity information window display/settings

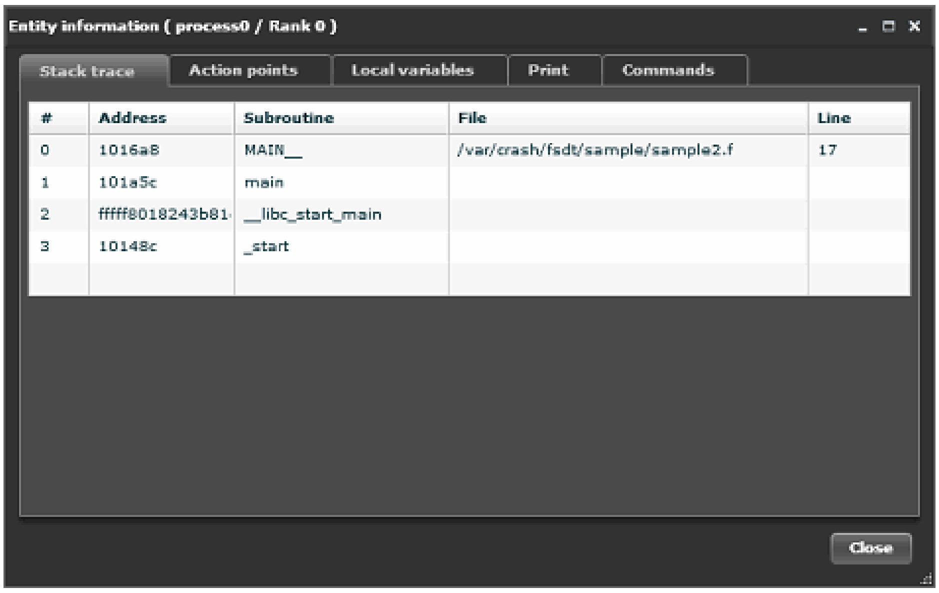
Tab name	Description
Stack trace	Displays the stack trace information and allows change of frame
Action points	Displays the list of breakpoints, watchpoints, and barrierpoints and allows related operations
Local variables	Displays the local variables with their respective values
Print	Displays program variables
Commands	Issues the fdb command

Select an entity in the **Application structure** panel, and then click the **Properties** button on the toolbar to display the **Entity information** window.

3.3.1 Stack trace tab

Traces of the called function and the call position are displayed. The display is the same as the **Stack trace** tab in the main window.

Figure 3.16 Entity information window (Stack trace tab)



#	Address	Subroutine	File	Line
0	1016a8	MAIN__	/var/crash/fsdt/sample/sample2.f	17
1	101a5c	main		
2	fffff8018243b81	__libc_start_main		
3	10148c	_start		

Change of frame

Right-click the call position you want to change, and then click **Change frame**.

However, this can only be done for process entities.

3.3.2 Action points tab

All action points are collectively displayed on the **Action points** tab.

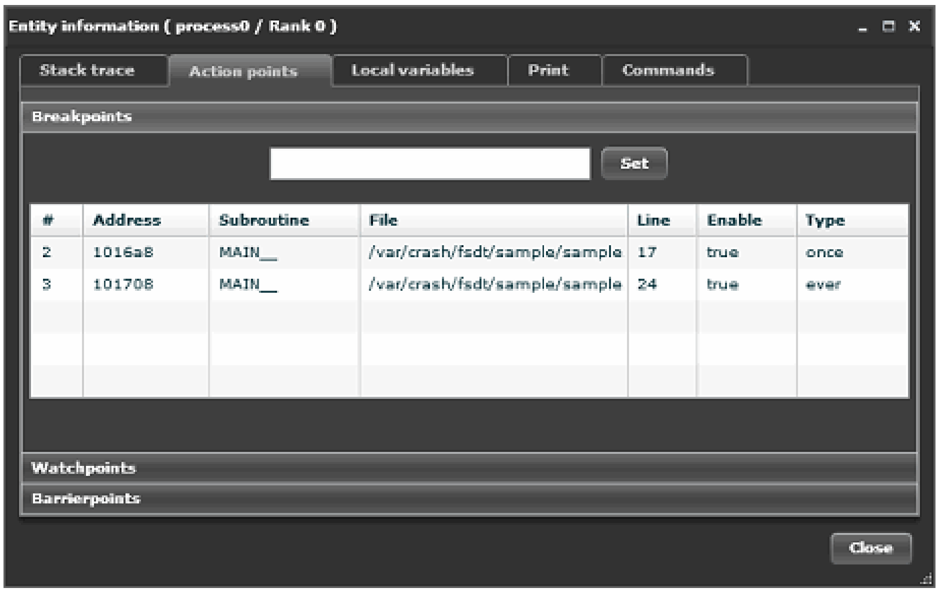
Click the **Breakpoints** bar, the **Watchpoints** bar, or the **Barrierpoints** bar to display the list of breakpoints, watchpoints, and barrierpoints, respectively.

3.3.2.1 Breakpoints bar

Displays breakpoint and temporary breakpoints, and allows you to perform various operations, such as set/delete breakpoints

The temporary breakpoint is displayed in the **Type** column as "Once".

Figure 3.17 Entity information window (Action points tab, Breakpoints bar)



#	Address	Subroutine	File	Line	Enable	Type
2	1016a8	MAIN__	/var/crash/fsdt/sample/sample	17	true	once
3	101708	MAIN__	/var/crash/fsdt/sample/sample	24	true	ever

Set

To set a breakpoint, specify the function identifier and the row number in the input box, and then click **Set**.

In the input box, specify arguments of the break command of fdb.

When debugging C++ programs, if a breakpoint is set by specifying the function identifier in the input box, an overload (two or more corresponding functions exist) may occur. In this case, a window with multiple candidates that can be set as breakpoints is displayed. Select the items you want to set as breakpoints.

Delete

To delete a breakpoint or a temporary breakpoint, right-click the breakpoint, and then click **Delete breakpoint**.

Enable/Disable

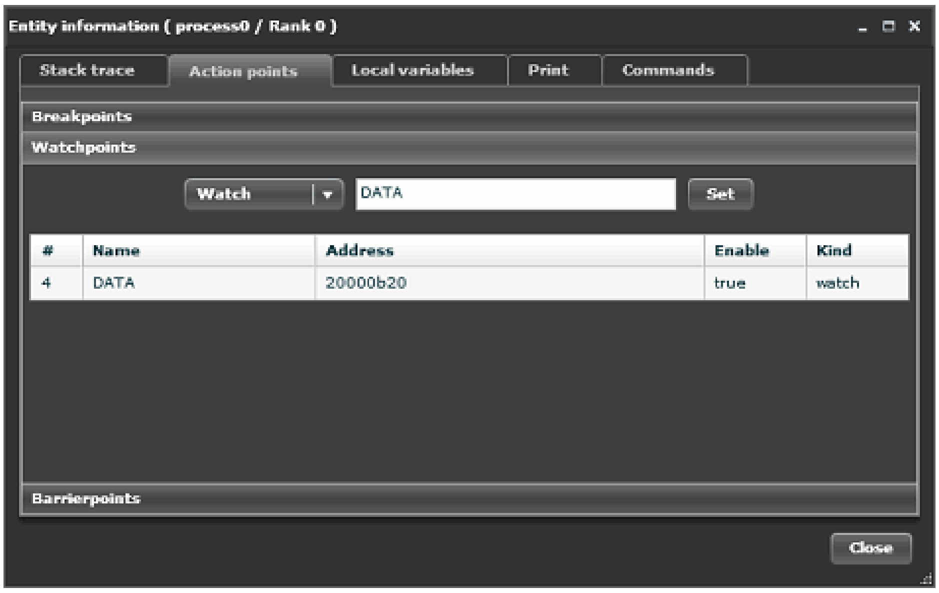
It is possible to temporarily disable a breakpoint or temporary breakpoint without deleting it.

For this, right-click a breakpoint, and then click **Enable breakpoint** or **Disable breakpoint** to switch to the enabled or disabled state, respectively.

3.3.2.2 Watchpoints bar

Displays a list of watchpoints and allows you to set/delete watchpoints.

Figure 3.18 Entity information window (Action points tab, Watchpoints bar)


Entity information (process0 / Rank 0)

DATA

#	Name	Address	Enable	Kind
4	DATA	20000b20	true	watch

The **Kind** column displays the following attributes:

- **Watch**

  Stops the program execution when the value is changed; can be set by selecting **Watch** for the watchpoint

- **Refer**

  Stops the program execution when the value is referred; can be set by selecting **Refer** for the watchpoint

- **Referwatch**

  Stops the program execution when the value is changed or is referred; can be set by selecting **Referwatch** for the watchpoint

Set

To set a watchpoint, specify the variable identifier in the input box, select an attribute (**Watch**, **Refer**, or **Referwatch**) from the list, and click **Set**.

In the input box, specify arguments of the watch, refer, or referwatch commands of fdb.

Delete

To delete a watchpoint, right-click the watchpoint, and then click **Delete watchpoint**.

Enable/Disable

It is possible to temporarily disable a watchpoint without deleting it.

For this, right-click the watchpoint, and then click **Enable watchpoint** or **Disable watchpoint** to switch to the enabled/disabled state, respectively.

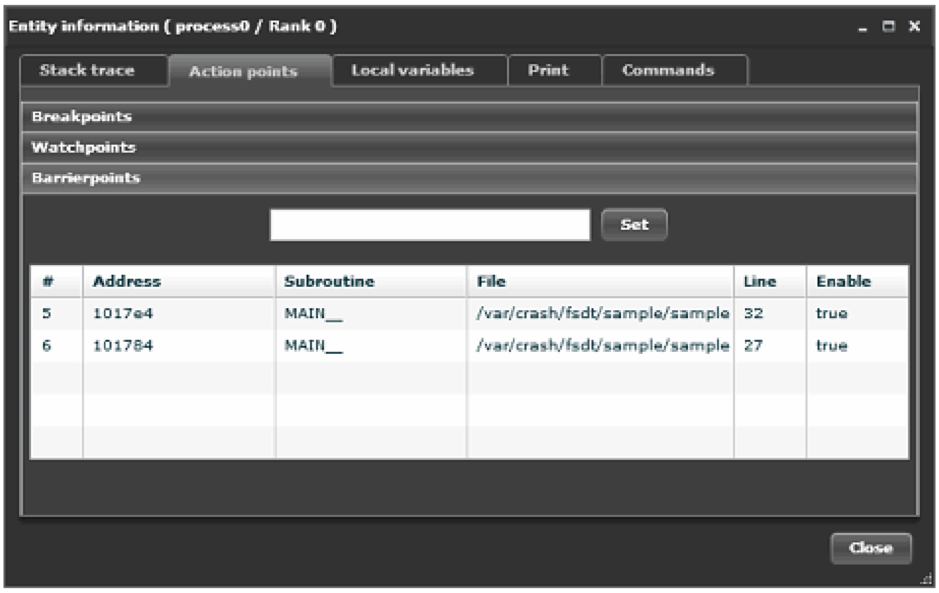
3.3.2.3 Barrierpoints bar

Displays a list of barrierpoints and allows you to set/delete barrierpoints

Barrierpoints can only be set for a process entity of the entity being displayed. The display area of the window is disabled if the current entity is a group entity or a thread entity.

A barrierpoint continues executing till all threads in the process reach the barrierpoint.

Figure 3.19 Entity information window (Action points tab, Barrierpoints bar)



#	Address	Subroutine	File	Line	Enable
5	1017e4	MAIN__	/var/crash/fsdt/sample/sample	32	true
6	101784	MAIN__	/var/crash/fsdt/sample/sample	27	true

Set

To set a barrierpoint, specify the function identifier and the row number in the input box, and then click **Set**.

In the input box, specify arguments of the barrier command of fdb.

Delete

To delete a barrierpoint, right-click the barrierpoint, and then click **Delete barrierpoint**.

Enable/Disable

It is possible to temporarily disable a barrierpoint without deleting it.

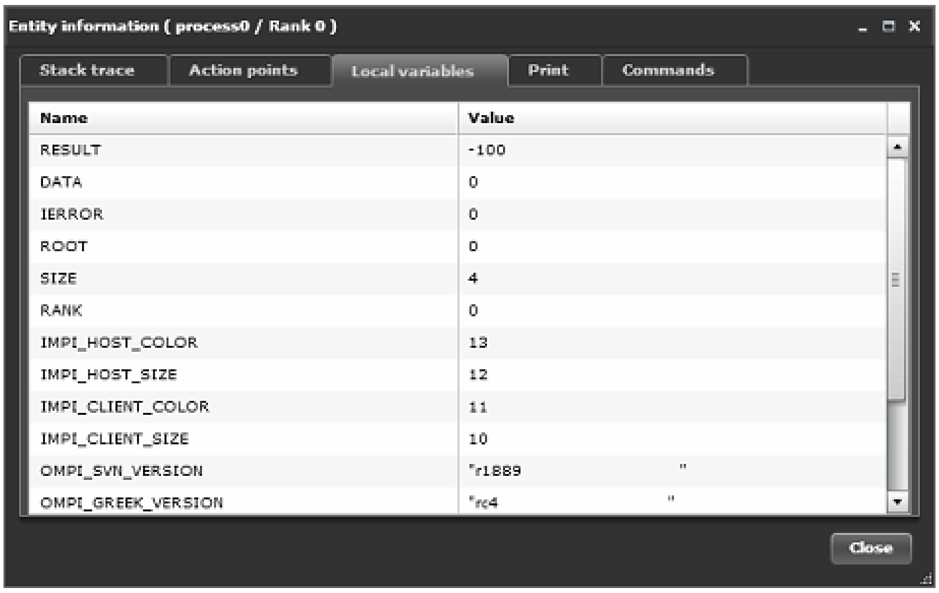
For this, right-click a barrierpoint, and then click **Enable barrierpoint** or **Disable barrierpoint** to switch to the enabled/disabled state, respectively.

3.3.3 Local variables tab

Displays a list of local variables with their respective values for the entity that is currently selected.

The display of this tab is the same as the **Local variables** tab in the main window.

Figure 3.20 Entity information window (Local variables tab)



Name	Value
RESULT	-100
DATA	0
IERROR	0
ROOT	0
SIZE	4
RANK	0
IMPI_HOST_COLOR	13
IMPI_HOST_SIZE	12
IMPI_CLIENT_COLOR	11
IMPI_CLIENT_SIZE	10
OMPI_SVN_VERSION	"r1889 "
OMPI_GREEK_VERSION	"rc4 "

When you right-click a local variable, a right-click menu with the following options is displayed.

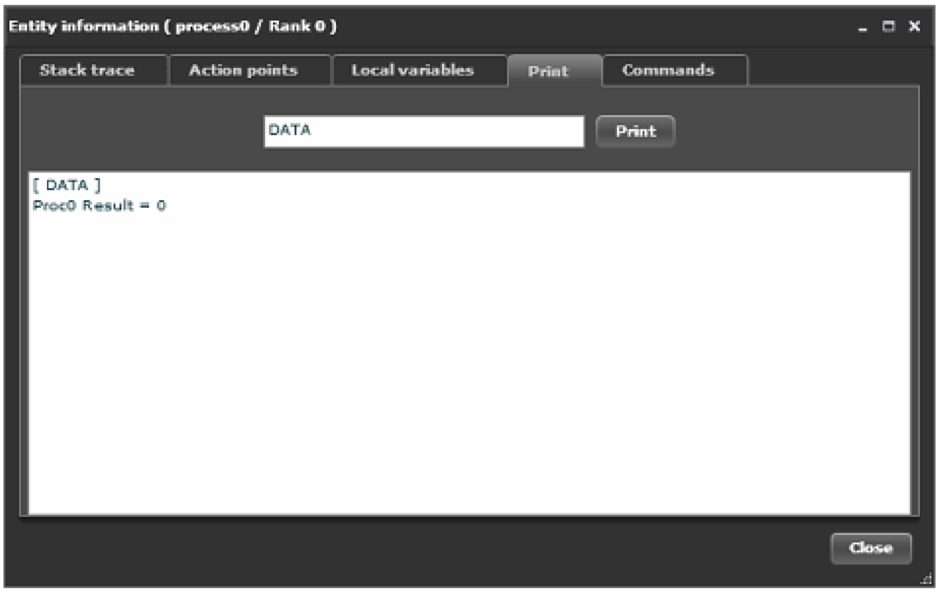
Table 3.17 Right-click menu on the Local variables tab

Command	Description
Print	Displays the values of program variables; the result is displayed on the **Print** tab
Print(Pointer)	If the program variable is a pointer, the destination of the pointer is displayed. The result is displayed on the **Print** tab.
Set	Sets the value for a variable; a dialog box is displayed where you can set the value for the variable
Display	Adds a variable to the display feature Refer to "3.2.3 Application structure panel".
Watch	Sets a watchpoint (for updating) The program execution stops at the point where the specified variable was changed.
Refer	Sets a watchpoint (for referencing) The program execution stops at the point where the specified variable was referred.
Referwatch	Sets a watchpoint (for updating or referencing) The program execution stops at the point where the specified variable was changed or was referred.

3.3.4 Print tab

Displays the values of program variables.

Figure 3.21 Entity information window (Print tab)



```
[ DATA ]
Proc0 Result = 0
```

Display

To display a value, specify the name of the program variable in the input box, and then click **Print**.

When the variable is an array and the same value appears consecutively, the value may be displayed only once (merged as a single item).

In the input box, specify arguments of the print command of fdb.

Initialization

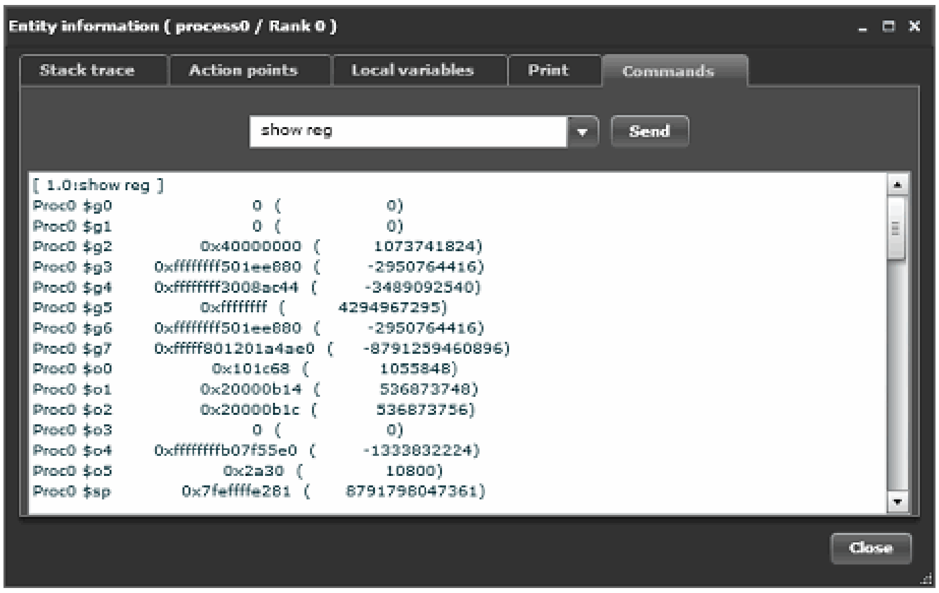
Right-click in the **Print** tab, and then click **Clear** to delete the displayed content.

3.3.5 Commands tab

Issues fdb commands for the entity that is currently selected.

Only a few of the fdb commands can be issued from the Debugger. Either select or type the command to be issued in the box, and then click **Send**. The command is sent to the entity of the **Entity information** window, and the result is displayed on the **Commands** tab.

Figure 3.22 Entity information window (Commands tab)



```
[ 1.0:show reg ]
Proc0 $g0               0  (           0)
Proc0 $g1               0  (           0)
Proc0 $g2      0x40000000  (  1073741824)
Proc0 $g3  0xffffffff501ee880  (  -2950764416)
Proc0 $g4  0xffffffff3008ac44  (  -3489092540)
Proc0 $g5      0xffffffff  (  4294967295)
Proc0 $g6  0xffffffff501ee880  (  -2950764416)
Proc0 $g7  0xfffff801201a4ae0  (  -8791259460896)
Proc0 $o0        0x101c68  (     1055848)
Proc0 $o1      0x20000b14  (   536873748)
Proc0 $o2      0x20000b1c  (   536873756)
Proc0 $o3               0  (           0)
Proc0 $o4  0xffffffffb07f55e0  (  -1333832224)
Proc0 $o5          0x2a30  (       10800)
Proc0 $sp    0x7fefffe281  (  8791798047361)
```

The fdb commands that can be selected from the box are listed below. Refer to the fdb manual, which can be displayed by executing the man command, for information on fdb commands.

```
show args, show barrier, show break, show break-t, show ffile, show fopt, show frame, show freg,
show function, show leak, show locals, show map, show namespace, show reg, show regs, show sfreg,
show signal, show source, show sources, show thread, show variable, show watch, show watch-t,
show xfreg, param execpath, param leak, param srcpath, condition, disas, dump, traceback,
traceback-t
```

Initialization

Right-click on the **Commands** tab, and then click **Clear** to delete the displayed content.

3.4 Exiting Debugger

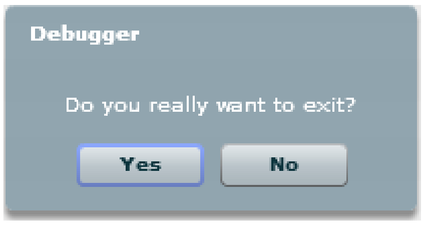
On the **File** menu, click **Exit** to exit the Debugger.

If you try to exit the Debugger during program execution on normal debug/corefile debug, the Debugger exit confirmation dialog box (for normal debug/corefile debug) that confirms if you want to exit the Debugger is displayed.

Click **Yes** button to exit the Debugger. The submitted job is also terminated.

Click **No** button to continue using the Debugger.

Figure 3.23 Debugger exit confirmation dialog box (normal debug/corefile debug)



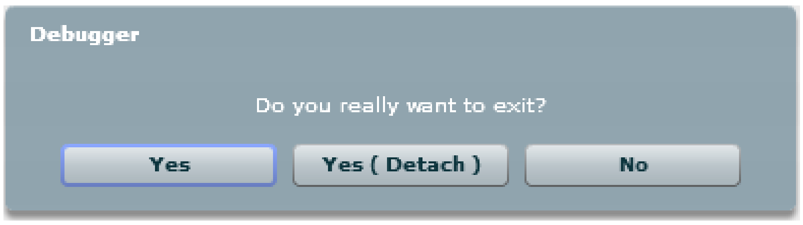
If you try to exit the Debugger during the job ID attach debug, the following Debugger exit confirmation dialog box (job ID attach debug) is displayed.

Click **Yes** button to exit the Debugger, and terminate the active program.

Click **Yes (Detach)** button to exit the Debugger, while the program execution continues.

Click **No** button to continue using the Debugger.

Figure 3.24 Debugger exit confirmation dialog box (Job ID attach debug).



Chapter 4 List of Debugger Operations

The following table lists the various debug operations with their respective operating instructions.

Table 4.1 List of debugging operations

Debug operation			Instruction
Execution control	Re-execution (Rerun)		- **Rerun** button on the toolbar - **Execution** > **Rerun** on the menu bar
	Continue execution (Continue)		- **Continue** button on the toolbar - **Execution** > **Continue** on the menu bar
	Source level execution	Step	- **Step** button on the toolbar - **Execution** > **Step** on the menu bar
		Next	- **Next** button on the toolbar - **Execution** > **Next** on the menu bar
		Finish	- **Finish** button on the toolbar - **Execution** > **Finish** on the menu bar
	Temporary stop/forced termination	Stop	- **Stop** button on the toolbar - **Execution** > **Stop** on the menu bar
		Kill	- **Kill** button on the toolbar - **Execution** > **Kill** on the menu bar
Stop position operations	Breakpoint	List display	- **Information** > **Action points** > **Breakpoints ...** of debugging operation menu on the menu bar - In the **Entity information** window, on the **Action points** tab, click the **Breakpoints** bar.
		Set	- Click the margin area of the source view panel. - In the **Entity information** window, on the **Action points** tab, click the **Breakpoints** bar. Next, specify the stop position information in the input box, and then click **Set**.
		Delete	- Click the breakpoint in the margin area of the source view panel. - Right-click the breakpoint in the margin area of the source view panel, and then click **Delete**. - In the **Entity information** window, on the **Action points** tab, click the **Breakpoints** bar. Next, right-click the breakpoint, and then click **Delete breakpoint**.
		Enable	- Right-click the disabled breakpoint in the margin area of the source view panel, and then click **Enable**. - In the **Entity information** window, on the **Action points** tab, click the **Breakpoints** bar. Next, right-click the disabled breakpoint, and then click **Enable breakpoint**.
		Disable	- Right-click the breakpoint in the margin area of the source view panel, and then click **Disable**. - In the **Entity information** window, on the **Action points** tab, click the **Breakpoints** bar. Next, right-click the breakpoint, and then click **Disable breakpoint**.

Debug operation			Instruction
	Temporary breakpoint	List Display	- **Information** > **Action points** > **Breakpoints ...** on the menu bar - In the **Entity information** window, on the **Action points** tab, click the **Breakpoints** bar.
		Set	- Click the action point icon for the temporary breakpoint in the source view panel, and then click the margin area.
		Delete	- Click the temporary breakpoint in the margin area of the source view panel. - Right-click the temporary breakpoint in the margin area of the source view panel, and then click **Delete**. - In the **Entity information** window, on the **Action points** tab, click the **Breakpoints** bar. Next, right-click the temporary breakpoint, and then click **Delete breakpoint**.
		Enable	- Right-click the disabled temporary breakpoint in the margin area of the source view panel, and then click **Enable**. - In the **Entity information** window, on the **Action points** tab, click the **Breakpoints** bar. Next, right-click the disabled temporary breakpoint, and then click **Enable breakpoint**.
		Disable	- Right-click the temporary breakpoint in the margin area of the source view panel, and then click **Disable**. - In the **Entity information** window, on the **Action points** tab, click the **Breakpoints** bar. Next, right-click the breakpoint, and then click **Disable breakpoint**.
	Watchpoint	List Display	- **Information** > **Action points** > **Watchpoints ...** on the menu bar - In the **Entity information** window, on the **Action points** tab, click the **Watchpoints** bar.
		Set	- Right-click the variable in the source view panel, and then click **Watch**, **Refer**, or **Referwatch**. - Right-click the local variable on the **Local variables** tab, and then click **Watch**, **Refer**, or **Referwatch**. - In the **Entity information** window, on the **Local variables** tab, right-click the local variable, and then click **Watch**, **Refer**, or **Referwatch**. - In the **Entity information** window, on the **Action points** tab, click the **Watchpoints** bar. Next, specify the variable identifier in the input box, and then click **Set**.
		Delete	- In the **Entity information** window, on the **Action points** tab, click the **Watchpoints** bar. Next, right-click the watchpoint, and then click **Delete watchpoint**.
		Enable	- In the **Entity information** window, on the **Action points** tab, click the **Watchpoints** bar. Next, right-click the disabled watchpoint, and then click **Enable watchpoint**.
		Disable	- In the **Entity information** window, on the **Action points** tab, click the **Watchpoints** bar. Next, right-click the watchpoint, and then click **Disable watchpoint**.
	Barrierpoint	List Display	- **Information** > **Action points** > **Barrierpoints ...** on the menu bar - In the **Entity information** window, on the **Action points** tab, click the **Barrierpoints** bar.

Debug operation			Instruction
		Set	- Click the action point icon for the barrierpoint in the source view panel, and then click the margin area. - In the **Entity information** window, on the **Action points** tab, click the **Barrierpoints** bar. Next, specify the stop position information in the input box, and then click **Set**.
		Delete	- Click the barrierpoint in the margin area of the source view panel. - Right-click the barrierpoint in the margin area of the source view panel, and then click **Delete**. - In the **Entity information** window, on the **Action points** tab, click the **Barrierpoints** bar. Next, right-click the barrierpoint, and then click **Delete barrierpoint**.
		Enable	- Right-click the disabled barrierpoint in the margin area of the source view panel, and then click **Enable**. - In the **Entity information** window, on the **Action points** tab, click the **Barrierpoints** bar. Next, right-click the disabled barrierpoint, and then click **Enable barrierpoint**.
		Disable	- Right-click the barrierpoint in the margin area of the source view panel, and then click **Disable**. - In the **Entity information** window, on the **Action points** tab, click the **Barrierpoints** bar. Next, right-click the barrierpoint, and then click **Disable barrierpoint**.
Variable operations	Display of value (print)		- Right-click the variable in the source view panel, and then click **Print**. - Right-click the local variable on the **Local variables** tab, and then click **Print**. - In the **Entity information** window, on the **Local variables** tab, right-click the local variable, and then click **Print**. - In the **Entity information** window, specify the variable on the **Print** tab.
	Change value (set)		- Right-click the variable in the source view panel, and then click **Set ...**. - Right-click the local variable on the **Local variables** tab, and then click **Set ...**. - In the **Entity information** window, right-click the local variable on the **Local variables** tab, and then click **Set ...**.
	Display feature	List display	- Refer to the **Application structure** panel in the main window.
		Set	- Right-click the variable in the source view panel, and then click **Display**. - Right-click the local variable on the **Local variables** tab, and then click **Display**. - In the **Entity information** window, right-click the local variable on the **Local variables** tab, and then click **Display**. - Right-click in the **Application structure** panel, and then click **Display ...**.
		Delete	- Right-click the variable in the **Application structure** panel, and then click **Delete display**.
Information display	Local variables		- Click the **Local variables** tab in the main window. - **Information** > **Local variables ...**on the menu bar - Click the **Local variables** tab in the **Entity information** window.

Debug operation		Instruction
	Call route	- Click the **Stack trace** tab in the main window. - **Information** > **Stack trace ...**on the menu bar - Click the **Stack trace** tab in the **Entity Information** window.
	Detailed entity information	- Use the **Information** menu on the menu bar. - Click the **Entity information window** button on the toolbar.

Appendix A Considerations for Using the Debugger

This appendix explains the considerations for using the Debugger.

Environment variable names

Environment variables that start with "TRT_" are used by the Debugger. Do not use these environment variables.

Variables display

There might be a delay in the GUI display response in the following cases:

- When the entire value of a large array is to be displayed by using the print feature
- When a program uses a large number of local variables

Debugging jobs that run two or more programs

Jobs (MPMD) that run two or more programs and the process of dynamically generating the MPI cannot be debugged.

Japanese character strings

Japanese character strings must not be specified in windows or dialog boxes of the Debugger. Note that Japanese characters may appear distorted when displayed in the console or the source view panel.

Using the tool runtime daemon for the Debugger

The Debugger uses communication modules, such as the tool runtime daemon, to establish communication between the user terminal and the job process. When a problem occurs in communication processing of the runtime daemon tool, an error message is displayed.

This could be because of the following reasons, so contact the system administrator.

- The runtime daemon tool is not active.
- The port number on which the runtime daemon tool can be run has exhausted.

Job submission

If the Debugger has terminated while a job is submitted and is waiting, the job remains in the waiting state.

The waiting job is terminated at once and starts again. If a problem occurs, delete the job.

Execution time of the job to be debugged

In case of the normal debug mode of the Debugger, the debug process runs as part of the job. The operation time of the Debugger is affected by the execution time of the job. Therefore, if the duration set for the job being debugged expires, the Debugger also terminates.

Corefile debug

When corefile debug is started, the Debugger submits a job to start the debug process internally. This job starts the debugging engine on the compute node as well as the job that the user submits using the pjsub command. The execution time of this job is 24 hours (depending on the system settings). The cost of this job is similar to when a Debugger user submits a typical job.

Attach debug

You cannot use the attach debug mode with a job ID if the job is a sequential program or a parallel MPI program.

Debugging XPF programs

Refer to the "*XPFortran User's Guide*" for information on debugging XPFortran (XPF) programs.

Considerations and limitations of fdb

When started, the Debugger starts the debugging engine (fdb) on the compute node. Therefore, the considerations and the limitations of the fdb also apply to the Debugger. Refer to the document that describes the considerations and limitations of fdb by using the method given below.

Execute the following command on the login node.

```
$ export MANPATH=/$install-dir/man:$MANPATH
$ man fdb
```

$*install-dir*: Directory where the document is stored

Debugger and the abnormal state of the system

If a job is interrupted during debugging, for example, the job execution environment stops or the system is in the abnormal state, the job submitted by the Debugger is terminated. When this happens, the job which was earlier running can restart automatically provided the necessary system settings are configured. However, the Debugger cannot restart debugging the job again.

Therefore, when you start the Debugger on the system that restarts jobs in the abnormal state, specify the --norestart option of the pjsub command in the **Job submit command** box in the **Debugger start setting** window or the job script. This restricts the job to be debugged from being restarted if the system is in the abnormal state.

Display the source file on source view panel

The source file displayed on the source view panel should be placed in the directory at compiling it. And the directory is accessible from the login node.

Please check that the source file is in the directory at compiling it to display it on the source view panel.

The how of debugging by fdb command

You can use the fdb command on the interactive job by the following order.

1. Start the interactive job on the login node by the pjsub command.
   Do not specify the script file.

   Example

   ```
   $ pjsub --interact
   ```

2. Run the target program in the submitted job in the background.

   Example

   ```
   $ mpiexec a.out &
   ```

3. Confirm the ID of the process which you want to debug.
   You can check it by the ps command in the submitted job.

4. Execute the fdb command by specifying the process ID.
   "pid" specifies the process ID of the running process.

   Example

   ```
   $ fdb -p pid
   ```

Start condition of Debugger

In both cases of the following, the Debugger does not start:

- -z option or --help option is specified for pjsub command
- The output message of pjsub command is changed by the script

Debugging COARRAY feature

In the Debugger, each image of COARRAY program is called "rank".

Ranks have a non-negative integer number which starts at 0 and Images have a positive integer number which starts at 1, so "rank number = image index - 1".

Refer to "Fortran User's Guide Additional Volume COARRAY" for details.

Debugging in environment with effective job swap function

Please put the check in the [Interactive job option] check button when debugging begins from the [Debug] tab in the debugger start setting window in the environment with effective job swap function.

It might become impossible to do the debugging operation when debugging starting without putting the check.

Appendix B Notes on Migration from FX10 System to FX100 System

This appendix provides notes on migrating from FX10 system (Generation Number:09 or later) to FX100 system.

For migrating from FX10 system (Generation Number:08 or earlier), refer to "Appendix C Compatibility Information (FX10 System)" also.

B.1 Start condition of Debugger is changed

Refer to "C.1.1 Start condition of Debugger is changed".

Appendix C Compatibility Information (FX10 System)

C.1   Migration to V2.0L10(Generation Number:11)

C.1.1   Start condition of Debugger is changed

a. Changes

The start condition of the Debugger was changed.

[Previous version]

In the following cases, the Debugger starts.

- -z option or --help option is specified for pjsub command
- The output message of pjsub command is changed by the script

[This version]

In the following cases, the Debugger not start.

- -z option or --help option is specified for pjsub command
- The output message of pjsub command is changed by the script

b. Influence

In the following cases, the Debugger not start.

- -z option or --help option is specified for pjsub command
- The output message of pjsub command is changed by the script

c. Coping

Do not start the Debugger specifying -z option and --help option for pjsub command.

Do not start the Debugger changing the output message of pjsub command by the script.

Appendix D Compatibility Information (FX100 System)

D.1 Migration to V2.0L10(Generation Number:02)

D.1.1 Start condition of Debugger is changed

Refer to "C.1.1 Start condition of Debugger is changed".