mpC Workshop

User's Guide

July 2002

Table of Contents

mpC Workshop User's Guide Overview 3
mpC Workshop Features 4
- mpC Workshop Advantages 4
- mpC Workshop Architecture 5
- mpC Daemon 5
- mpC Adviser 5
- Virtual Parallel Machines (VPMs) 6
- Distributed Projects 6
- Distributed Debugging 6
- mpC Workshop Windows 7
  - Text Editor Window 7
  - Output Window 8
  - Workspace Window 8
  - Watch Window 8
  - Quick Watch Window 9
  - Debug Info Window 9
  - Current Cursor Window 10

Working with VPMs 11
- Adding a New VPM 11
- Editing an Existing VPM 12
- Removing VPMs 12
- Selecting a VPM 12

Working with Projects 13
- Creating New Project 14
- Opening an Existing Project 14
- Setting Current Target 14
- Adding/removing Targets 15
- Adding/removing Builds 15
- Adding/removing Files 16
- Compiling 17
- Connecting to Server 18
- Building the Current Target 18
- Running Parallel Programs 19
- Changing settings 19
  - Changing Project Settings 19
  - Changing Target Settings 20
  - Changing Build Settings 20
  - Changing File Settings 20
  - Project Settings Dialog Options 21

Text Editor 26
- About Syntax Coloring 26
- Customizing the Text Editor 26
- Finding and Replacing Text 27
- Navigating in Files 29
- Editing Text 29
- Managing Files 31
  - Creating a New File 31
  - Opening a File 31
  - Saving Files 32
  - Printing a File 32

Debugger 34
    Managing Cursors 34
        Splitting/Merging Cursors 35
        Changing Cursor Color 36
        Setting Network Filter 37
    Moving Commands 38
        Go Command 38
        Go to cursor Command 38
        Step into Command 39
        Step over Command 39
        Step out Command 39
    Managing Breakpoints 39
        Setting Breakpoints 40
        Removing Breakpoints 41
        Disabling/Enabling Breakpoints 42
        Changing Breakpoint Condition 42
        Setting "ignore count" of a Breakpoint 42
    Managing Watchpoints 43
        Setting Watchpoints 44
        Removing Watchpoints 45
        Disabling/Enabling Watchpoints 45
        Changing Watchpoint Condition 45
        Setting "ignore count" of a Watchpoint 46
    Managing Displays 46
        Modifying Displays 47
        Removing Displays 47
        Setting Displays 47
    Debugging programs 48
        Preparing Programs for Debugging 49
        Debugger Windows 50
        Debugger Dialog Boxes 50
        Using Spreadsheet Fields 51
        Viewing the Value of a Variable 51
        Using Break Info window 52
        Managing Breakpoints 53
        Managing Watchpoints 53
        Managing Displays 53
        Running to a Location 53
        Stepping into Functions 54
        Stepping over or out of a Function 54
        Viewing Call Stacks 55
        Working with current cursor 55
        Working with mpC daemon 56
Glossary 57

mpC Workshop User's Guide Overview

This guide explains how to use mpC Workshop, an Integrated Development Environment for the mpC programming language.

The guide consists of the following chapters:

- mpC Workshop features
- Working with Virtual Parallel Machines
- Working with Projects
- Text Editor
- Debugger

mpC Workshop Features

mpC Workshop is a sophisticated tool that allows you to develop mpC programs. mpC Workshop includes Text Editor, debugger, which allows you to debug parallel programs written in the mpC programming language, VPM (Virtual Parallel Machine) management tools and many other useful features.

This chapter covers the following topics:

- mpC Workshop advantages
- mpC Workshop architecture
- mpC Daemon
- mpC Adviser
- Virtual Parallel Machines (VPMs)
- Distributed projects
- Distributed debugging
- mpC Workshop windows

mpC Workshop Advantages

mpC Workshop includes text editor, mpC compiler and source-level parallel mpC debugger. The main advantage of the mpC parallel debugger is that it allows a user to look at a parallel program as a whole, unlike other parallel debuggers, which treat a parallel program as a set of separate processors communicating with each other.

The editor implements all of the standard functions that are expected in a programming editor: source code indenting, syntax coloring, full text search facilities and context help.

The mpC parallel debugger is the main part of mpC Workshop. The mpC parallel debugger has many advantages for debugging of parallel programs over conventional Windows debuggers such as Microsoft Visual Debugger. And what is more the mpC debugger is the first really parallel debugger for Windows platform. It allows you:

- to see a lot of useful information without entering commands. The mpC parallel debugger displays all the important information about a single process, showing the source code, stack trace, and stack frame for the process.
- to debug remote programs over network, even over internet. For example sitting at your cozy apartments in London you can debug a program running on some NT cluster in Los Alamos.
- to handle your executable so that you can control execution of any process and keep track of values of variables in any process of the parallel program.

Another feature of the mpC Workshop is VPM (Virtual Parallel Machine) management tool. A Virtual Parallel Machine represents a network of computers on which an mpC program will be executed. VPM mechanism provides especially useful means for mpC programs debugging and execution.

mpC Workshop also provides facilities for managing distributed projects. If the VPM consists of computers with different platforms then at least one executable must be build for each platform. Distributed project allows you to build all the executables with one click.

mpC Workshop Architecture

mpC Workshop includes a compiler, a run-time support system, a GUI IDE with an advanced syntax-oriented editor, mpC adviser module and parallel debugger. The mpC Workshop allows creating and managing projects, editing and compiling mpC-programs, supports the source-level debugging of mpC programs.

mpC Workshop uses the client-server model with GUI environment on the client side and mpC command-line environment on the server one. The client and server software can be installed independently and in particular case can be installed on single host for local use.

Server part includes the mpC compiler and utilities, mpC Runtime Support System, daemon, libraries and debugger. It requires Microsoft Visual Studio and MPI library to be installed and provides full system functionality including VPM control, compiling, running and debugging of parallel applications.

Client part provides a graphical user interface for server part and supports the full functionality of the system. It contains GUI utilities for VPM control (see Virtual Parallel Machines overview) and program execution and the IDE for parallel programming. The IDE contains the syntax-oriented editor, VPM and project management tools and parallel debugger. It doesn't require any additional software.

Thus the client makes interaction with user and graphical data representation and server itself performs the work. In particular, when the debug session is active the debugger process is executed on server and the debugger GUI sends all debug commands to daemon that passes the command to debugger and returns the result to client GUI.

mpC Daemon

The mpC daemon is a utility that receives commands from the client part of mpC Workshop and initiates the execution of these commands. For example when you click **Rebuild** from the **Build** menu, the command is passed to the mpC daemon and the daemon initiates building process on all builders of the current VPM. Once all the output files have been built the mpC daemon broadcasts them to all nodes of the current VPM. Also the mpC daemon initiates the execution of programs, the creation of VPMs and many other common operations. The computer, on which the mpC daemon is started, runs the process corresponding to the host-processor.

mpC Adviser

mpC adviser is a special feature that allows you to see type and distribution of a statement or expression when you find and fix causes of compiler warning and error messages. Another useful purpose for this feature is to use it to find causes of runtime errors. After you select an expression or statement and click **Advice** from the **Build** menu, information about this expression or statement is displayed in the **Output** window

Virtual Parallel Machines (VPMs)

mpC *Virtual Parallel Machine (VPM)* is a set of computers on which an mpC application is executed. Computers comprising VPM are called VPM *nodes*. One VPM node may execute more than one processes of a parallel application. The VPM concept allows you to treat the computers executing your parallel program as one machine, executing one program – not as a number of computers executing separate processes of the parallel program.

Distributed Projects

To simplify the building process of large parallel programs mpC Workshop provides distributed project mechanism. You create a project and mpC Workshop takes care of building and execution of the targets defined in the project.

A distributed project consists of several *targets* and each target includes one or more *builds*. In general VPM may consists of computers with different platforms, and for every platform at least one executable must be built. With the distributed project mechanism you don't have to think about building of all these executables for several platforms. All output files of the distributed project as if it were usual project with one target file. And each executable is built on the computer with appropriate platform.

Distributed Debugging

The mpC parallel debugger is the main part of mpC Workshop. Debugging is the most difficult part of parallel programs development. When debugging a program you usually want to values of variables or expressions. In a parallel program each variable or expressions generally has as many values as the parallel program has processes. And each process may be executed on a separate computer. So to display values of some variable for all processes the debugger must have information from all the computers running the application. The mpC parallel debugger allows you to see values of any variable or expression for all processes of the debugged program. You can also restrict the number of processes, for which values of expression and variables are displayed, by setting network filter. Another feature of the mpC parallel debugger is debug cursors. The debug cursors mechanism allows you to use the moving commands, i.e. **Go**, **Step in** and so on, not for each separate process but for groups of processes. That makes the debugging process more simple and intuitive.

The mpC parallel debugger allows remote debugging. You can debug programs on cluster that is hundreds miles away from your current location. The only thing you need is a connection between your mpC workshop client and mpC workshop server running on one of machines of the remote network.

Along with the described features the mpC parallel debugger provides you with a number of facilities that make debugging of parallel programs easier than ever.

mpC Workshop Windows

In the mpC Workshop there are windows of two types – document windows and docking windows. The position and size of document windows can be changed within the mpC Workshop window. They can be maximized and minimized. Docking windows, however, attach to the borders of the application window, or float anywhere on your screen.

The only document window is the Text Editor window. All the other windows are docking windows.

The following windows are described in this chapter:

- Text Editor window
- Output window
- Workspace window
- Watch window
- Quick Watch window
- Debug info window
- Current cursor window

Text Editor Window

A window where you write, display, and edit code. You can open as many Text Editor windows as you have modules, so you can easily view the code in different forms or modules, and copy and paste among them.

You can open a Text Editor window from:

- The Workspace window by double clicking the form. (See Workspace window)
- The **File** menu by choosing **Open** command.

Editor Features

You can select parts of text with mouse or by Shift-moving around.

The editor allows you to have unlimited multilevel undo and redo actions per document.

You can customize the way the text appears in the editor. For specified items, you can change the:

- Background and foreground colors
- Type and size of fonts

Editor Elements

Code Pane - the area where you type your code. You can also vary the type and size of indenting and tabs in your code.

Margin Indicator Bar - a gray area on the left side of the Text Editor window where margin indicators such as breakpoints are displayed.

Selection Margin - the white space immediately to the left of the code text where can click to select lines of text.

Horizontal and Vertical Scroll Bars - allow you to scroll the Code pane horizontally and vertically so that you can view the code that extends beyond the edges of the Code pane.

Output Window

Displays status messages at run time or diagnostics information at build time. You can display the Output window by choosing **Output** from the **Toolbars** list in the **View** menu or by pushing the Toggle Output button, .

Elements

Output Pane List - displays a list of output panes to view.

Output Pane - displays the status messages for the selection in the Output Pane list.

Workspace Window

Displays current project, current target of the project and all of the items contained in the project. Tabs at the bottom of the Workspace window allow you to view as the files associated with the project as the current VPM. Each view is hierarchical.

You can rename your projects or project items by right-clicking on the selected item, selecting **Rename** from the context menu, and typing the new name.

Elements

***Target** tab* displays the current target of the project and the hierarchical list of the builds and files associated with the current target.

***VPM** tab* displays the current VPM and hostnames of the computers associated with the current VPM.

You can expand and collapse the project lists and view the projects and items included in the project by clicking plus (+) or minus (-) to the left of the project name.

Watch Window

Displays the values of selected variables or watch expressions according to the current network filter (see Setting network filter). This window is updated at a breakpoint or watchpoint only. The variables and expressions displayed in the **Watch** window can be specified either in the **Expression** tab of the **Debug object manager** dialog (see Setting displays) or in the **Quick Watch** dialog.

Elements

The column **Name** displays a list of variables or expressions to be evaluated according to the settings of the current network filter.

The next columns display the values of selected variables or expressions for the processes selected in the current network filter. Number of columns in the pane depends on settings of the current network filter. Headings of columns contain global numbers of virtual processors and their coordinates within the network(s) chosen in the current network filter.

Quick Watch Window

You can use the Quick Watch window to quickly examine values of expressions and variables or to add a variable or expression to the Watch window.

The **Quick Watch** dialog box contains a text box, where you can type a variable name or an expression, and a spreadsheet field that displays the current values of the variable or the expression that you specified according to the settings of network filter for the **Watch** window. On the spreadsheet there are displayed names of the variables and expressions, and their values on the processes specified by the current network filter.

Debug Info Window

Displays debugging information. Contains three tabs – **Debug cursors** tab, **Nets** tab and **Call stacks** tab.

Debug cursors tab	Displays debug cursors (see Cursors overview) according to the settings of the current network filter.
Nets tab	Displays a list of the currently existing network objects as well as of processes of a parallel program.
Call stacks tab	Displays call stacks of processes of a parallel program according to the settings of the current network filter. The window is divided into two parts. The box at the top of the window contains drop-down list of processes of the parallel program with colors of cursors they belong to. You can select a process from the list and at the lower pane of the window you will see a call stack for the selected process.

You can show or hide any of the tabs by choosing or deselecting them from the **Debug windows** list in the **View** menu.

Current Cursor Window

Displays current cursor. Current cursor is the cursor, current position of which is displayed in the **Text Editor** window each time when the program stops in break mode. For more information about cursors consult Cursors overview.

You can expand the current cursor by clicking (+) box, which opens into a list of processes comprising the current cursor. You can also change current cursor color (see Changing cursor color).

Working with VPMs

mpC *Virtual Parallel Machine (VPM)* is a set of computers on which an mpC application is executed. Computers comprising VPM are called VPM *nodes*. One VPM node may execute more than one processes of the parallel application. VPM description must have one or more entries of the following form:

```
<node_name> [processors(opt)] [processes(opt)]
```

The token `processors` specifies number of processors on the node and the token `processes` specifies the number of processes this VPM node executes. A VPM node can be either *node* or *builder*. A builder is a VPM node that not only runs processes of a parallel program but also builds executables, while a node just runs processes. A computer can be a builder within some VPM only if Microsoft Visual C++ is installed on this computer. So any computer with Visual C++ may be a builder within some VPM and a node within some other VPM. If VPM is heterogeneous, i.e. if it consists of computers with different platforms, then the number of builders in such a VPM must be no less than the number of computer platforms within the VPM – at least one for each platform. If all computers comprising a VPM have the same platform than one builder is enough. You can build or run an MPC program only if some VPM that includes your computer is open. Selecting a VPM you close the current VPM, if any, and open the selected VPM.

To manage VPMs you can perform the following actions:

- Adding a new VPM
- Editing an existing VPM
- Removing VPMs
- Selecting a VPM

To perform any of the actions described in this chapter the client part of mpC Workshop must be connected to server part (see mpC Workshop architecture, Connecting to server).

Adding a New VPM

To add a new VPM:

1. On the **Build** menu click **VPM settings**. If the item **VPM settings** is disables connect to server (see Connecting to server). The **VPM manager** dialog appears.
2. The **VPM settings** list box displays the list of available VPMs. The **VPM nodes** list box displays description of the selected VPM. To add a new VPM click the **New** button. The **Edit VPM** dialog appears. After performing appropriate actions click **OK** on the **Edit VPM** dialog (see Editing an existing VPM).
3. Select the new VPM from the **VPM settings** list and click **Select**.

Editing an Existing VPM

To edit an existing VPM:

1. On the **Build** menu click **VPM settings**. If the item **VPM settings** is disables connect to server (see Connecting to server). The **VPM manager** dialog appears.
2. Select the VPM you want to edit from the **VPM settings** list. Click **Edit**. The **Edit VPM** dialog appears.
3. If you create a new VPM the **VPM name** text box is enabled and you must type name for the new VPM in this text box. When you edit an existing VPM the **VPM name** text box is disabled. In the text field you can edit VPM description. To add a new entry to the VPM description you can either manually type hostname, number of processors on the host and number of processes of a parallel program that will be executed on this host, or click **Add nodes**, select the host you want to add to VPM, click **OK** and change the number of processes and the number of processors of necessary. Number of processes and number of processors are optional parameters. By default they equal to 1.
4. Click **OK** or click **Cancel** if you changed your mind.

Removing VPMs

To remove a VPM:

1. On the **Build** menu click **VPM settings**. If the item **VPM settings** is disables connect to server (see Connecting to server). The **VPM manager** dialog appears.
2. Select the VPM you want to remove from the **VPM settings** list. Click **Remove**.
3. To close dialog click **Close**.

Selecting a VPM

You must select a VPM on which you build and run programs.

To select a VPM

1. On the **Build** menu click **VPM Settings**. The **VPM Manager** dialog appears.
2. Select a VPM from the left list. The right list displays computers and numbers of processes for the selected VPM.
3. Click **Select**.

Working with Projects

To simplify the building process of large parallel programs mpC Workshop provides distributed project mechanism. You create a project and mpC Workshop takes care of building and execution of the targets defined in the project.

Target is a build unit that includes one or more builds. Each *build* defines settings required for building of a particular binary file and specifies the files, which are necessary for building the binary. Each binary file may be built for a specific platform. All binary files are built on "builder" computers or *builders*. Each computer within a VPM (see Virtual Parallel Machines overview) can be either *node* or *builder*. A node only runs one or more processes of a parallel program, while a builder both runs processes and builds executables. In general a parallel program may be executed on several platforms and, consequently, the executable must be built for each platform. That’s one of the reasons why a target may contain more than one builds. Another reason is that the executable may depend on some library and the settings required for building of this library can be defined in another build.

By default an mpC project contains two targets – "Debug" and "Release". You can add new targets to the project or change settings of the existing targets. There can be only one current target. You can add builds to the current target or remove them from it. Also you can add files to or remove files from any build of the current project.

Building process consists in building binary files for each build of the current target.

This chapter covers how to manage distributed projects by use of the following actions:

- Creating new project
- Loading existing project
- Setting current target
- Adding/removing targets
- Adding/removing builds
- Adding/removing files
- Compiling
- Connecting to Server
- Building the current target
- Running parallel programs

Creating New Project

To create a new project:

1. On the **Project** menu click **New**. The **New Project** dialog appears.
2. Specify Output file name and Project name.
3. Specify the **Location** of project files.
4. Specify the **Target directory**. Target directory of a project is a directory, which contains files required for building output files. The target directory is contained in the directory specified by the environment variable %MPCLOAD% on each builder of the current VPM (see Virtual Parallel Machines overview). You can specify directory contained in some other directory contained in the %MPCLOAD% directory. Target directory mustn't be Windows "full path". That is "dir1\dir2" and "dir1\dir2\dir3" are correct directory names, while "c:\dir1" and "c:\dir1\dir2" are incorrect. If the directories don't exist they are created.

Opening an Existing Project

To open an existing project

1. On the **Project** menu, click **Open**.
2. In the **Open** dialog box browse to locate the project you want to open. Project configuration file has ".mpf" extension.
3. Click the **Open** button.

When you open a project the current project is closed.

Setting Current Target

When you set the current project target, subsequent build commands act on the current target and build its output.

To set the active project configuration

- On the **Project** menu, select **Set Current Target** and either double-click on a target or select a target and click **OK.**

Adding/removing Targets

To add a target to project

1. On the **Build** menu click **Targets**. The **Edit Project** dialog box appears.
2. Click **Add**. Enter the target name.
3. Select in the **Target** box the new configuration. Click **Edit**.
4. In the **Edit Target** dialog select **Contains builds** from the list at the top and click **Add** to add a build to the target (see Adding/removing builds). If you want to remove an existing build from the target select the build and click **Remove**. After adding builds click **OK**.
5. Click **OK** in the **Edit Project** dialog box.

To remove a target from project

1. On the **Build** menu click **Targets**. The **Edit Project** dialog box appears.
2. Select in the **Target** box the target you want to remove. Click the **Remove** button.
3. Respond **Yes** to remove the target.
4. Click **OK**.

Adding/removing Builds

A build consists of settings that determine the characteristics of the final output file.

To add a build to a project

1. On the **Build** menu click **Targets**. The **Edit Project** dialog box appears.
2. In the **Target** box select the target to which you want to add builds. Click **Edit**.
3. In the **Edit Target** dialog select **Contains builds** from the list at the top and click **Add** to add a new build to the selected target. The **Add Build** dialog appears.
4. Choose from the list at the top of the dialog one of the following items: **New build**, **Another build from project**, **Build on disk.** If you choose **New build,** type the name for the new build in the text box. You can create a new build either with default settings or you can copy settings for the new build from one of existing builds by checking the **Copy data from existing build** checkbox and choosing one of builds from the **Select build** list. If you choose **Another build from project,** list of project builds not included to the selected target appear. You can choose one of the builds. And if you choose **Build on disk** the **File name** text box appears. You can either enter a file name with full path – the file you are specifying must have certain format and ".bsf" extension – or click the **Browse** button to the right of the text box and browse to locate the file you want to specify.
5. Click **OK** in the **Add Build** dialog.

6. If you want to add one else build click Add in the **Edit Target** dialog.
7. Click **OK** in the **Edit Target** dialog and then in the **Edit Project** dialog.

To remove a build from a project

1. On the **Build** menu click **Targets**. The **Edit Project** dialog box appears.
2. In the **Target** box select the target to which you want to add builds. Click **Edit**.
3. In the **Edit Target** dialog select **Contains builds** from the list at the top and click **Add** to add a build to the target.
4. Select the build you want to remove from the **Builds** box, click **Remove** and respond **Yes**.
5. Click **OK** in the **Edit Target** dialog and then in the **Edit Project** dialog.

Adding/removing Files

You can add files either to the whole project or to the current target or to a build. If you add a file to the whole project the file is added to all builds of all targets of the project. If you add a file to the current target the file is added to all builds of the target. When removing files from project you can specify from what builds the file must be removed.

To add a file to a project

1. In the **Workspace** window on the **Target** tab right-click the current project. From the context menu choose **Add files to project.**
2. In the **Open** dialog browse to the files you want to add, select them and click **Open**. Files added to all builds of the project.

To add a file to the current target

1. In the **Workspace** window on the **Target** tab right-click the current target. If the hierarchical list isn’t expanded click (+) to the left of the project name or double-click the project name. From the context menu choose **Add files to target’s builds**.
2. In the **Open** dialog browse to the files you want to add, select them and click **Open**. Files added to all builds of the target.

To add a file to a build

1. In the **Workspace** window on the **Target** tab right-click the build to which you want to add files. From the context menu choose **Add files to build**.
2. In the **Open** dialog browse to the files you want to add, select them and click **Open**.

To remove a file from a build

- Select the file you want to remove from a build and press the **Delete** button on your keyboard. Respond **Yes** to remove.

  - or –

- Right-click the file you want to remove and choose **Delete file** from the context menu. Respond **Yes** to remove.

To remove a file from more than one builds

1. Right-click the file you want to remove and choose **Delete file from builds** from the context menu.
2. Select builds from which you want to remove the file and click **Selected builds** or click **All builds** to remove the file from all builds of the project.
3. Respond **Yes** to remove.

Compiling

The first step in the process of building a program is creating source code files with the code statements in header and/or source files. When you invoke the mpC compiler, for each mpC source file the compiler generates a file containing C code.

Compilation is performed on all builders of the current VPM (see Virtual Parallel Machines overview) and information about the compilation is displayed in the **Output** window, including warning and error messages. The **Output** window connects you to the line of code where the message is generated. To see the code that generates a diagnostic message double-click a diagnostic message in the Output window. In the **Text Editor** window a pointer shows the line, which generates the diagnostic message.

To find causes of the diagnostic messages you can use the mpC adviser. The mpC adviser allows you to see type and distribution of a statement or expression. Select a statement or expression value of which you want to see and click **Advice** from the Build menu. Information about the statement or expression is displayed in the **Output** window.

Before compilation you can change settings for the file you are going to compile (see Changing File Settings). To start compilation your mpC Workshop client must be connected to server (see Connecting to Server).

To start compilation

1. Open the source file that you want to compile. If it is already open, click that window to move the focus there.

   - or -

   Select the file in the **Workspace** window.
2. Click **Compile** *filename* from the **Build** menu.

Connecting to Server

To build and run programs your mpC Workshop client must be connected to the mpC server. The mpC server distributes the files necessary for building binaries to appropriate builders. In fact your mpC Workshop client connects to the mpC daemon and the mpC daemon interacts with the mpC server. Before connecting to server make sure that on the machine you are going to connect to, the mpC daemon is started. If the daemon isn't started you must establish telnet session to this computer and start the mpC daemon with the *mpcd* command. You can also start the mpC daemon locally, on your machine. To do this you needn't open telnet session – just execute the *mpcd* command from the command line. Note that to start the mpC daemon locally the mpC Workshop server must be installed on your machine.

To connect to server

1. On the **Build** menu click **Connect to server**. The **Connect to server** dialog appears.
2. Enter a hostname of the machine, on which the mpC server is running.

   - or -

   Choose one of the available machines from the list.

   - or -

   Click **Advanced**, and enter IP-address and port in the text boxes at the bottom of the dialog.

   - or -

   Click **Local server**. You can do this only if the mpC Workshop server is installed on your machine. The mpC daemon is launched on the local machine and the mpC Workshop client connects to this daemon. If you check the **Hidden** box no new window appears for the launched daemon. If you check the **Once** box the local mpC daemon is killed when you close your mpC Workshop client or when you disconnect from server.
3. Click **OK**. If connection can't be established for some reason the message box appears. Click **OK** to specify another hostname.

If your workshop is connected to the server then at the bottom of the main window the following picture is displayed: . Otherwise the picture is the following .

Building the Current Target

To build a target means to build all binary files on all builders of the current VPM and then distribute the binaries to all nodes. Each builder sends binaries to the nodes with the same platform (see Distributed projects overview). Only one target can be build.

To build the current target your mpC Workshop client must be connected to the mpC server (see Connecting to server) and one of VPMs must be selected (see Selecting VPMs).

To build the current target

- On the **Build** menu click **Build** to process only the files in the current target that have changed since the last build.

  -or –

- On the **Build** menu click **Rebuild** to process all the files in the current target.

The output of the build process you can see in the **Output** window.

Running Parallel Programs

When you have completed building the current target, you can start the program from mpC Workshop. You can also run programs in the integrated debugger. To run a program your mpC Workshop client must be connected to the mpC server and one of VPMs must be selected.

To run a parallel program

- On the **Build** menu, click **Execute.**

To run a program in the integrated debugger

1. On the **Debug** menu click **Start Debug**. After clicking **Start Debug** all the processes of the parallel program belong to one green cursor (see Cursors overview) corresponding to the first operator in the main function.
2. On the **Debug** menu choose either **All** or **Green** from the **Go** list. As all the processes belong to one cursor the result is identical. If you have a source file open and it has the focus, choose either **All** or **Green** from the **Go to Cursor** list. Also you can click **Step Into**.

Changing settings

Changing Project Settings

To change project settings

1. Click **Settings** on the **Project** menu or right-click the project in the **Workspace** window and choose **Settings** from the context menu. The **Project Settings** dialog appears.
2. In the **Project Settings** dialog on the **Settings For** pane select the project. Only the **General** tab is displayed. For project you can only change a builder on which all targets are built. The list box displays host names and number of processes for the current VPM. From the drop-down list you can choose a builder computer on which all binaries for all targets will be built.
3. Click **OK**.

Changing Target Settings

To change target settings

1. Click **Settings** on the **Project** menu or right-click the current target in the **Workspace** window and choose **Settings** from the context menu. The **Project Settings** dialog appears.
2. In the **Project Settings** dialog on the **Settings For** pane select a target. Only the **General** tab is displayed. In the **Executable name** text box you can type the name of the output file for the selected target. In the **Program arguments** text box you can type command line arguments for the program.
3. Click **OK**.

Changing Build Settings

Each build of the project can have its own settings specifying characteristics of the final output file for this build. In general, settings for a build affect all files belonging to that build. However, for each individual file you can specify settings overriding build settings.

To change build settings

1. Click **Settings** on the **Project** menu or right-click a build of the current target in the **Workspace** window and choose **Settings** from the context menu. The **Project Settings** dialog appears.
2. In the **Project Settings** dialog on the **Settings For** pane select the build, for which you want to modify settings. When a build is selected the dialog has four tabs – **General**, **MPC**, **C**, **Link**. Select the options that you want to apply to the build (see Project Settings dialog Options).
3. Click **OK**.

Changing File Settings

For each individual file you can specify settings overriding build settings. If the file belongs to several builds then within each new build it may have different settings. That is, if change settings for a file belonging to some build and this file belongs to another build than within that another build settings for this file aren't changed.

To change file settings

1. Click **Settings** on the **Project** menu or right-click a file in the **Workspace** window and choose **Settings** from the context menu. The **Project Settings** dialog appears.
2. In the **Project Settings** dialog on the **Settings For** pane select the file, for which you want to modify settings. Remember that when you change settings the same file belonging to two different projects is treated as two different files. When a file is selected the dialog has three tabs – **General**, **MPC**, **C**. Select the options that you want to apply to the file (see Project Settings dialog Options)
3. Click **OK.**

Project Settings Dialog Options

Here are options and fields of the Project Settings dialog. All that described in this topic can be applied for files and builds only.

General tab

When changing build settings

In the **Target path** text box you can specify the directory containing all files that are necessary for building the output file for this build. The target directory is contained in the directory specified by the environment variable %MPCLOAD% on each builder of the current VPM (see Virtual Parallel Machines overview).

In the **Target C files path** you can specify the directory containing the C files generated by mpC compiler. The directory must be contained in the directory specified in the **Target path** field.

In the **Target Obj files path** you can specify the directory containing the .obj files generated by C compiler.

By checking or clearing the **Broadcast output file** check box you can define whether the output file of this build must be broadcasted to other computers of the current VPM or not.

When changing file settings

The **Target path** text box has the same meaning when you change file settings as it has when you change build settings.

The **Exclude from build** check box defines whether this file takes part in build process or not.

The **Use own settings** check box defines whether this file’s own settings will be used for building output file or the settings of the build to which this file belongs.

The **Use own target path** check box defines whether the path specified in the **Target path** text box will be used for this file or the build’s, to which this file belongs, target path.

MPC tab

The same for both file and build settings

From the **Category** dropdown list you can choose either **General** or **Warnings**.

If you choose the **General** category the **Parser mode** dropdown list and the **Generate debug info** check box are displayed.

You can choose one of the following parser modes: **SHORT**, **ANSI**, **LONG**, and **ALL**. By default the **SHORT** mode is used. This mode allows using only the short form of mpC keywords. The **ANSI** mode allows to use only **ANSI** C keywords. The **LONG** mode allows to use only the full form of mpC keywords. The **ALL** mode allows to use both forms of mpC keywords. For example, 'net' and `mpc_net' are identifiers in the **ANSI** mode and mpC keywords in the **ALL** mode. `net' is an identifier in the **LONG** mode and a mpC keyword in the **SHORT** mode. Finally, `mpc_net' is an identifier in the **SHORT** mode and an mpC keyword in the **LONG** mode. Presence of these modes supports the compatibility with previously written C code.

The **Generate debug info** check box specifies whether mpC compiler generates debug information or not.

From the **Warnings** dropdown list you can choose **Off**, **Default**, **All** or **Customize**.

Off	No warning messages are printed
Default	Only important messages are printed
All	All warning messages are printed
Customize	You can specify which warning messages are on and which are off

If you choose the **Customize** category you can control the amount and kinds of warnings produced by the mpC compiler using selections in the **Warnings** list box:

Incorrect declarations	Warnings about the (possible) incorrectness of declarations (default)
Incorrect initialization	Warnings about the incorrect initialization (default)
Uncertain size of operands	Warnings produced when the compiler can't check sizes of operands. As example, when one of operands of assignment operator is distributed over the (sub)network which size can't be evaluated in compile time, and the other operand is an undistributed vector
Uncertain distribution of operands	Warnings produced when the compiler can't check the distribution of operands. As example, when one of operands of binary operator is distributed over the (sub)network which size can't be evaluated in compile time (default).
Location or usage of statements	Warnings about the location or usage of statements (default).
Declaration or usage of statements	Warnings about the declaration or usage of labels
Arguments of function or network types	Warnings regarding the declaration/usage of formal parameters and arguments of functions or network types

C tab

The same for both file and build settings

You can choose one of the following categories from the **Category** dropdown list:

General	You can choose warning level and optimizations level, as well as specify preprocessor definitions
Code generation	You can specify the processor, for which C code will be optimized by the compiler, and structure member alignment
Optimizations	You can either specify optimization the C compiler will do or disable all optimization
Preprocessor	You can specify symbols you want to define as well as symbols you want to undefine. Also you can specify additional include directories.

The options that can be specified on this tab affect C files only.

These options control the number of warning messages produced by the compiler. If you choose the **General** category, you can choose one of these options from the **Warning level** dropdown list.

None	Turns off all warning messages.
Level 1	Displays severe warning messages. This is the default.
Level 2	Displays a less-severe level of warning message than level 1.
Level 3	Displays less-severe warnings than level 2, such as warnings about function calls that precede their function prototypes. Level 3 is the most sensitive warning level recommended for production purposes.
Level 4	Displays informational warnings, which in most cases can be safely ignored. Level 4 should be used occasionally to provide "lint" level warnings and is not recommended as your usual warning level setting.
Warnings As Errors	Treats all warnings as errors. If there are any warning messages, an error message is emitted and compilation continues.

The Optimizations options determine how the C compiler fine-tunes the performance of your program. Four of the five optimization categories (**Default**, **Disable (Debug)**, **Maximize Speed**, and **Minimize Size**) in the Optimizations drop-down list box in the **Optimizations** category require no further optimization on your part. If you select the fifth optimization category, **Customize**, you can set specific optimizations using the selections in the **Optimizations** list box.

Minimize Size	Creates the smallest code in the majority of cases.
Maximize Speed	Creates the fastest code in the majority of cases.
Assume No Aliasing	Tells the compiler that your program does not use aliasing. An alias is a name that refers to a memory location that is already referred to by a different name. Using this option allows the compiler to apply optimizations it couldn't otherwise use, such as storing variables in registers and performing loop optimizations.
Assume Aliasing Across Function Calls	Tells the compiler that no aliasing occurs within function bodies but might occur across function calls. After each function call, pointer variables must be reloaded from memory.
Global Optimizations	Provides local and global optimizations, automatic-register allocation, and loop optimization.
Disable (Debug)	Turns off all optimizations in the program and speeds compilation. This option simplifies debugging because it suppresses code movement.
Generate Intrinsic Functions	Replaces some function calls with intrinsic or otherwise special forms of the function that help your application run faster. Programs that use intrinsic functions are faster because they do not have the overhead of function calls, but may be larger because of the additional code created.
Improve Float Consistency	Improves the consistency of floating-point tests for equality and inequality by disabling optimizations that could change the precision of floating-point calculations.
Favor Small Code	Minimizes the size of .EXE files by instructing the compiler to favor size over speed. The compiler can reduce many C constructs to functionally similar sequences of machine code. Occasionally these differences offer trade-offs of size versus speed. If you do not select this option, code may be larger and may be faster.

Favor Fast Code; Default	Maximizes the speed of .EXE files by instructing the compiler to favor speed over size. The compiler can reduce many C constructs to functionally similar sequences of machine code. Occasionally these differences offer trade-offs of size versus speed.
Frame-Pointer Omission	Suppresses creation of frame pointers on the call stack. This option speeds function calls, because no frame pointers need to be set up and removed. It also frees one more register, **x86 Specific —**>EBP on the Intel 386 (or later), **END x86 Specific** for storing frequently used variables and subexpressions.
Full Optimization	Combines optimizing options to produce the fastest possible program.

Link tab

For build settings only

Linker is a 32-bit tool that links object files and libraries to create a 32-bit executable.

On the **Link** tab you can choose one of the following categories from the **Categories** dropdown list:

General	You can specify name of output file for a build and libraries that are necessary for building this file. Also you can specify whether debug information will be generated or not.
Input	You can specify libraries to be used when building the output file, libraries to be ignored when building the output file, symbols to add to the symbol table and additional library path.
Output	In the **Reserve** text box you can specify the total stack allocation in virtual memory. The optional value specified in the **Commit** text box is subject to interpretation by the operating system. A higher *commit* value saves time when the application needs more stack space, but increases the memory requirements and possibly the startup time. Specify the *reserve* and *commit* values in decimal or C-language notation. The **Base Address** option sets a base address for the program, overriding the default location for an .EXE file (at 0x400000).

Text Editor

The mpC Workshop environment includes an integrated text editor to manage, edit, and print source files. Most of the procedures for using the editor should seem familiar if you have used other Windows-based text editors. With the Text editor, you can:

- Perform advanced find and replace operations in a single file or multiple files.
- Select lines, multiple lines, or columns.
- Manage the source window.

The following topics are discussed in this chapter:

- About syntax coloring
- Finding and Replacing Text
- Navigating in files
- Editing text
- Managing Files

About Syntax Coloring

Syntax coloring uses different colors for various code elements, such as keywords or comments. This coloring gives you visual cues about the structure and state of your code. Syntax coloring is based on the language of the source file. It is global; once you set syntax coloring for a particular language, all files of that language are colored the same way.

Customizing the Text Editor

The mpC Workshop integrated Text editor supports syntax coloring based on the mpC language. Syntax coloring uses different colors for various code elements, such as keywords or comments. This coloring gives you visual cues about the structure and state of your code.

You can set font style, size, and color for each code element to suit your preferences and work habits. You can also set fonts in the left margin of the Text editor window.

To change a font style, size, or color

1. From the **Tools** menu, choose **Options.**
2. In the **Category** box, select the component of the **Text editor** window you want to format. To change appearance of the source code choose **Source Windows**, to change font in the left margin select **Left Margin**.
3. In the **Font** box, select the font you want. The **Font** box displays the different fonts installed on your system. The text sample in the sample box changes to the font you select.

4. In the **Size** box, select the font size you want. The **Size** box displays the sizes available for the selected font. The text sample in the sample box changes to the size you select.
5. In the **Colors** box, select the type of text you want to color.
6. In the **Background** box, select a background color; in the **Foreground** list box, select a foreground color.
7. Click **OK**.

The font and size settings apply to everything within the selected category, while the foreground and background color settings apply only to the selected element of that category.

To change a tab size

1. From the **Tools** menu, choose **Options**.
2. In the **Tab size** of the **Editor** tab specify tab character size. You can also choose if the appropriate number of space will be inserted instead of tab characters.
3. Click **OK**.

Finding and Replacing Text

The Text editor supports full string searching. With full string searching, you specify the entire search string before the search begins. You can search for text in a single source file or in multiple files.

With the **Find** command, you can search the active window or all open documents for the following types of text strings:

- **Whole Word Match** Matches all occurrences of a text string not preceded or followed by an alphanumeric character or the underscore (_).
- **Case Match** Searches for text that matches the capitalization of the text string.

To find or replace a text string

1. Move the insertion point to where you want to begin your search. The editor uses the location of the insertion point to select a default search string.
2. From the **Edit** menu, choose **Find**. The **Find and Replace** dialog box appears.
3. In the **Find what** box, type the search text. You can also use the drop-down list to select from a list of the most recently used search strings.
4. If you want to replace the string specified in the **Find what** box, type the replacement text in the **Replace with** text box. You can also use the drop-down list to select from a list of the most recently used replace strings
5. Select any of the **Find** options.

6. To start the search, click the **Find/Replace** button. The **Find and Replace** dialog box disappears and the **Find/Replace** dialog box appears when the search begins.

   - or -

   To replace all occurrences of the search string with the replace string click **Replace All**.

7. To continue search, you can either click **Find** in the **Find/Replace** dialog box or close the **Find/Replace** dialog box and use the **Find Next** or **Find Previous** shortcut keys. The shortcut key for **Find Next** is F3; the key combination for **Find Previous** is SHIFT+F3.

   - or -

   To replace the current occurrence of the searched string click **Replace**.

To start a find without using the Find dialog box

- Type or select a search string in the **Find** box on the Standard toolbar, and press ENTER.

To find a text string in multiple source files

1. From the **File** menu, choose **Find In Files**.
2. In the **Find what** box, type the search text. You can also use the drop-down list to select from a list of the most recently used search strings.
3. In the **In files/file types** box, select the file types you want to search. You can use the drop-down list to select from common file types or to type text specifying other file types.
4. In the **In folder** box, select the primary folder that you want to search. Click the **Browse** button to display the **Choose Directory** dialog box if you want to change drives and directories.
5. If necessary, select one or more of the **Find** options.
6. Click the **Find/Replace** button to begin the search. The **Output** window displays the list of file locations where the text string appears. Each occurrence lists the fully qualified filename, followed by the line number of the occurrence and the line containing the match.
7. To open a file containing a match, double-click the entry in the **Output** window. An editor window containing the file opens, with the line containing the match selected. You can jump to other occurrences of the text string by double-clicking the specific entries in the **Output** window.

To replace a text string in multiple source files

1. From the **File** menu, choose **Find In Files.**
2. In the **Find what** box, type the search text. You can also use the drop-down list to select from a list of the most recently used search strings.
3. In the **Replace with** text box, type the replace string. You can also use the drop-down list to select from a list of the most recently used replace strings.
4. In the **In files/file types** box, select the file types you want to search. You can use the drop-down list to select from common file types or to type text specifying other file types.
5. In the **In folder** box, select the primary folder that you want to search. Click the **Browse** button to display the **Choose Directory** dialog box if you want to change drives and directories.
6. If necessary, select one or more of the **Find** options.
7. Click the **Find/Replace** button to begin the search. The **Find/Replace** dialog box appears.

   - or -

   Click **Replace All** to replace all occurrences of the searched string.
8. In the **Find/Replace** dialog box you can click either **Find** to go to the next occurrence of the searched text or **Replace** to replace the current occurrence.

When you jump to a found string location specified in the **Output** window, the corresponding source file is loaded if it is not already open in the editor.

Note: You can display the output from your last multiple-file search done during your current session by choosing the **Output** command from the **View** menu and by choosing the **Find In Files** tab in the **Output** window.

Navigating in Files

The Text editor provides a variety of methods to move around in a source file. In addition to the standard Windows navigation mechanisms, the Text editor includes an assortment of commands that enable you to move to almost any location in a file.

You can move between open files with the **Window** menu, or by pressing CTRL+TAB or CTRL+SHIFT+TAB.

Editing Text

With the Text editor, you can cut, copy, and paste selected text using menu commands. You can also undo and redo selected editing actions.

The Text editor provides the following editing commands:

- Cutting, copying, pasting, and deleting text
- Undoing and redoing editing actions

All editing commands require a selection in order to work. Some of the commands can make a selection based on the current cursor location; otherwise, the default selection will be the character adjacent to the cursor. For example, the **Delete** command removes the character to the right of the cursor if there is no selection.

When you cut text from the file, the text is removed from your file and placed on the Clipboard. When you delete text from the file, the text is removed from your file, and the Clipboard is not used. All Windows applications share the same Clipboard. Commands that use the Clipboard overwrite whatever was previously placed onto the Clipboard by other commands or other Windows applications.

You can edit your text using the following actions.

Cut	Removes selected text from the active window.
Copy	Duplicates selected text in the active window.
Paste	Pastes cut or copied text into an active window.
Delete	Deletes text without copying it to the Clipboard.
Undo	Restores the text.
Redo	Reapplies the prior edit.

To cut or copy and paste text

1. Select the text you want to cut or copy.
2. From the **Edit** menu, choose **Cut** or **Copy**. The cut or copied text is placed onto the Clipboard and is available for pasting.
3. Move the insertion point to any source window where you want to insert the text.
4. From the **Edit** menu, choose **Paste**.

To delete text

1. Select the text you want to delete.
2. From the **Edit** menu, choose **Delete**. The deleted text is not placed onto the Clipboard, and cannot be pasted.

Use the **Undo** command to undo previous editing actions. Use the **Redo** command to reapply editing actions that have been undone. **Redo** is unavailable unless you have used the **Undo** command.

To undo an edit action from the **Edit** menu, choose **Undo.**

To redo an edit action from the **Edit** menu, choose **Redo**.

You can select lines, multiple lines, and column blocks of text to cut, copy, delete, indent, and unindent. Most of the selection commands have extensions (the word "Extend" is appended to the name of the command) that move the cursor and extend the selection. By default, these commands are bound to the same key combination as the primary selection command plus the SHIFT key.

To select a line of text

- In the selection margin, point to the beginning of the text you want to select and click the left mouse button.

To select multiple lines of text

1. In the selection margin, point to the beginning of the text you want to select.
2. Drag either up or down to select the lines of text.

Managing Files

Creating a New File

The **New** command creates a new source file. Creating a source file does not affect other open source files. You can create source files of many types, including mpC header or source files.

To create a new source file click **New** from the **File** menu. A new editor window is opened for the new file.

Opening a File

When you open a source file, its name is added to the **Window** menu. You cannot use the **Open** command on the **File** menu to open another copy of an open source file.

To open a file

1. From the **File** menu, choose **Open**.
2. Select the drive and directory where the file is stored.
3. Specify the types of files to display in the **Files of type** box. Files with the chosen extension are displayed in the list box. The **Files of type** box initially lists commonly used file extensions.
4. Select a filename, then click **Open**.

You can also open a file by double-clicking the file icon in the **Target** tab of the **Workspace** window.

The names of the four most recently opened files are displayed at the end of the **File** menu. To open one of these files, choose its name from the menu.

Saving Files

To save a file

1. Switch to the source window.
2. From the **File** menu, choose **Save**. If you already named the file, the **Save** command saves changes without displaying the **Save As** dialog box. If your file is unnamed, the **Save As** dialog box appears.
3. In the **File name** box, type the filename.
4. Select the drive and directory where you want to save the file.
5. Click **Save**.

To save all open files

- From the **File** menu, choose **Save All**.

To save selected open files

1. From the **Window** menu, choose **Windows**.
2. Select one or more files from the file list.
3. Click **Save**.
4. Click **Cancel**.

You can also save another copy of an existing file. This procedure is useful for maintaining revised copies of a file while keeping the original unchanged.

To save a new file or another copy of an existing file

1. Make the file active by clicking the source window.
2. From the **File** menu, choose **Save As**.
3. In the **File name** box, type the filename.
4. Select the drive and the directory where you want to save the file.
5. Click **Save**.

Printing a File

With the Text editor, you can print selected text or a complete file. Text is printed in the default font for the printer if the default editor font is used. Otherwise, the text prints with the selected editor font, if that font is available on the printer.

To print selected text

1. Select the text you want to print.
2. From the **File** menu, choose **Print.** The **Print** dialog box appears. The **Print** dialog box appears. Choose the **Selection** option. In the **Number of copies** box, enter the number of copies you want to print. By default number of copies is 1.
3. Click **OK**.

To print an entire file

1. Move the focus to the source file you want to print.
2. From the **File** menu, choose **Print.** The **All** option is automatically selected. In the **Number of copies** box, enter the number of copies you want to print. By default number of copies is 1.
3. Click **OK**.

Debugger

The purpose of the mpC parallel debugger is to allow you to see what's going on inside a parallel program while it executes. With the mpC parallel debugger you can start your program, specifying anything that might affect its behavior, make your program stop on specified conditions, examine values of variables and expressions on all processes of the parallel program during its execution and solve many other problems you encounter while debugging a parallel program. The mpC parallel debugger provides an intuitive and easy-to-use graphical interface that allows you to keep track of your parallel program execution without switching between numerous windows. Since the mpC parallel debugger treats a parallel program as a whole all the information concerning the program is displayed in one window. Another feature supported by the mpC debugger is *remote* debugging. This feature allows you to debug programs running on remote machines over network, both local and Internet. Remote debugging is "natural" for the mpC debugger because of its architecture. The mpC debugger consists of two parts – client and server. The client part of the debugger runs on the machine, on which you are currently working, while the server part of the debugger must be started on the machine that runs the process of your parallel program having global rank 0.

In the following topics there are given overviews of the main features of the mpC parallel debugger, as well as overview of the debugging process:

- Cursors overview
- Moving commands overview
- Breakpoints overview
- Watchpoints overview
- Displays overview
- Debugging programs overview

Managing Cursors

When you debug an ordinary single-process program there is only one 'current' execution point. With parallel program the situation is much more difficult. Any parallel program consists of several, sometimes several hundreds, processes and each of these processes has its 'current' execution point. During debugging each process of a parallel program is associated with some cursor that corresponds to process' current execution point. The cursors that correspond to the same execution point are grouped into one compound cursor. Thus, we introduce two notions – *process cursor* and *compound cursor*. *Process cursor* is the 'current' execution point of the process. *Compound cursor* is a group of process cursors corresponding to the same line in the source code. Hereinafter we will call compound cursor "debug cursor" or just "cursor". When the debugged program breaks execution all currently existing debug cursors are displayed in the **Cursor** window (see Debug info window). Cursors can be either red or amber or green. Since a parallel program usually consists of many processes, in general there may be several cursors of each color. In the mpC debugger you continue execution of cursors. It means that when you move a cursor, you continue execution of all processes corresponding to this cursor. All debugger moving commands can be divided in two categories: "step" commands and "go" commands. **Step into**, **Step over** and **Step out** commands belong to the first category, while **Go** and **Go to cursor** commands belong to the second category. You can use step commands for processes corresponding to green cursors only. Go commands can be used for processes corresponding either to green cursors only or to both green and amber cursors. Using the debugger moving commands, you can't continue execution of the processes corresponding to red cursors (see Moving Commands). Green and amber cursors can be split into several new cursors or merged with other cursors of the same color and corresponding to the same position in the source code. Also green and amber cursors can be recolored. Thus, by making some cursor amber you can hold execution of the processes corresponding to this cursor and continue execution of other processes – these corresponding to green cursors - using the **Go** or **Go to cursor** command.

Cursors allow you to treat a parallel program as a whole, while other known parallel debugger treat a parallel program as a set of independent processes communicating with each other. You can divide the parallel program into several cursors and control execution of groups of processes.

This chapter explains how to use commands that allow you to manage cursors. The following topics are discussed in this chapter:

- Splitting/Merging cursors
- Changing cursor color
- Selecting network filter

Remember that you can't merge, split or change color of red cursors.

Splitting/Merging Cursors

When debugging a parallel program it's sometimes helpful to restart after some breakpoint a part of processes while holding execution of the rest of the processes. You can do it by making some of green cursors amber. If you want to continue execution of a part of processes corresponding to one green cursor you can split this cursor into two cursors and make one of them amber. To the amber cursor there will correspond the processes, execution of which you want to hold. Also you can merge green or amber cursors.

Remember that the cursors you want to merge must have the same color and process cursors of all processes corresponding the cursors must correspond to the same line of the source code.

To split a cursor

1. On the **Debug cursors** tab of the **Debug info** window right-click the cursor you want to split. The **Cursor manager** dialog appears with the **Split Cursors** tab active. If the **Debug cursors** tab isn't displayed click **View->Debug Windows->Debug cursors**
2. In the **Select cursor** combo box you can select a cursor to split.

3. Select the process or processes you want to add to new cursor from the left list and click >>. The process appears in the right list. If you change your mind about adding some process to the new cursor select this process from the right list and click <<.
4. Thus you have two new cursors. In the left and right lists there are presented processes constituting these cursors. For any cursor you can set color by choosing one from the **Select color** dropdown list.
5. Click **Split**.

To merge cursors

1. On the **Debug cursors** tab of the **Debug info** window right-click one of the cursors you want to merge. The **Cursor manager** dialog appears with the **Merge Cursors** tab active.
2. In the **Select cursor** combo box you can select one of cursors to merge.
3. Select the cursors to merge from the list. In the list there are presented all the cursors with the current position corresponding to the same line of source code.
4. For new cursor you can set color by selecting one in the **Select color** combo box.
5. Click **Merge.**

Changing Cursor Color

If you want to hold execution of processes belonging to some green cursor you can make this cursor amber. Or you can split a green cursor into two and make one of the resulting cursors amber. Also you can make an amber cursor green. During debugging you will need to change cursors color many times.

To change cursors color

- Right-click the cursor you want to recolor in the **Debug cursors** tab of the **Debug info** window. If you want to make the cursor amber choose from the **Make amber** submenu one of the following options: **Selected**, **All** and **All visible**. If you choose **Selected** then only the cursor you right-clicked becomes amber. If you choose **All** then all the green cursors become amber. And if you choose **All visible** then only these green cursors, which are currently displayed (see Setting Network Filter), become amber. To make cursors green choose from the **Make green** submenu one of the options described above.

  - or -

- Right-click any empty place on the pane in the **Debug cursors** tab. By choosing from **Make green** or **Make amber** submenus either **All** or **All visible** options you can change color of all or all visible cursors respectively.

Remember that you can’t change color of red cursors.

Setting Network Filter

If your parallel program consists of many processes, then at some breakpoint too much information may be displayed in the **Debug info** and **Watch** windows. To decrease the amount of displayed information you can set network filter for any of these windows. You can set independent network filters for any of the follwing windows: **Debug cursors** and **Call stack** tabs of the **Debug info** window, the **Watch** window, the **Quick watch** dialog and the **Break info** dialog. After setting network filter only the information concerning the processes specified by network filter is displayed. In fact, when you set network filter you choose not processes but virtual processors or entire networks. And for the processes corresponding to the virtual processors chosen in the network filter information is displayed. For example if you want to see values of watched variables only for the processes corresponding to the virtual processors of the network *nw* you can choose this network in the network filter and apply this filter to the **Watch** window.

In other words, by setting a network filter you specify a condition that is applied to global ranks of virtual processors (see mpC Tutorial). This condition is called *network filter condition.* The windows, to which this network filter is applied, display information for only these processes that correspond to the virtual processors satisfying the network filter condition.

Remember that in general any network is limited in scope to the block, in which this network is defined. Consequently, some network may be visible at one breakpoint and invisible or even non-existent at another breakpoint. So if you choose some network, say *nw1*, in the network filter at some breakpoint, say *1*, and the next breakpoint, say *2*, is out of scope of this network, then at the breakpoint *2* no information is displayed in the windows, to which this network filter is applied. But the previous statement isn't absolutely correct because in general not all processes come to the execution point corresponding to the line of code containing the breakpoint *2* (see Cursors Overview). The following statement is more correct: for these processes, current execution position of which is within the scope of *nw1*, information is displayed, while for the processes that have reached the point out of scope of *nw1* information isn't displayed.

Another detail to discuss is that when you set a network filter you can choose virtual processors as either members of some network or just virtual processors. The pane in the **Net filter** dialog displays hierarchical list of currently existing networks and virtual processors. You can expand any of networks or the **Processes** by clicking the (+) box, which opens into a tree that may contain additional boxes. If you expand **Processes** the list of processes comprising the parallel program is displayed. If you choose processes from this list then the filter is valid till the end of program execution. If you choose network or subnetwork the filter is valid as long as this network or subnetwork exists.

To set network filter

1. Right-click in one of the windows for which network filter can be set, and choose **Select filter** from the context menu. The **Net filter** dialog appears.
2. Select networks and/or virtual processors. By checking the checkboxes at the bottom of the dialog you can select windows to which this filter will be applied.
3. Click **Select**.

To clear network filter

- Right-click in one of the windows for which network filter can be set, and choose **Clear filter** from the context menu.

Moving Commands

Moving commands are **Go**, **Go to cursor**, **Step Into**, **Step Over** and **Step Out**. Each of these command execute processes belonging to either green or amber cursors. The commands **Go** and **Go to cursor** can be used for processes belonging to both amber and green cursors, while the step commands can be used for processes belonging to green cursors only. To start debugging, click **Start debug** on the **Debug** menu and then click the **Go**, **Step Into** or **Go To Cursor**.

This chapter describes the debugger moving commands and their actions. The following commands are discussed in this chapter:

- Go command
- Go to cursor command
- Step into command
- Step over command
- Step out command

Go Command

Executes code from the current statement until a breakpoint or the end of the process is reached, or until the application pauses for user input. From the **Go** submenu of the **Debug** menu you can choose either **Green** or **All** to apply the command to processes belonging to either green or both green and amber cursors respectively. After you click **Go** every process, to which the command is applied, is executed from its current statement. To make only a part of processes move on the **Go** command split green cursors if necessary, make the appropriate cursors amber and choose **Green** from the **Go** submenu of the **Debug** menu.

Go to cursor Command

Executes the program as far as the line that contains the insertion point. Like the **Go** command this command can be applied to the processes belonging to as green as amber cursors. When your program stops in break mode, some green cursors may correspond to the line, which is after the line containing the insertion point. In that case you can either make these green cursors amber and choose **Green** from the **Go to cursor** submenu or just click **Go to cursor->Green** thus making the processes that have already passed the insertion point run until either exit or barrier point has reached.

Step into Command

Single-steps through instructions in processes of the parallel program, and enters each function call that is encountered. If the next statement is a function call, the debugger steps into that function, then pauses execution at the beginning of the function. Can be applied only to the processes that belong to green cursors. In general there may be several green cursors and processes belonging to different cursors may have different next instructions. So when you command **Step into**, the processes belonging to different cursors may single-step through different instructions. Remember that you can always make a part of green cursors amber.

Step over Command

Single-steps through instructions in processes of the parallel program. If this command is used when a process reaches a function call, the debugger executes this function, but pauses after the function returns. This command can be applied only to the processes, which belong to green cursors. In general there may be several green cursors and processes belonging to different cursors may have different next instructions. So when you command **Step over**, the processes belonging to different cursors may step through different instructions. Remember that you can always make a part of green cursors amber.

Step out Command

Executes processes out of a function call, and stops on the instruction immediately following the call to the function. Using this command, you can quickly finish executing the current function after determining that a bug is not present in the function. In general there may be more than one green cursor and processors belonging to different cursors may be stopped inside different functions. So when you command **Step out**, the processes belonging to different cursors may execute out of different function calls. Remember that you can always make a part of green cursors amber.

Managing Breakpoints

A "breakpoint" makes your program stop whenever a certain point in the program is reached. Setting a breakpoint you specify the place where your program should stop. For each breakpoint, you can add conditions to control in finer detail whether your program stops.

When you debug usual one-process program with a conventional debugger there is only one ‘current’ execution position and if in this position your program should stop because it has reached a breakpoint then it stops provided all the breakpoint conditions are satisfied. With a parallel program the situation is different. Any parallel program consists of more than one process. And each of processes has its own ‘current’ execution position. So in general not all processes reach a place specified by a breakpoint. And what is more a breakpoint can be set as for all the processes of a parallel program as for a group of processes or even for one process only. The processes, for which a breakpoint is set, can be specified by a conditional expression. Conditional expression of a breakpoint can be any syntactically correct mpC Boolean expression but it mustn’t contain function calls, communications and network type conversions, as well as it mustn’t cause side effects. For example if an mpC program contains the following text:

```
…
net SimpleNet (N) w;
int my_i_coord;
my_i_coord = I coordof w;
…
```

and conditional expression of a breakpoint is

```
my_I_coord > 0 &&  my_I_coord < 3
```

then only these processes, which correspond to virtual processors that have 1 and 2 I coordinates within the w network, stop at this breakpoint.

Let us consider the situation when some process of a parallel program has stopped at a breakpoint. The other processes can be in one of the following states: continue execution, be stopped at the same breakpoint, be stopped at some other breakpoint, be waiting at a synchronization point and finish execution. *Synchronization* point is a point at which either a process communicates with other processes or execution is held until all the processes of some group have reached this point. A parallel program stops only if all of its processes stop. At the moment when a parallel program stops the process cursor associated with some process can be either green or red. If the process can be restarted with one of moving commands, for example **Go** (see Moving Commands), then the process cursor associated with this process is green. Otherwise the process cursor is red. Process cursor can be red if the process is stopped at a synchronization point or if it has finished execution. The processes are combined into *compound cursors* according to the colors of their cursors and to the lines of source code corresponding to the point at which processes are stopped. That is if some two processes having green cursors stop at the points corresponding to the same line of source code then these two processes belong to one compound cursor.

This chapter explains how to manage breakpoints. The following topics are covered:

- Setting breakpoints
- Removing breakpoints
- Disabling/Enabling breakpoints
- Changing breakpoint condition
- Setting "ignore count" of a breakpoint

Setting Breakpoints

To control the execution of your application, you set *breakpoints*, which are places where the code should pause and call the debugger. You can set a breakpoint at design time or at run time. Remember that in general not all processes reach the breakpoint – some of them may stop at communication points or just pass the line containing breakpoint.

NOTE: You can set a breakpoint at run time only when all processes are stopped.

To set a breakpoint

In the **Text Editor** window, move the insertion point to the line of code where you want to set a breakpoint and right-click in the selection margin.

From the context menu choose:

- **Add breakpoint**. The **Breakpoint** dialog appears.

  In the dialog:

  1. Select or enter a filename you want to set a breakpoint in.
  2. Enter number of the line where you want to set a breakpoint. By default there will be number of the current line.
  3. Enter conditional expression. Each time when your program reaches a breakpoint the expression is evaluated, and your program stops only if the condition is TRUE. (See also Changing breakpoint condition). By default the field is empty. In this context an empty condition expression is always TRUE.
  4. Enter "ignore count" of a breakpoint. "ignore count" is number of times to skip a breakpoint before stopping. For example, if the ignore count value is N, the breakpoint does not stop the next N times your program reaches it. By default the ignore count value is 0. (See also Setting "ignore count" of a breakpoint)

     - or –

- **Debug Object manager**. The **Debug Object manager** dialog appears. In the **Breakpoints** tab click **Add**. The **Breakpoint** dialog appears. Follow the instruction from the previous item. Another way to display the Debug Object manager dialog is to click **Edit -> Breakpoints.**

When you set a breakpoint, a dot is displayed in the selection margin of the Text Editor window next to the line containing the breakpoint. The dot is red if the breakpoint is enabled and gray if disabled.

Removing Breakpoints

To remove an existing breakpoint

- Right-click in the selection margin and choose **Debug Object manager** from the context menu. In the Debug Object manager dialog on the **Breakpoints** tab select the breakpoint you want to remove and click the **Remove** button.

  - or –

- Click **Edit -> Breakpoints** and follow the instructions from the previous item.

When you remove a breakpoint the dot in the selection margin of the **Text Editor** next to the line containing the breakpoint disappears.

Disabling/Enabling Breakpoints

Rather than deleting a breakpoint you might prefer to "disable" it. This makes the breakpoint inoperative as if it had been deleted, but remembers the information on the breakpoint so that you can "enable" it again later.

To disable/enable an existing breakpoint

- In the Debug Object manager dialog on the **Breakpoints** tab clear/check the breakpoint you want to disable/enable

  - or -

- In the Debug Object manager dialog on the **Breakpoints** tab select the breakpoint you want to modify and click the **Modify** button. The breakpoint dialog appears. In the breakpoint dialog check/clear **Disable the breakpoint**.

  - or -

- Double-click the dot in the selection margin next to the line containing the breakpoint you want to disable/enable. If the color of the dot was red, i.e. the breakpoint was enabled, it becomes gray, i.e. the breakpoint becomes disabled. And vice versa, if the dot was gray it becomes red after double-clicking.

Changing Breakpoint Condition

The simplest sort of breakpoint breaks every time your program reaches a specified place. You can also specify a "condition" for a breakpoint. A condition is just a Boolean expression in the mpC programming language. A conditional expression mustn’t contain communications, network type conversions and function calls. A breakpoint with a condition evaluates the expression each time your program reaches it, and your program stops only if the condition is TRUE. By default a breakpoint has no conditional expression.

To specify a condition expression for a breakpoint

1. From the **Edit** menu, choose **Breakpoints**.
2. In the Debug Object manager dialog on the **Breakpoints** tab select the breakpoint you want to modify and click the **Modify** button.
3. In the breakpoint dialog in the **Condition** text box, type an expression, such as `x==3`, that evaluates to true or false, or change the existing expression.

Setting "ignore count" of a Breakpoint

A special case of a breakpoint condition (see Changing breakpoint condition) is to stop only when the breakpoint has been reached a certain number of times. This is so useful that there is a special way to do it, using the "ignore count" of the breakpoint. Every breakpoint has an ignore count, which is an integer. Most of the time, the ignore count is zero, and therefore has no effect. But if your program reaches a breakpoint whose ignore count is positive, then instead of stopping, it just decrements the ignore count by one and continues. As a result, if the ignore count value is N, the breakpoint does not stop the next N times your program reaches it.

To set an ignore count value of a breakpoint

1. From the **Edit** menu, choose **Breakpoints**.
2. In the Debug Object manager dialog on the **Breakpoints** tab select the breakpoint you want to modify and click the **Modify** button.
3. In the breakpoint dialog in the **Ignore count** text box, type the ignore count value. It must be integer. By default ignore count of a breakpoint is 0.

Managing Watchpoints

A "watchpoint" is a special breakpoint that stops your program when the value of an expression changes. You must use a different command to set watchpoints, but aside from that, you can manage a watchpoint like any other breakpoint: you enable, disable, and delete it. You can use a watchpoint to stop execution whenever the value if an expression changes, without having to predict a particular place where this may happen. Just like breakpoint a watchpoint may have its conditional expression and its "ignore count". Watchpoint condition must meet the same requirements as breakpoint condition. Watchpoint can be set for all processes of a parallel program or for a group of processors only (see also Breakpoints overview). You can add a watchpoint either at design time or when your program is stopped at breakpoint (in break mode).

By setting a watchpoint you specify an expression to be evaluated. The watchpoint expression can be any syntactically correct mpC expression, but it mustn’t contain function calls, communications and network type conversions, as well as it mustn’t cause side effects. The expression is evaluated, and subsequently, each time the value of the expression changes, the program stops, and the new value is displayed in the **Watch** window. If the watched expression contains local variables you should set the watchpoint within the scope of these variables, i.e. for example within the function in which the local variables are defined. Otherwise the watchpoint will be ignored. Note that if you want value of a variable defined within some block to be evaluated and there is a variable with the same identifier defined in a block containing this block then the value of the latter variable will be evaluated independently of the location where you set the watchpoint. That is if an mpC program contains the following code

```
/*1*/int [*]f ()
/*2*/{
/*3*/  int i;
         …
/*4*/  {
/*5*/    int i;
/*6*/    i ++;
/*7*/  }
/*8*/}
```

and you set a watchpoint evaluating the value of i at the line 6, the value of the variable i defined within the function f will be evaluated.

This chapter explains how to manage watchpoints. The following topics are discussed:

- Setting watchpoints
- Deleting watchpoints
- Disabling/Enabling watchpoints
- Changing watchpoint condition
- Setting "ignore count" of a watchpoint

Setting Watchpoints

You can use a watchpoint to stop execution whenever the value of an expression changes, without having to predict a particular place where this may happen. The expression can be a variable, a function call, or any other valid mpC expression.

To set a watchpoint at design time or in break mode

In the **Text Editor** window, move the insertion point to the line of code where you want to set a watchpoint and right-click in the selection margin.

From the context menu choose:

- **Add watchpoint**. The **Watchpoint** dialog appears.

  In the **Watchpoint** dialog:

  1. Enter a watchpoint expression in the **Expression** text box.
  2. Select or enter a filename you want to set a watchpoint in.
  3. Enter number of the line where you want to set a watchpoint. By default there will be number of the current line.
  4. Enter conditional expression. Each time when value of a watchpoint expression changes the conditional expression is evaluated, and your program stops only if the condition is TRUE. (See also Changing watchpoint condition). By default the field is empty. In this context an empty condition expression is always TRUE.
  5. Enter "ignore count" of a watchpoint. "ignore count" is number of times to skip a watchpoint before stopping. For example, if the ignore count value is N, the watchpoint does not stop the next N times the watchpoint expression changes. By default the ignore count value is 0. (See also Setting "ignore count" of a watchpoint)

  - or –

- **Debug Object manager**. The **Debug Object manager** dialog appears. In the **Watchpoint** tab click **Add**. The **Watchpoint** dialog appears. Follow the instruction from the previous item. Another way to display the **Debug Object manager** dialog is to click **Edit** -> **Watchpoints.**

When you set a watchpoint, an oval is displayed in the selection margin of the **Text Editor** window next to the line containing the watchpoint. The oval is cyan if the watchpoint is enabled and gray if disabled.

Removing Watchpoints

It is often necessary to eliminate a watchpoint once it has done its job and you no longer want your program to stop there. This is called "deleting" the watchpoint. A watchpoint that has been deleted no longer exists; it is forgotten.

To remove an existing watchpoint

- Right-click in the selection margin and choose **Debug Object manager** from the context menu. In the **Debug Object manager** dialog on the **Watchpoints** tab select the watchpoint you want to remove and click the **Remove** button.

  - or –

- Click **Edit -> Watchpoints** and follow the instructions from the previous item.

When you remove a watchpoint the oval in the selection margin of the **Text Editor** next to the line containing the watchpoint disappears.

Disabling/Enabling Watchpoints

Rather than deleting a watchpoint you might prefer to "disable" it. This makes the watchpoint inoperative as if it had been deleted, but remembers the information on the watchpoint so that you can "enable" it again later.

To disable/enable an existing watchpoint

- In the **Debug Object manager** dialog on the **Watchpoints** tab clear/check the watchpoint you want to disable/enable

  - or -

- In the **Debug Object manager** dialog on the **Watchpoints** tab select the watchpoint you want to modify and click the **Modify** button. The **Watchpoint** dialog appears. In the **Watchpoint** dialog check/clear the **Disable watchpoint** checkbox.

  - or -

- Double-click the oval in the selection margin next to the line containing the watchpoint you want to disable/enable. If the color of the oval was cyan, i.e. the watchpoint was enabled, it becomes gray, i.e. the watchpoint becomes disabled. And vice versa, if the oval was gray it becomes cyan after double-clicking.

Changing Watchpoint Condition

The simplest sort of watchpoint breaks every time when value of the watchpoint expression changes. You can also specify a "condition" for a watchpoint. A condition is just a Boolean expression in the mpC programming language. A conditional expression mustn’t contain communications, network type conversions and function calls. A watchpoint with a condition evaluates the conditional expression each time value of the watchpoint expression changes, and your program stops only if the condition is TRUE. By default a watchpoint has no conditional expression.

To specify a condition expression for a watchpoint

1. From the **Edit** menu, click **Watchpoints**.
2. In the **Debug Object manager** dialog on the **Watchpoints** tab select the watchpoint you want to modify and click the **Modify** button.
3. In the **Watchpoint** dialog in the **Condition** text box, type an expression, such as `x==3`, that evaluates to true or false, or change the existing expression.

Setting "ignore count" of a Watchpoint

A special case of a watchpoint condition (see Changing watchpoint condition) is to stop only when value of the watchpoint expression has changed a certain number of times. This is so useful that there is a special way to do it, using the "ignore count" of the watchpoint. Every watchpoint has an ignore count, which is an integer. Most of the time, the ignore count is zero, and therefore has no effect. If the ignore count value is N, the watchpoint does not stop the next N times when value of the watchpoint expression changes.

To set an ignore count value of a watchpoint

1. From the **Edit** menu, choose **Watchpoints**.
2. In the **Debug Object manager** dialog on the **Watchpoints** tab select the watchpoint you want to modify and click the **Modify** button.
3. In the **Watchpoint** dialog in the **Ignore count** text box, type the ignore count value. It must be an integer. By default ignore count of a watchpoint is 0.

Managing Displays

Display is a variable or expression, the value of which in every process is watched during program execution in debug mode. The expression you specify by setting a display is added to the list of watched expressions in the **Watch** window. List of the set displays you can see in the **Expressions** tab of the **Debug objects** dialog. If you are particularly interested how value of some variable changes during execution but don’t want the program to stop when the value changes you can add this variable to the list of watched variables by setting display. Display can be any syntactically correct mpC expression but it mustn’t contain function calls, communications and network type conversions, as well as it mustn’t cause side effects.

Like a breakpoint or watchpoint, a display is set at some line in the current source file. If the specified expression contains local variables you should set the display within the scope of these variables, i.e. for example within the function in which the local variables are defined. Otherwise the display will be ignored. Note that if you want to know value of a variable defined within some block and there is a variable with the same identifier defined in a block containing this block then the value of the latter variable will be evaluated independently of the location where you set the display.

If some process doesn’t reach scope of some variable from the list of watched expressions when the whole program stops, then in the cell in the **Watch** window, which displays value of this variable on this processor, error message is displayed.

This chapter explains how to manage displays. The following topics are discussed:

- Setting displays
- Modifying displays
- Removing displays

Modifying Displays

To modify a display

1. Right-click in the selection margin of the **Text Editor** window and choose **Debug Object manager** from the context menu.

   - or -

   Choose **Expressions** from the **Edit** menu.
2. In the **Expression** tab of the **Debug Object manager** dialog select the expression you want to modify. Click **Modify**. The **Expression** dialog appears.
3. If necessary, modify the expression in the **Expression** text box.
4. If necessary, change name of the file, in which expression is located. The dropdown list box contains the most recently used filenames.
5. If necessary, change the line number.
6. Click **OK**.

Removing Displays

To remove a display

1. Right-click in the selection margin of the **Text Editor** window and choose **Debug Object manager** from the context menu.

   - or -

   Choose **Expressions** from the **Edit** menu.
2. In the **Expression** tab of the **Debug Object manager** dialog select the expression you want to remove. Click **Remove**.
3. Click **Close**.

When you remove a display the rhomb in the selection margin of the **Text Editor** next to the line containing the display disappears.

Setting Displays

To set a display

In the **Text Editor** window, move the insertion point to the line of code where you want to set a display and right-click in the selection margin.

From the context menu choose:

- **Add expression**. The **Expression** dialog appears.

  In the **Expression** dialog:

  1. Type or paste the expression to be watched in the **Expression** text box.
  2. If necessary, change name of the file, in which expression is located. By default the text box contains the name of the current source file. The dropdown list box contains the most recently used filenames.
  3. If necessary, change the line number. By default the text box contains the number of the line containing insertion point.
  4. Click **OK** in the **Expression** dialog and then in the **Debug Object manager** dialog.

  - or -

- **Debug Object manager**. The **Debug Object manager** dialog appears. In the **Expression** tab click **Add**. The **Expression** dialog appears. Follow the instructions from the previous item.

Also you can choose **Expressions** from the **Edit** menu. The **Debug Object manager** dialog appears with the **Expressions** tab active.

When you set a display, a magenta rhomb is displayed in the selection margin of the **Text Editor** window next to the line containing the display.

Debugging programs

You have created your application and resolved the build errors. Now it's time to find and correct those logic errors that keep your application or stored procedures from running correctly. You can do this with the mpC Workshop’s integrated mpC debugger. The mpC debugger allows you to test your mpC programs. You can manage and set breakpoints and watchpoints, as well as view values of variables.
The mpC programming language provides facilities to manage a kind of resource named computing space. It is a set of virtual processors of different performances interconnected with links of different bandwidth. mpC Programmer can manage computing space by allocating and discarding regions of computing space named networks. Managing computing space in mpC is similar to managing memory in C. At run-time virtual processors correspond to processes of parallel program. When debugging a parallel program, you can view information about all available virtual processors in the program. Using Parallel Debugger, you can start and stop individual groups of virtual processors (see Cursors overview).
The mpC Parallel Debugger provides information about the *nets* and *subnets,* which exist in the debugged program. So you don’t have to group virtual processors by hand to achieve the grouping of virtual processors that is logically correct for the debugged program. The mpC Parallel Debugger groups processes automatically.
The mpC Parallel Debugger has client-server architecture. Such architecture allows the debugger to support remote debugging. Since client and server communicate with each other over TCP/IP protocol, you can debug programs running on remote machines in New York from your workplace in London.

This chapter explains how to debug mpC parallel programs using the mpC parallel debugger. This chapter discusses:

- Preparing programs for debugging
- Debugger windows
- Debugger dialog boxes
- Using spreadsheet fields
- Viewing the value of a variable
- Using Break Info window
- Managing breakpoints
- Managing watchpoints
- Managing displays
- Running to a location
- Stepping into functions
- Stepping over or out of a function
- Viewing call stacks
- Working with current cursor
- Working with mpC daemon

Preparing Programs for Debugging

When you create a new distributed project, both a **Debug** and a **Release** target are automatically created with default options set for each. Default settings for **Debug** target define that debug information is generated for each build in this target. For any build you can define whether debug information is generated for output file of this build or not. The mpC parallel debugger works with debug information generated by the mpC compiler only (see mpC Workshop architecture). So for the C files generated by the mpC compiler you can switch on any optimization you need. You can change settings for C compiler in the **C** tab of the **Project settings** dialog (see Project Settings dialog options).

To generate debug information for a build

1. Click **Settings** on the **Project** menu or right-click a build of the current target in the **Workspace** window and choose **Settings** from the context menu. The **Project Settings** dialog appears.
2. In the **Project Settings** dialog on the **Settings For** pane select the build, for which you want to modify settings. On the **MPC** tab choose **General** from the list at the top.
3. Check the **Generate debug info** check box.
4. Click **OK**.

Debugger Windows

Several specialized windows display debugging information for your program. When you are debugging, you can access these windows using the **Debug Windows** list on the **View** menu.

The following table lists the debugger windows and describes the information they display.

Output	Displays information about the build process, including any compiler, linker, or build-tool errors.
Watch	Displays names and values of variables and expressions for each process specified in the current network filter.
Debug info	Displays the current debug cursors (in the **Cursors** tab), currently available networks (in the **Nets** tab) and call stacks for each process specified in the current network filter (in the **Call stacks** tab).
Current cursor	Displays current debug cursor.
Debug Info	Displays breakpoints and watchpoints, which caused processes of the parallel program to stop. The dialog appears when the program stops in break mode.

Debugger Dialog Boxes

In addition to windows, the debugger uses two dialog boxes to manipulate breakpoints, watchpoints and variables. You can access the **Debug object manager** dialog box using either the **Breakpoints** or the **Watchpoints** or the **Expressions** command on the **Edit** menu. You can access the **Quick Watch** dialog box using the **Quick Watch** command from the **Debug** menu.

Debug object manager	Displays a list of all breakpoints (in the **Breakpoints** tab), a list of all watchpoints (in the **Watchpoints** tab) and a list of all watched expressions (in the **Expressions** tab) assigned to your project.
Quick Watch	Displays a variable or expression. Use **Quick Watch** to quickly view a variable or expression or to add it to the Watch window.

Using Spreadsheet Fields

The debugger interface uses spreadsheet fields, which have an interface similar to that of Microsoft Excel. These spreadsheet fields appear in the Watch window, the Variables window, and the **Quick Watch** dialog box.

Spreadsheet fields contain controls for easy viewing of array, object, structure, and pointer variables. If the variable is a pointer, the branch immediately below the pointer contains the value pointed to. If the variable is an array, object, or structure, the branch below the variable contains the component elements or members.

The variables are marked with a box containing a plus sign (+) in the Name column. You can expand the variable by clicking the + box, which opens into a tree that may contain additional boxes. When a variable is expanded, the box in the Name column contains a minus sign (–). You can collapse an expanded variable by clicking the – box. As an alternative, you can expand a variable by selecting it and pressing the PLUS SIGN or RIGHT ARROW key. You can collapse a variable by selecting it and pressing the MINUS SIGN or LEFT ARROW key.

Scalar variables, which have no components to expand, do not have boxes in the Name column.

When working with these fields, you can autosize a column to fit its contents by double-clicking the divider. You can size a column manually by dragging the divider at the right edge of the column.

In the **Watch** window the columns that are next to the **Name** column display values of variables on the processes specified in the current network filter, one column for one process. If you change current call stack position of a process – you can do this in the **Call stack** tab of the **Debug info** window – values of all variables in the corresponding column of the **Watch** window are colored in red.

Viewing the Value of a Variable

To view the value of a variable or expression

1. Wait for the debugger to stop at a breakpoint or watchpoint.
2. On the **Debug** menu, click **Quick Watch**.
3. Type or paste the variable or an expression into the **Expression** text box, and click **Recalculate**. The **Expression values** list displays name of the specified variable or expression and its values for the processes specified in the network filter of the **Watch** window. If you want to add this variable or expression to the **Watch** window, click **Add Watch**.
4. Click **Close**.

To view the value of a variable or expression in the Watch window

1. Wait for the debugger to stop at a breakpoint or watchpoint.
2. On the **View** menu, click **Debug Windows**, then click **Watch**.

3. Type or paste the variable or an expression into the **Name** column on the pane of the **Watch** window. Press ENTER (of typing). The **Watch** window evaluates the variable or expression immediately and displays the value or an error message. The window is only updated when the debugger stops at a breakpoint or watchpoint. Along with the typed variables the Watch window displays the variables specified in the Expressions tab of the **Debug object manager** dialog.

If you add an array or object variable to the **Watch** window, plus sign (+) or minus sign (–) boxes appear in the Name column. Use these boxes to expand or collapse your view of the variable described in Spreadsheet Fields.

By default the **Watch** window displays values for all processes of the debugged program. You can set a network filter for this window and watch values only on the processes corresponding to the virtual processors specified in the network filter (see Setting network filter).

If you change the current call stack position for a process that satisfies the network filter condition for the **Watch** window, then all values in the column displaying values for this process are colored in red.

Using Break Info window

When execution of your program stops in break mode the **Break Info** window appears. The **Break Info** window displays list of the breakpoints and watchpoints, which caused processes of the program to stop.

To view line of source code containing some breakpoint or watchpoint select this breakpoint or watchpoint and click **Go to**. If the file containing the line of source is already open the window displaying this file receives focus and text cursor is positioned before the first character in the line.

To view more detailed information about some breakpoint or watchpoint select it from the list and click **Details**. The **Details** dialog has two fields - watch field and cursors pane. The watch field displays either values of the watchpoint expression, if the dialog displays details of a watchpoint, or the string "Undefined", if the dialog displays details of a breakpoint. The cursors pane displays the processes, which stopped at this breakpoint or watchpoint, and the cursors, to which these processes belong. If some process stopped at the breakpoint or watchpoint doesn't satisfy the current **Watch** window network filter condition, then the spreadsheet field doesn't display information for this process. However, if no process of these stopped at the breakpoint or watchpoint satisfy the current **Watch** window network filter condition then the watch field displays information for all of these processes. The processes displayed in the cursors pane satisfy the similar rule but instead of **Watch** window network filter the **Debug cursors** window network filter is used. For more information about network filters consult Setting Network Filter. For a watchpoint the watch field displays values of watchpoint expression on the processes, which stopped at this watchpoint.

NOTE: If only one unconditional breakpoint stops the program execution, the **Break Info** window doesn't appear.

Managing Breakpoints

You can set breakpoints to specify the places where your program should stop. Parallel program stops only if all processes are stopped. Imagine the situation that not all processes have reached some breakpoint – the breakpoint may be set in the body of some condition statement. What happens with the rest of processes? Some of them may run until they finish execution, other may reach other breakpoint or watchpoint, or wait at communication or barrier point. A parallel program stops only if all of its processes stop for one or another reason. To make sure that all processes will stop at breakpoint you should specify the place, which can be reached by all processes.

For more detailed information on breakpoints consult the topic Breakpoints overview.

Managing Watchpoints

Just like a breakpoint, watchpoint stops processes of a parallel program. But watchpoint stops a process when the value of the watchpoint expression changes in this process. Consequently, different processes may be stopped in different points by the same watchpoint.

For more detailed information on watchpoints consult the topic Watchpoints overview.

Managing Displays

Display is a variable or expression that can be specified in the **Expressions** tab of the **Debug object manager**. You can watch all values of the expressions in the **Watch** window. The values are updated only when the program stops at a breakpoint or watchpoint. Any display can be set or deleted.

For more detailed information on displays consult the topic Displays overview.

Running to a Location

In general not all processes of a parallel program reach a certain breakpoint. When parallel program stops every process belong either to red or to green cursor (see Cursors overview). You can't manipulate processes belonging to red cursors. You can't command them **Go** or **Go to cursor**. But you can manipulate processes belonging to green or amber cursors. You can split a green cursor into several new ones and some of the new cursors make amber. The commands **Go** and **Go to cursor** can be applied either to green cursors only or to both green and amber cursors.

To run until a breakpoint is reached

1. Set a breakpoint.
2. On the **Debug** menu, click **Start Debug.**

3. From the **Debug** menu select **Go** and then either **Green** or **All** – when you start debugging the result of the two commands is the same because all processes belong to one green cursor corresponding to the first expression of the main function.

To run to the cursor (while the debugger is not running)

1. In a source file, move the insertion point to the location where you want the debugger to break.
2. On the **Debug** menu, click **Start Debug.**
3. From the **Debug** menu select **Go To Cursor** and then either **Green** or **All**.

To run to the cursor (while the debugger is running but halted)

1. In a source file, move the insertion point to the location where you want the debugger to break.
2. Split green cursors and recolor some of the new cursors if necessary.
3. From the **Debug** menu select **Go To Cursor**, and then select either **Green** if you want only processes belonging to green cursors to run to the specified location or **All** if you want the processes belonging to both green and amber cursors to run to the specified location.

Stepping into Functions

Only the processes belonging to green cursors can step into functions. Remember that while your program is stopped at a breakpoint or watchpoint you can split or merge green and amber cursors.

To run the program and execute the next statement (Step Into)

1. While the program is paused in break mode, click **Step Into** from the **Debug** menu. The debugger executes the next statement in all processes belonging to green cursors, then pauses execution in break mode. *Note that in general there may be more than one green cursor and processors belonging to different cursors may have different "next statements"*. If the next statement is a function call, the debugger steps into that function, then pauses execution at the beginning of the function.
2. Repeat step 1 to continue executing the program one statement at a time.

If you step into a nested function call, the debugger steps into the most deeply nested function. For example, on the line of code fun1(fun2()) the debugger steps into the function fun2, then pauses.

Stepping over or out of a Function

Only the processes belonging to green cursors can step over or out of functions. Remember that while your program is stopped at a breakpoint or watchpoint you can split or merge green and amber cursors.

To step over a function

1. While the program is paused in break mode, click **Step Over** from the **Debug** menu. The debugger executes the next function on the processes belonging to green cursors, but pauses after the function returns. *Note that in general there may be more than one green cursor and processors belonging to different cursors may have different "next function".*
2. Repeat step 1 to continue executing the program, one statement at a time.

To step out of a function

While the program is paused in break mode in some point inside the function, click **Step Out** from the **Debug** menu. The debugger continues until it has completed execution of the return from the function, then pauses. *Note that in general there may be more than one green cursor and processors belonging to different cursors may be stopped inside different functions.*

Viewing Call Stacks

Usually a parallel program consists of more than one processes and each process has its call stack. The mpC debugger allows you to view call stack of each process as well as to change current call stack position. You can choose a process from the dropdown list at the top of the **Call stack** tab of the **Debug info** window and you see the call stack for this process on the pane. You can apply network filter to the **Call stack** tab. When you change current call stack position for some process, all values for this process displayed in the **Watch** window are colored in red (see Setting Network Filter).

To view the call stack for a process

1. While the program is paused in break mode, on the **View** menu, click **Debug Windows**, then click **Call Stack.**
2. Choose a process from the dropdown list at the top of **Call stack** tab. On the pane you can see functions names, source files for each function and the line corresponding to the call point.

Working with current cursor

Current cursor is the cursor, current position of which is displayed in the **Text Editor** window each time when the program stops in break mode. There can be only one current cursor. The current cursor is displayed in the **Current cursor** window. You can make any green or amber cursor current by right-clicking this cursor in the **Debug cursors** tab of the **Debug info** window and choosing **Make current** from the context menu. You can also split the current cursor or merge it with other cursors. If you merge the current cursor with other cursors the resulting cursor becomes current. If you split the current cursor into several ones then the cursor having the same identifier as the initial current cursor before splitting becomes current (see Splitting/Merging Cursors).

Working with mpC daemon

mpC daemon is a program, which executes commands received from the client part of mpC Workshop. When you debug a program using the mpC parallel debugger the client part of mpC Workshop passes all commands to the mpC daemon and the daemon performs these commands. To start debugging your mpC Workshop client must be connected to server, which is mpC daemon. You can start the mpC daemon either locally, on your machine, or on a remote machine. In the former case the mpC daemon is started with *mpcd* command from the command line. If you want to debug your program on some remote computer first make sure that the mpC daemon is started on this computer. If the daemon isn't started on the remote machine you must open telnet session on this machine and start the daemon with *mpcd* command. Remember that the directory containing mpC daemon executable must be included in the %PATH% environment variable on the computer. If the debugged program hangs for some reason you should stop the mpC daemon with Ctrl+C, execute the following command line: *mpckill executable_name*, start the daemon again and connect to server by choosing **Connect to server** from the **Build** menu (see Connecting to Server).

Glossary

Break mode is a state of parallel program in which all the processes are stopped and some of them have reached a breakpoint or a watchpoint.

Build is settings entity. A build defines settings for one output file.

Builder is a computer, VPM node, on which one or more binary files are built.

Compound cursor is a group of process cursors corresponding to the same line of source code.

Computing space is set of virtual processors of different performances connected with links of different communication speeds accessible to the user for management. At runtime computing space maps to the set of processes of parallel program.

Current cursor is the cursor, current position of which is displayed in the Text Editor window each time when the program stops in break mode.

Cursor (debug cursor) can be either compound cursor or process cursor depending on context.

Cursor identifier is a number assigned to cursor that equals the minimum global rank among global ranks of virtual processors comprising this cursor. Since a virtual processor can’t belong to more than one cursor, there can’t be two different cursors with the same identifier at the same time.

Display is expression or variable, the value of which is evaluated during program execution. Execution isn’t stopped when value is changed. Value is just updated every time it changes.

Docking window is a window that has two display modes: floating or docked. In floating mode, a window has a thin title bar and can appear anywhere on your screen. A floating window is always on top of all other windows. In docked mode, a window is fixed to a dock along any of the four borders of the main application window.

Global rank of a virtual processor is a number, which is assigned to every virtual processor. Different virtual processors have different global ranks. Global rank of the host-processor is zero.

Host-processor is the virtual processor, which is defined from the beginning till the end of the mpC program execution. This processor is associated with the standard input stream.

mpC daemon is a utility that receives commands from the client part of mpC Workshop and initiates the execution of these commands.

Network is set of virtual processors, which is a region of computing space, connected with links of different communication speeds.

Network filter condition is condition that is applied to global ranks of virtual processors. The windows, to which this network filter is applied, display information for only these processes that correspond to the virtual processors satisfying the network filter condition.

Node is a computer, VPM node, which executes one or more processes of a parallel program. An executable file can’t be built on node – only executed.

Process cursor is the ‘current’ execution point of the process.

Subnetwork (of a region of computing space) is set of virtual processors, which is a subset of region of computing space.

Synchronization point is a point at which either a process communicates with other processes of a parallel program or execution is held until all the processes of some group have reached this point.

Target is a collection of builds, build entity that can be built in one or more binary files. Some of the output binary files may be built for different platforms.

Target path is the directory containing all files that are necessary for building the output file for this build. The target directory is contained in the directory specified by the environment variable %MPCLOAD% on each builder of the current VPM.

VPM (Virtual Parallel Machine) is a set of computers on which mpC application is executed.

Virtual Processor is a node of computing space. At runtime each virtual processor maps to one process of parallel program. So the number of virtual processors equals the number of processes of parallel program.

Index

A

add build 15
add file 16
add target 15
add VPM 11
adviser 5
Aliasing 24
arguments of function or network types 22
Assume Aliasing Across Function Calls 24
Assume No Aliasing 21

B

break mode 57
breakpoint 39, 40, 41, 53
breakpoint condition 40
breakpoint dialog 40, 42
build 13, 57
build file 15
builder 11, 13, 57
Building the current target 18

C

call stack 55
call stack tab 55
change build settings 20
change project setings 19
change target settings 20
changing file setting 20
client-server architecture 48
code pane 8
compiling 17
compound cursor 34, 40, 57
computing space 48, 57
condition expression 42
conditional expression 40, 42, 43, 44
connect to server 18
connection to mpC daemon 56
copy 29, 30, 31
create new file 31
current cursor 55, 57
Current cursor window 50
current target 14
cursor 35, 40
  amber 34
  green 34
  red 35
cursor color 36
cursor identifier 57
cut 29, 30, 31

D

Debug info window 9, 50
debug object manager 41, 42, 44, 45, 46
debugger 3, 4
Debugger dialog boxes 50
debugging program 48
declaration or usage of statements 22
delete 30, 31
disable breakpoint 42
disable watchpoint 45
Disable(debug) 21
disabled breakpoint 40
display 46, 57
Distributed Debugging 6
distributed project 6, 13
docking window 57

E

edit target 15
edit VPM 12
Editing text 29
enable breakpoint 42
enable watchpoint 45
enabled breakpoint 40
Expressions dialog 47

F

Favor Fast Code 21
Favor Small Code 24
Finding and Replacing Text 27
font color 27
font size 27
Frame-Pointer Omission 25
Full Optimization 25

G

Generate Intrinsic Functions 24
global rank 57
go command 34, 38, 53, 54
go to cursor command 34, 38, 53

H

hostname 18
host-processor 57

I

ignore count 41, 43, 44, 46
ignore count value 39, 42, 43
Improve Float Consistency 24
incorrect declarations 22
Incorrect initialization 22

L

local variable 43
location or usage of statements 22

M

Managing displays 53
Managing watchpoints 53
margin indicator bar 7
merge cursor 35
Minimize Size 24
Modifying displays 47
Moving commands 38
mpC 3
mpC adviser 5
mpC daemon 5, 56, 57
mpC server 18
mpC Workshop 3, 7
mpC Workshop Advantages 4
mpC Workshop architecture 5
mpcd 18, 56
MPCLOAD 14

N

Navigating in files 29
net filter dialog 37
network 57
network filter 34, 37, 38
network filter condition 37, 57
node 11, 13, 57

O

open 31
open project 14
Opening a File 31
Options dialog 26
Output pane 8
Output pane list 8
Output window 8, 50

P

paste 29, 30
prepare programs for debugging 49
Printing a File 32
process cursor 34, 58
Project Settings dialog options 21
project target directory 14

Q

Quick Watch window 9

R

redo 29, 30
remote debugging 34
remove breakpoint 41
remove build 15
remove file 16
remove target 15
Removing displays 47
Removing VPMs 12

Running parallel programs 19

S

Saving Files 32
select text 29
select VPM 12
selection margin 7
Setting displays 47
split cursor 35
spreadsheet field 51
step into 38
Step into command 39
step out 38
Step out command 39
step over 38
Step over command 39
Stepping into functions 54
Stepping over or out of a function 54
subnetwork 58
synchronization point 40, 58
syntax coloring 26

T

target 13, 58
target path 21, 58
Target tab 8
Target window 8
Text Editor overview 26
Text Editor window 7, 8
Toggle Output button 8

U

uncertain distribution of operands 22
uncertain size of operands 22
undo 29, 30

V

Viewing the value of a variable 51
Virtual Parallel Machine 3, 4, 6, 11, 58
virtual processor 39, 58
virtual space 26
VPM 6, 11
VPM node 6, 11
VPM tab 8
VPM window 8

W

warning level 23
Watch window 8, 9, 50
watchpoint 43, 44, 45, 46
watchpoint condition 43, 45, 46
watchpoint dialog 45
Workspace window 8