



MPJ Express: An Implementation of MPI in Java

Windows User Guide

19th March 2014

Document Revision Track

Version	Updates	By
1.0	Initial version document	Aamir Shafi
1.1	A new configuration 'hybrid configuration' is added for executing parallel java applications exploiting hybrid parallelism	Aleem Akhtar, Mohsan Jameel, Aamir Shafi
1.2	A new configuration 'Native MPI configuration' is added for executing parallel java applications on top of a native MPI library.	Bibrak Qamar, Mohsan Jameel, Aamir Shafi

Table of Contents

1	Introduction	5
1.1	Configurations	5
1.1.1	Multicore configuration.....	6
1.1.2	Cluster configuration	6
2	Getting Started with MPJ Express.....	6
2.1	Pre-requisites	8
2.2	Running MPJ Express in the Multicore Configuration	9
2.3	Running MPJ Express in the Cluster Configuration.....	13
2.3.1	niodev Device.....	13
2.3.2	Hybrid Configuration.....	17
3	MPJ Express Debugging	28
3.1	The <code>mpjrun</code> Script	28
3.2	Core Library	28
3.3	MPJ Express Daemons (Cluster configuration only).....	29
3.3.1	Step One: Modifying the <code>wrapper.conf</code> file	29
3.3.2	Step Two: Modifying <code>MPJDaemon.java</code> file	29
	<i>Additional Optional Information: Running daemons in console mode on compute nodes</i>	<i>29</i>
4	Known Issues and Limitations	29
5	Contact and Support	33
6	Appendices	34
	Appendix A: Running MPJ Express without the runtime (manually)	34
	Appendix B: Changing protocol limit switch	36
	Appendix C: MPJ Express Testsuite.....	36

Compiling source code and Testsuite 36

Running Testsuite..... 37

1 Introduction

MPJ Express is a reference implementation of the mpiJava 1.2 API, which is an MPI-like API for Java defined by the Java Grande forum. The mpiJava 1.2 API is the Java equivalent of the MPI 1.1 specification document (<http://www.mpi-forum.org/docs/mpi-1.1-html/mpi-report.html>).

This release of the MPJ Express software contains the core library and the runtime infrastructure. The software also contains a comprehensive test suite that is meant to test the functionality of various communication functions.

MPJ Express is a message passing library that can be used by application developers to execute their parallel Java applications on compute clusters or network of computers. Compute clusters is a popular parallel platform, which is extensively used by the High Performance Computing (HPC) community for large scale computational work. MPJ Express is essentially a middleware that supports communication between individual processors of clusters. The programming model followed by MPJ Express is Single Program Multiple Data (SPMD).

Although MPJ Express is designed for distributed memory machines like network of computers or clusters, it is possible to efficiently execute parallel user applications on desktops or laptops that contain shared memory or multicore processors.

The MPJ Express software can be configured in two ways. The first configuration—known as the Multicore Configuration—is used to execute MPJ Express user programs on laptops and desktops. The second configuration—known as the Cluster Configuration—is used to execute MPJ Express user programs on clusters or network of computers. The cluster configuration has three modes: 1) NIO mode—is used to execute MPJ Express user programs on clusters using Java NIO. 2) Hybrid Configuration—is used to execute MPJ Express user programs on clusters of multicore computers. 3) Native MPI Configuration—is used to execute MPJ Express user programs on top of a native MPI library (MPICH, MVAPICH and Open MPI or MS-MPI etc).

1.1 Configurations

The MPJ Express software can be configured to work on clusters (network of computers), on laptops/desktops (multicore processors) or on cluster of multicore processors.

1.1.1 Multicore configuration

The multicore configuration is meant for users who plan to write and execute parallel Java applications using MPJ Express on their desktops or laptops—typically such hardware contains shared memory and multicore processors. In this configuration, users can write their message passing parallel application using MPJ Express and it will be ported automatically on multicore processors. We envisage that users can first develop applications on their laptops and desktops using multicore configuration, and then take the same code to distributed memory platforms including clusters. Also this configuration is preferred for teaching purposes since students can execute message passing code on their personal laptops and desktops. It might be noted that user applications stay the same when executing the code in multicore or cluster configuration.

Under the hoods, the MPJ Express library starts a single thread to represent MPI process. The multicore communication device uses efficient inter-thread mechanism.

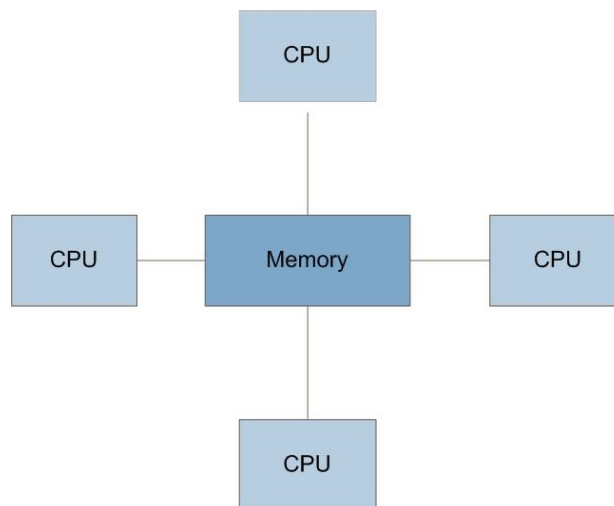


Figure 1: MPJ Express Multicore Configuration Targets the Shared Memory and Multicore Processor Machines

1.1.2 Cluster configuration

The cluster configuration is meant for users who plan to execute their parallel Java applications on distributed memory platforms including clusters or network of computers.

As an example, consider a cluster or network of computers shown in Figure 2 that shows eight compute nodes connected to each other via private interconnect. The MPJ Express cluster configuration will start one MPJ Express process per node, which communicates to each other using message passing.

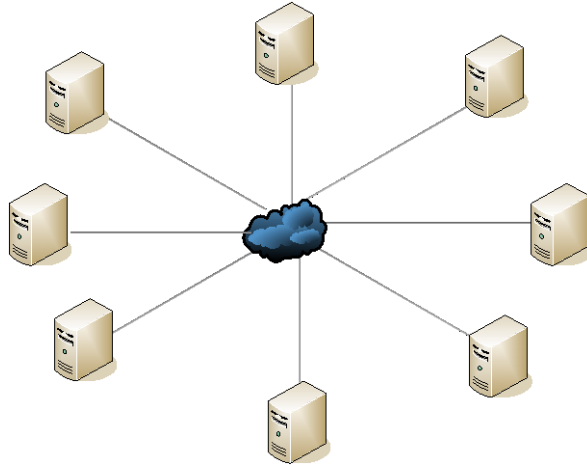


Figure 2: MPJ Express Cluster Configuration Targets the Distributed Memory Platforms Including Clusters and Network of Computers

Application developers can opt to use either of the four communication devices in the cluster configuration:

1. Java New I/O (NIO) device driver known as `niodev`
2. Myrinet device driver known as `mxdev`
3. Hybrid device driver known as `hybdev`
4. Native device driver known as `native`

The Java NIO device driver (also known as `niodev`) can be used to execute MPJ Express programs on clusters or network of computers. The `niodev` device driver uses Ethernet-based interconnect for message passing. On the other hand, many clusters today are equipped with high-performance low-latency networks like Myrinet. MPJ Express also provides a communication device for message passing using Myrinet interconnect—this device is known as `mxdev` and is implemented using the Myrinet eXpress (MX) library by Myricom. These communication drivers can be selected using command line switches.

Modern HPC clusters are mainly equipped with multicore processors (Figure 3). The hybrid configuration is meant for users who plan to execute their parallel Java applications on such a cluster of multicore machines. Hybrid configuration transparently uses both multicore configuration and cluster configuration for intra-node communication and cluster configuration (NIO device only) for inter-node communication, respectively.

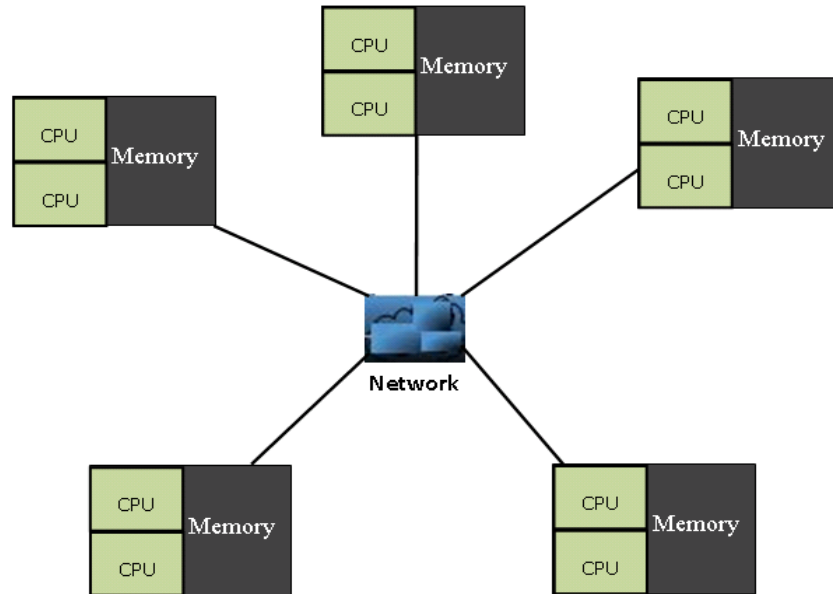


Figure 3: MPJ Express Hybrid Configuration targeting cluster of multicore machines

The fourth device—the Native MPI configuration—is meant for users who plan to execute their parallel Java applications using a native MPI implementation for communication. In this configuration the bulk of messaging logic is offloaded to the underlying MPI library. This is attractive because MPJ Express can exploit latest features, like support for new interconnects and efficient collective communication algorithms, of the native MPI library. Under Windows, this configuration is currently tested and supported for MS-MPI—as the underlying native MPI library.

2 Getting Started with MPJ Express

This section shows how MPJ Express programs can be executed in the multicore, cluster and hybrid configuration

2.1 Pre-requisites

- Java 1.6 (stable) or higher
- Apache ant 1.6.2 or higher (For those who are interested in Compiling source code)
- Perl (Optional): MPJ Express needs Perl for compiling source code because some of the Java code is generated from Perl templates. The build file will generate Java files from

Perl templates if it detects perl on the machine. It is a good idea to install Perl if you want to do some development with MPJ Express.

- A native MPI library such as MS-MPI (For those who are interested in running MPJ Express in native configuration)
- Visual Studio: MPJ Express needs Visual Studio to generate dynamic library (nativempjdev.dll) for running parallel Java programs in the native MPI configuration.

2.2 Running MPJ Express in the Multicore Configuration

This section outlines steps to execute parallel Java programs in the multicore configuration.

1. Download MPJ Express and unpack it
2. Assuming unpacked 'mpj express' is in 'c:\mpj', Right-click My Computer→Properties→Advanced tab→Environment Variables and export the following system variables (user variables are not enough)
 - a. Set the value of variable `MPJ_HOME` as `c:\mpj` [see Fig 4, Fig 5 and Fig 6]
 - b. Append the value of variable `Path` as `c:\mpj\bin` [see Fig 7]

See the snapshots below

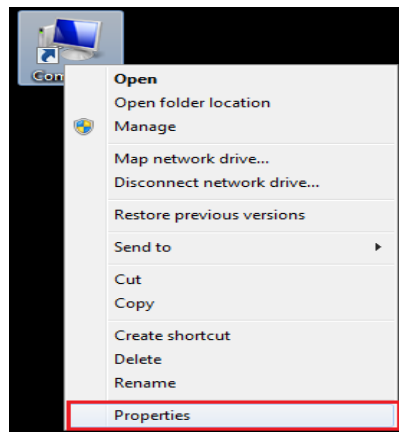


Figure 4: Right click on my computer and select Properties

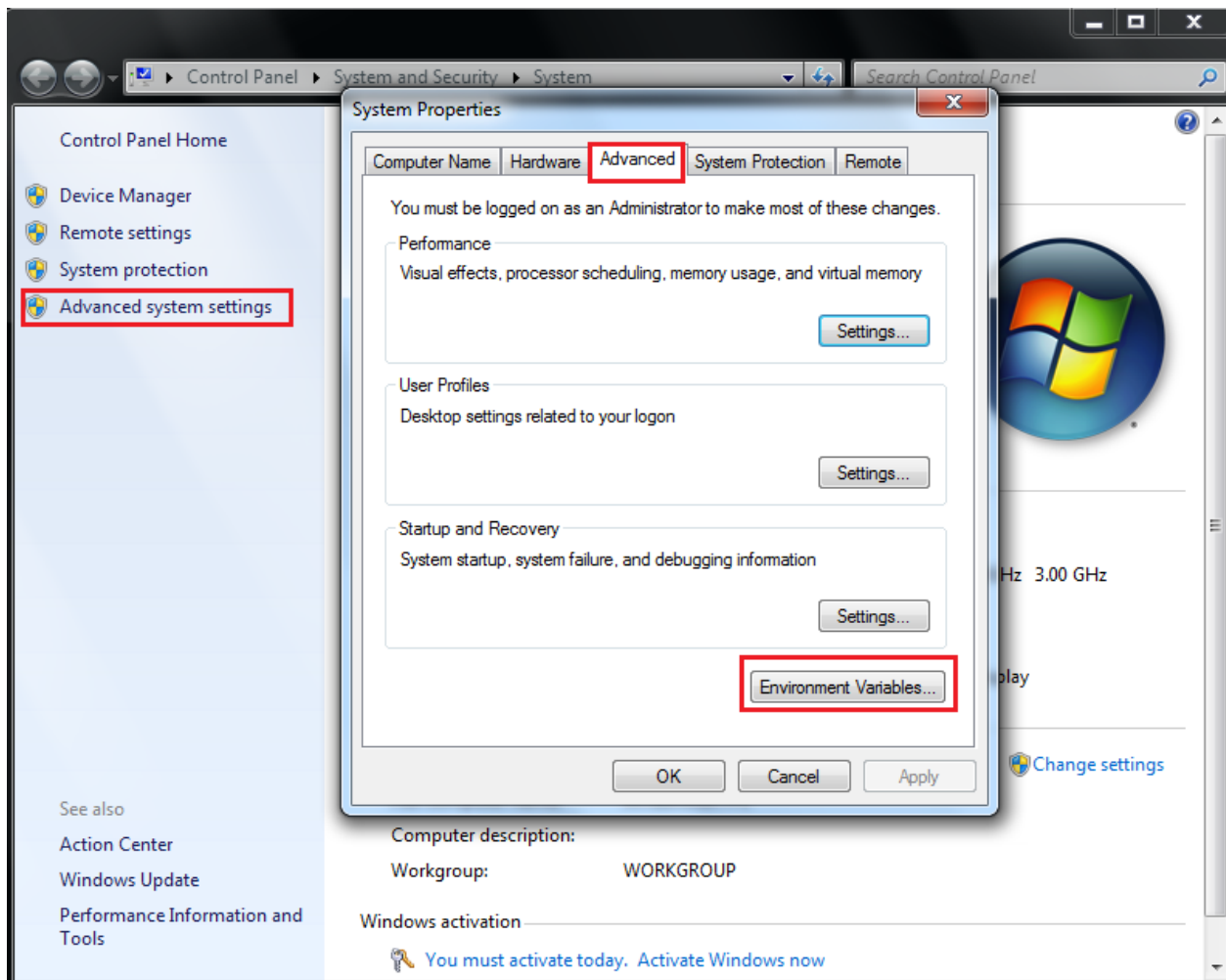


Figure 5: Select Environment Variables to Add/Edit variables

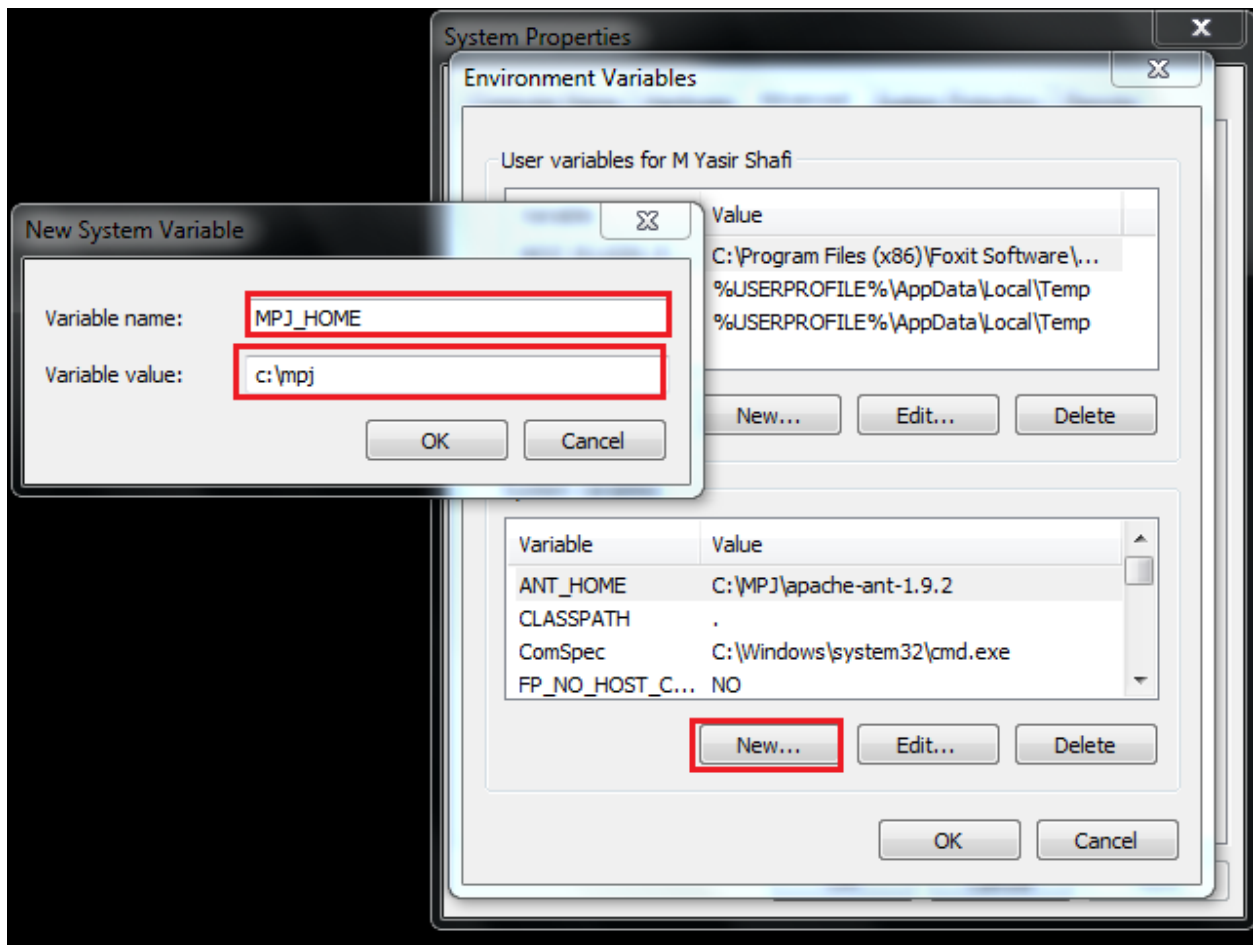


Figure 6: Add MPJ_HOME as new Environment Variable

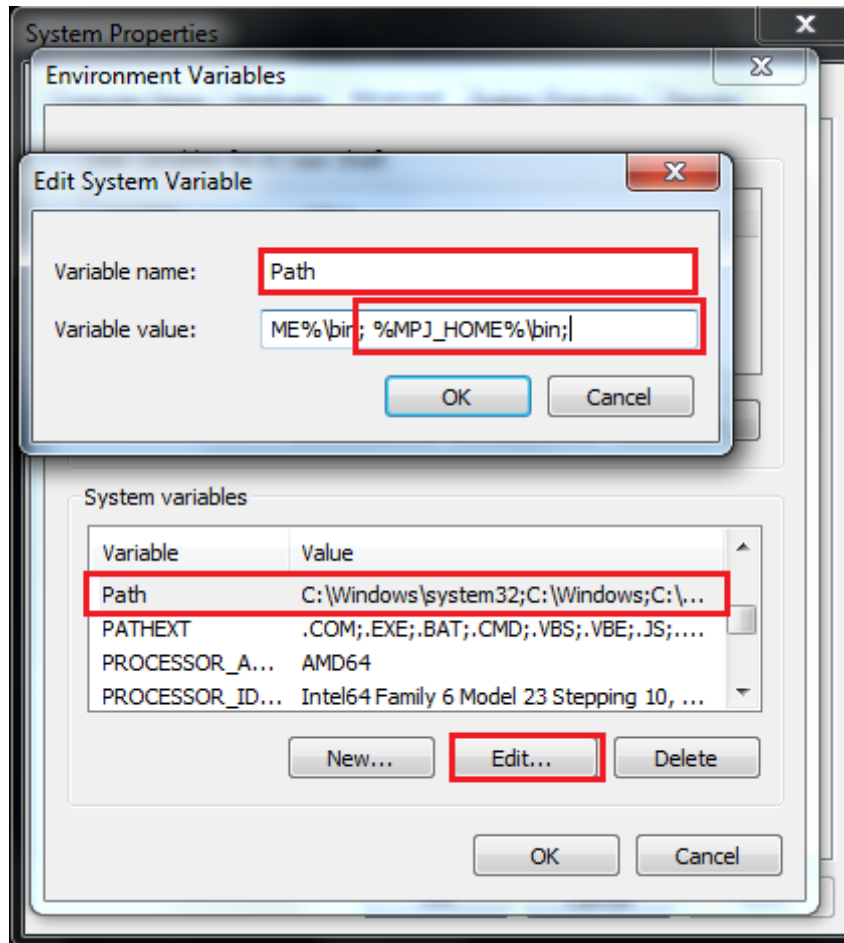


Figure 7: Append Path variable

3. For windows with Cygwin (assuming 'mpj express' is in 'c:\mpj')

The recommended way to is to set variables as in Windows

If you want to set variables in cygwin shell

```
export MPJ_HOME="c:\mpj"
export PATH=$PATH:"$MPJ_HOME\bin"
```

4. Create a new working directory for MPJ Express programs. This document assumes that the name of this directory is "mpj-user".
5. Compile the MPJ Express library (Optional): `cd %MPJ_HOME%; ant`
6. Running test cases (Optional) [Test suite is provided with MPJ Express]
 - a. Compile (Optional): `cd %MPJ_HOME%/test; ant`

b. Execute: `mpjrun.bat -np 2 -jar %MPJ_HOME%/lib/test.jar`

7. Write Hello World MPJ Express program and save it as `HelloWorld.java`

```
import mpi.*;

public class HelloWorld {

    public static void main(String args[]) throws Exception {
        MPI.Init(args);
        int me = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        System.out.println("Hi from <"+me+">");
        MPI.Finalize();
    }
}
```

8. Compile: `javac -cp .;%MPJ_HOME%/lib/mpj.jar HelloWorld.java`

9. Execute: `mpjrun.bat -np 2 HelloWorld`

10. JVM arguments (Optional): JVM arguments may be specified to the `mpjrun` script that passes these directly to the executing MPJ Express processes. For example, the following command modifies the JVM heap size: `mpjrun.bat -np 2 -Xms512M HelloWorld`

11. Application Arguments (Optional): Users may pass arguments to their parallel applications by specifying them after `"-jar <jarname>"` or `"classname"` in the `mpjrun` script:

a. The user may pass three arguments "a", "b", "c" to the application as follows:

`mpjrun.bat -np 2 HelloWorld a b c`

b. Application arguments can be accessed in the program by calling the `String[] MPI.Init(String[] args)` method. The returned array stores user arguments [a,b,c].

2.3 Running MPJ Express in the Cluster Configuration

2.3.1 niodev Device

This section outlines steps to execute parallel Java programs in the cluster configuration with `niodev` communication device driver.

1. Download MPJ Express and unpack it

2. Assuming unpacked 'mpj express' is in 'c:\mpj', Right-click My Computer→Properties→Advanced tab→Environment Variables and export the following system variables (user variables are not enough)
 - a. Set the value of variable `MPJ_HOME` as `c:\mpj` [see Fig 4, Fig 5 and Fig 6]
 - b. Append the value of variable `PATH` as `c:\mpj\bin` [see Fig 7]See the Snapshots [here](#)
3. Create a new working directory for MPJ Express programs. This document assumes that the name of this directory is “mpj-user”.
4. Write a machines file stating machines name, IP addresses, or aliases of the nodes where you wish to execute MPJ Express processes. Save this file as 'machines' in `mpj-user` directory. This file is used by scripts like `mpjboot`, `mpjhalt`, `mpjrun.bat` and `mpjrun.sh` to find out which machines to contact.

Suppose you want to run a process each on 'machine1' and 'machine2', then your machines file would be as follows

```
machine1
machine2
```

Note that in real world, 'machine1' and 'machine2' would be fully qualified names, IP addresses or aliases of your machine

5. Start the daemons
 - a. Run `%MPJ_HOME%\bin\installmpjd-windows.bat`. **Users of Windows Vista and 7 needs to “Run as Administrator”. Also, for running properly, users of Windows Vista and 7 might need to turn off their firewall. For more details click [here](#)**
 - b. Go to Control-Panel→Administrative Tools→Services→MPJ Daemon and start the service. It is important to start the daemon as a user process (preferably the currently logged in user) instead of a SYSTEM process. To start the daemons as user process, right-click MPJ Daemon service, click Properties, click "Log On" tab, For the option "Log on as:", select This account and put in the user name and password of this account, and start the service

See the following screenshots for more elaboration



Figure 8: Start->Control Panel

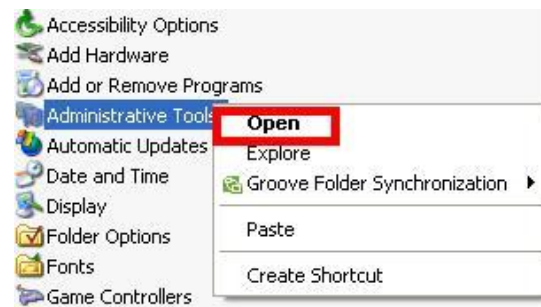


Figure 9: Administrative Tools



Figure 10: Services

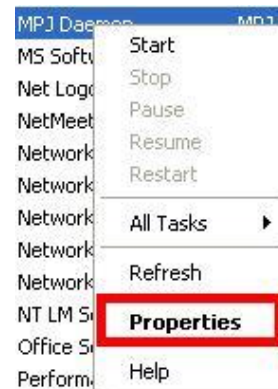


Figure 11: MPJ Daemon Properties

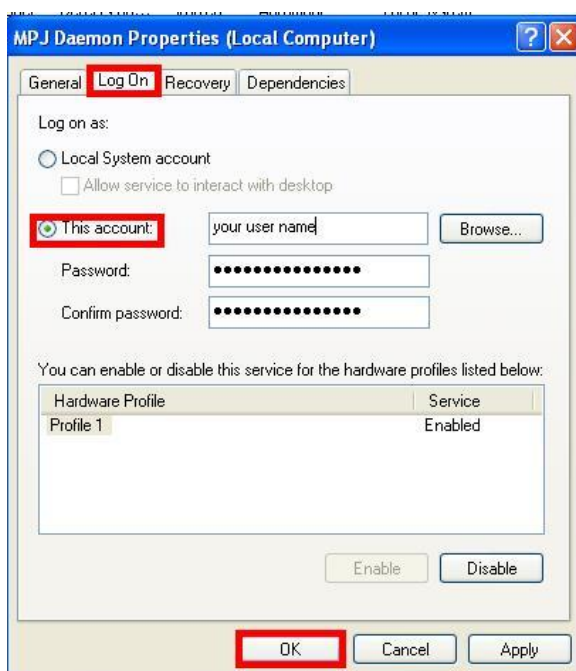


Figure 12: Logon Tab

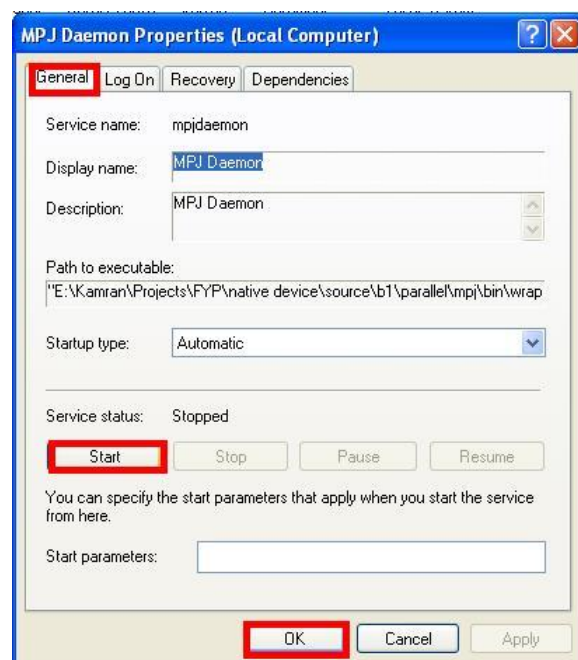


Figure 18: Start the Service

6. Compile the MPJ Express library (Optional): `cd %MPJ_HOME%; ant`
7. Running test cases (Optional) [Test suite is provided with MPJ Express]
 - a. Compile (Optional): `cd %MPJ_HOME%/test; ant`
 - b. Execute: `mpjrun.bat -np 2 -dev niodev -jar %MPJ_HOME%/lib/test.jar`
8. Write Hello World MPJ Express program and save it as `HelloWorld.java`

```
import mpi.*;

public class HelloWorld {

    public static void main(String args[]) throws Exception {
        MPI.Init(args);
        int me = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        System.out.println("Hi from <"+me+">");
        MPI.Finalize();
    }
}
```

9. Compile: `javac -cp .;%MPJ_HOME%/lib/mpj.jar HelloWorld.java`
10. Execute: `mpjrun.bat -np 2 -dev niodev HelloWorld`
11. Stop the daemons: After you are done with executing all the programs, make sure that you halt the daemons. For this go to Control-Panel→Administrative Tools→Services→MPJ Daemon and stop the service
12. JVM arguments (Optional): JVM arguments may be specified to the `mpjrun` script that passes these directly to the executing MPJ Express processes. For example, the following command modifies the JVM heap size: `mpjrun.bat -np 2 -dev niodev -Xms512M HelloWorld`
13. Application Arguments (Optional): Users may pass arguments to their parallel applications by specifying them after `"-jar <jarname>"` or `"classname"` in the `mpjrun` script:
 - a. The user may pass three arguments “a”, “b”, “c” to the application as follows:
`mpjrun.bat -np 2 -dev niodev HelloWorld a b c`

- b. Application arguments can be accessed in the program by calling the `String[] MPI.Init(String[] args)` method. The returned array stores user arguments [a,b,c].

2.3.2 Hybrid Device

This section outlines steps to execute parallel Java programs in the hybrid configuration using multicore and cluster configurations. Hybrid configuration depends on Multicore configuration and Cluster configuration. Make sure that document sections 2.2 and 2.3 are completed successfully.

1. Start the daemons
 - a. Run `%MPJ_HOME%/bin/installmpjd-windows.bat`. **Users of Windows Vista and 7 needs to “Run as Administrator”. Also, for running properly, users of Windows Vista and 7 might need to turn off their firewall. For more details click [here](#)**
 - b. Go to Control-Panel→Administrative Tools→Services→MPJ Daemon and start the service. It is important to start the daemon as a user process (preferably the currently logged in user) instead of a SYSTEM process. To start the daemons as user process, right-click MPJ Daemon service, click Properties, click "Log On" tab, For the option "Log on as:", select This account and put in the user name and password of this account, and start the service. [see Figures 8-13]
2. Compile the MPJ Express library (Optional): `cd %MPJ_HOME%; ant`
3. Running test cases (Optional) [Test suite is provided with MPJ Express]
 - a. Compile (Optional): `cd %MPJ_HOME%/test; ant`
 - b. Execute: `mpjrun.bat -np 2 -dev hybdev -jar %MPJ_HOME%/lib/test.jar`
4. Write Hello World MPJ Express program and save it as `HelloWorld.java`

```
import mpi.*;

public class HelloWorld {

    public static void main(String args[]) throws Exception {
        MPI.Init(args);
        int me = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        System.out.println("Hi from <"+me+">");
        MPI.Finalize();
    }
}
```

5. Compile: `javac -cp .;%MPJ_HOME%/lib/mpj.jar HelloWorld.java`
6. Execute: `mpjrun.bat -np 2 -dev hybdev HelloWorld`
7. Stop the daemons: After you are done with executing all the programs, make sure that you halt the daemons. For this go to Control-Panel→Administrative Tools→Services→MPJ Daemon and stop the service
8. JVM arguments (Optional): JVM arguments may be specified to the mpjrun script that passes these directly to the executing MPJ Express processes. For example, the following command modifies the JVM heap size: `mpjrun.bat -np 2 -dev hybdev -Xms512M HelloWorld`
9. Application Arguments (Optional): Users may pass arguments to their parallel applications by specifying them after "-jar <jarname>" or "classname" in the mpjrun script:
 - a. The user may pass three arguments “a”, “b”, “c” to the application as follows:
`mpjrun.bat -np 2 -dev hybdev HelloWorld a b c`
 - b. Application arguments can be accessed in the program by calling the `String[] MPI.Init(String[] args)` method. The returned array stores user arguments [a,b,c].

2.3.4 Native MPI configuration

This section outlines steps to execute parallel Java programs in the Native MPI configuration.

1. Download MPJ Express and unpack it
2. Assuming unpacked 'mpj express' is in 'c:\mpj', Right-click My Computer→Properties→Advanced tab→Environment Variables and export the following system variables (user variables are not enough)
 - a. Set the value of variable `MPJ_HOME` as `c:\mpj` [see Fig 4, Fig 5 and Fig 6]
 - b. Append the value of variable `PATH` as `c:\mpj\bin` [see Fig 7]See the Snapshots [here](#)

3. Create a new working directory for MPJ Express programs. This document assumes that the name of this directory is “mpj-user”.
4. Write a machines file stating machines name, IP addresses, or aliases of the nodes where you wish to execute MPJ Express processes. Save this file as 'machines' in `mpj-user` directory. This file is used by script `mpjrun.bat` and native MPI library's job execution framework (a.k.a `mpiexec`) to find out which machines to contact.

Suppose you want to run a process each on 'machine1' and 'machine2', then your machines file would be as follows

```
machine1  
machine2
```

Note that in real world, 'machine1' and 'machine2' would be fully qualified names, IP addresses or aliases of your machine

5. Since MPJ Express native MPI configuration relies on a native MPI it is assumed that the user has installed and tested the native MPI library. Better to run a simple helloworld like program to test the native MPI. Currently MPJ Express is only tested on MS-MPI (under Windows).
6. Compile the MPJ Express library (Optional): `cd %MPJ_HOME%; ant`
7. Compile the JNI wrapper library (Mandatory): This requires Visual Studio to generate a dynamic library (`nativempjdev.dll`) to be used by MPJ Express to interface with the native MPI library. Open Visual Studio and follow the steps provided below:
 - a. File→New→Project: Create a Win32 Project with the name of **nativempjdev**

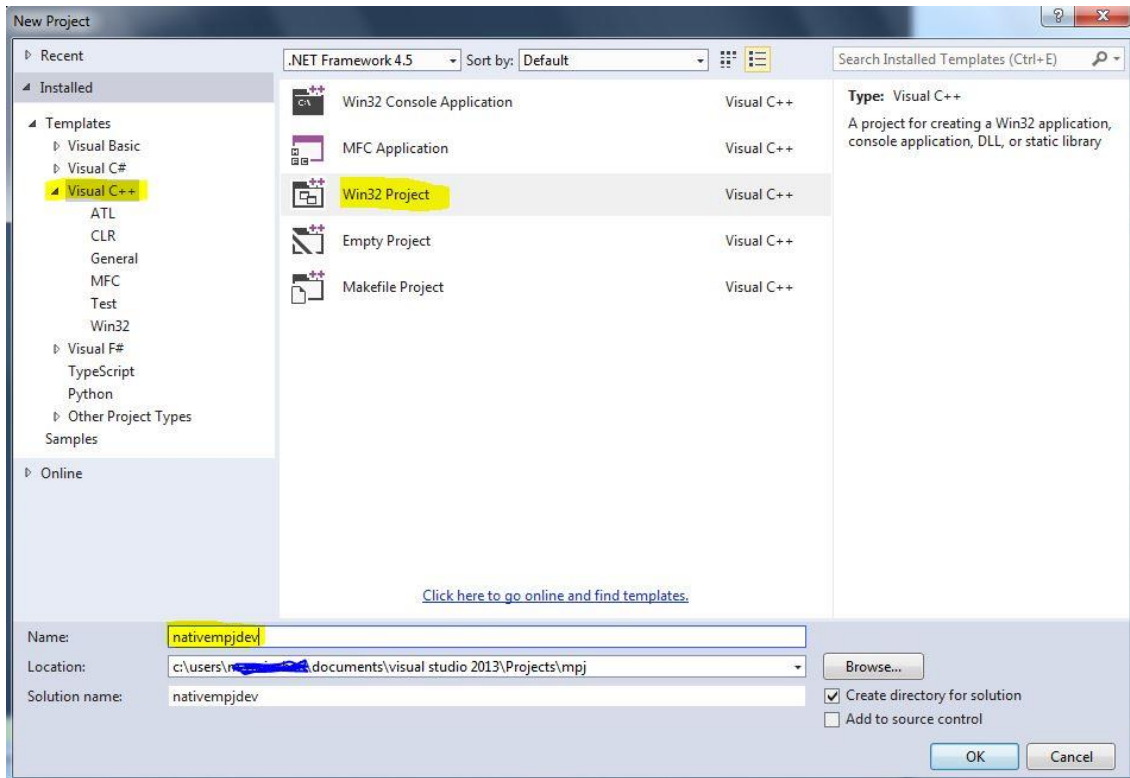


Figure 9: Create a Win32 Project with the name of nativempjdev

- b. Click next → set Application type as DLL and in Additional options tick Empty project → finish

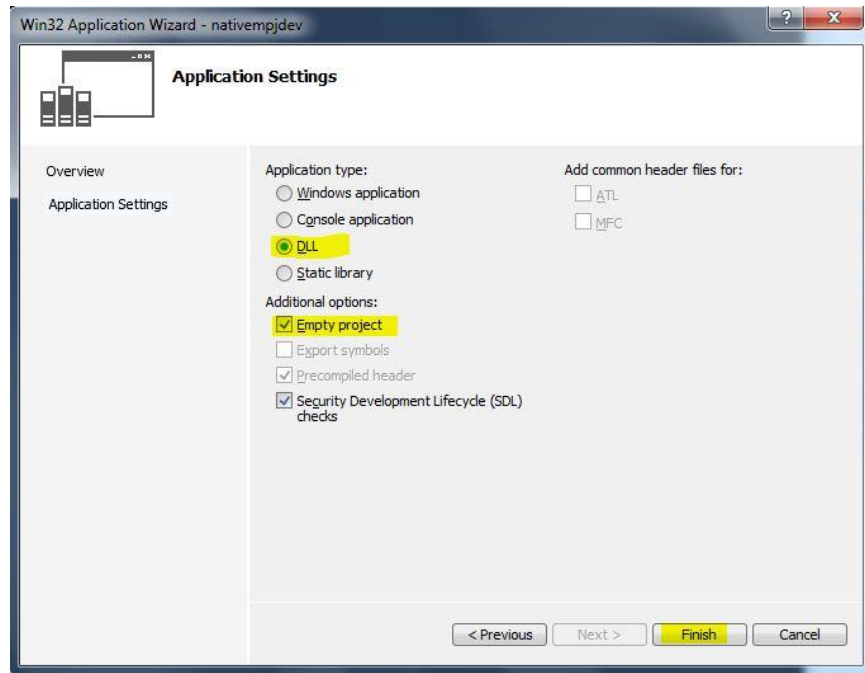


Figure 10: Set Application type as DLL and in Additional options tick Empty project

- c. Right click on project nativempjdev in the Solution Explorer and go to properties.
Set Additional Include Directories

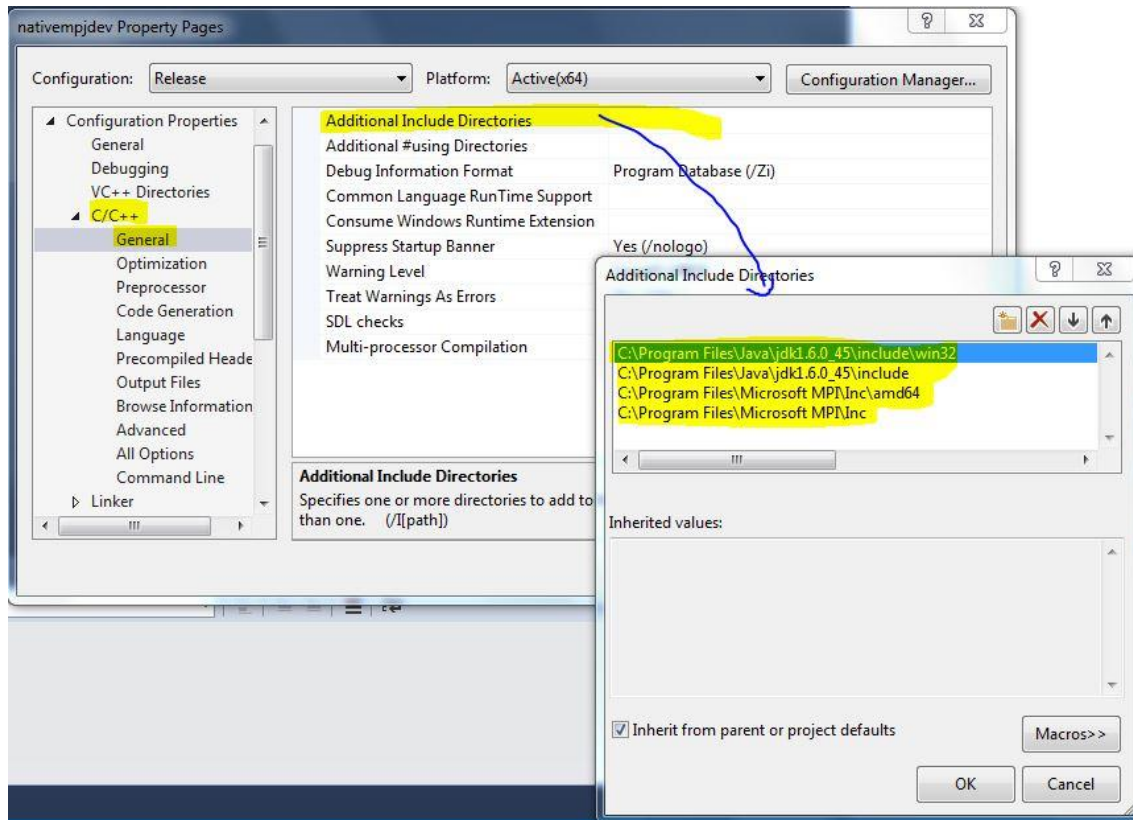


Figure 11: Set Additional Include Directories

- d. Set Additional Library Directories in the Linker

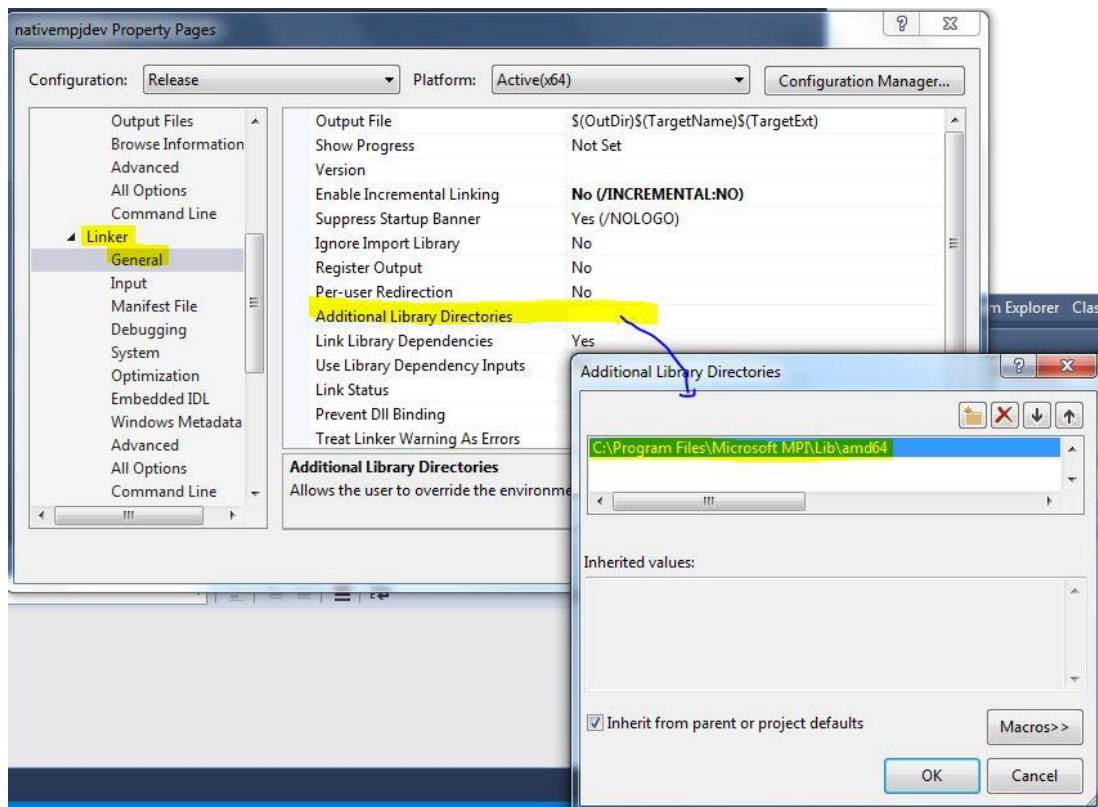
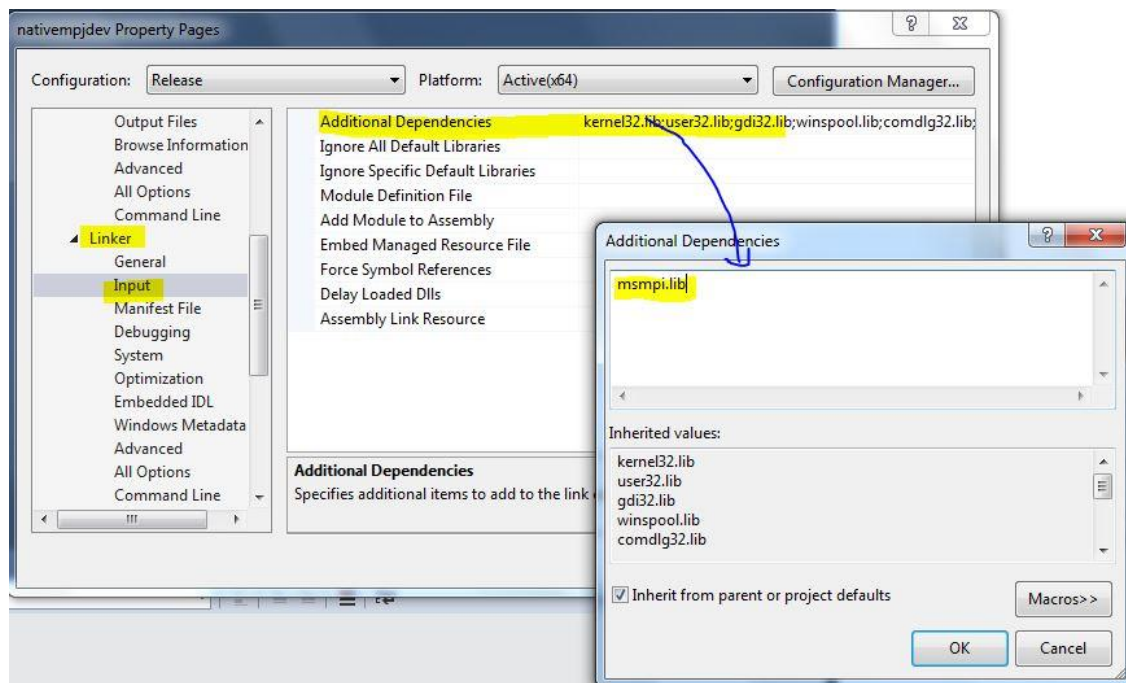


Figure 12: Set Additional Include Directories

e. Set Additional Dependencies (msmpi.lib) in the Linker



f. Right click on 'Header Files' → Add → Existing Item... to add Header Files

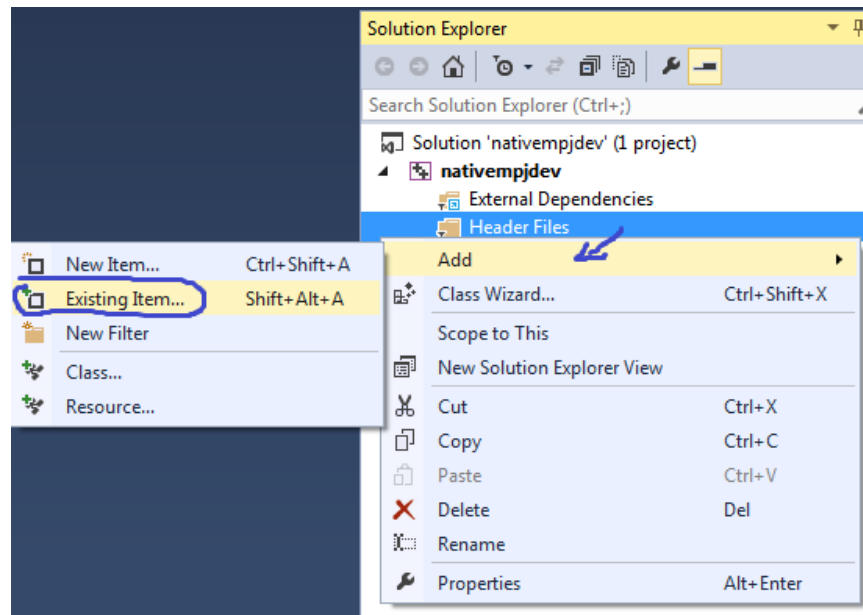


Figure 13: Navigate to Header Files under the solution nativempjdev

- g. Browse into %MPJ_HOME%\src\mpjdev\natmpjdev\lib and select the header files (*.h)

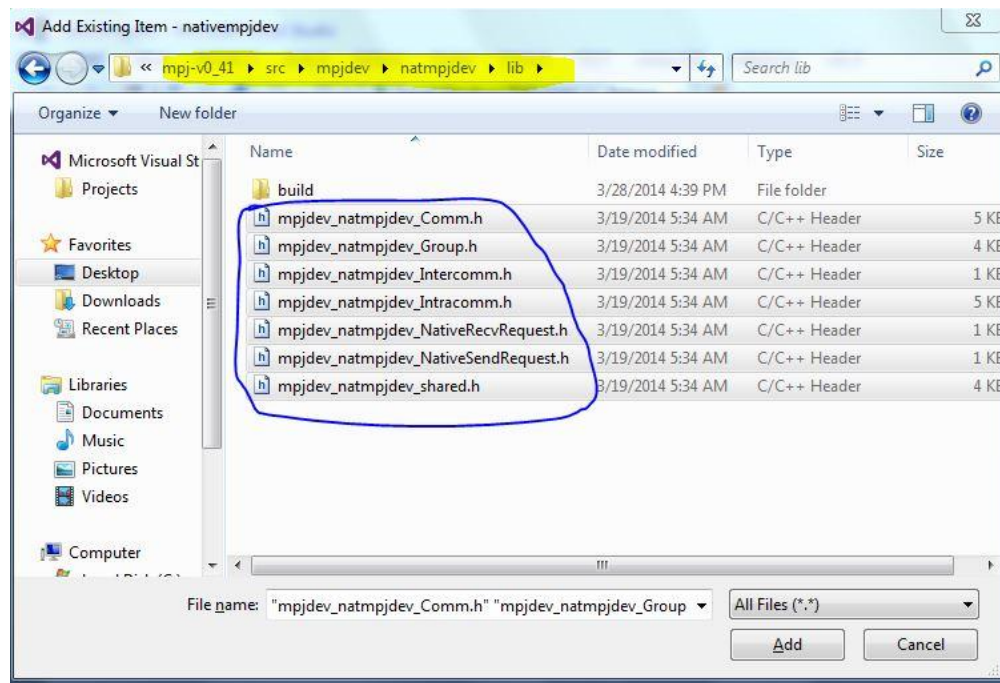


Figure 14: Add header files

- a. Right click on 'Source Files' → Add → Existing Item... to add Source Files

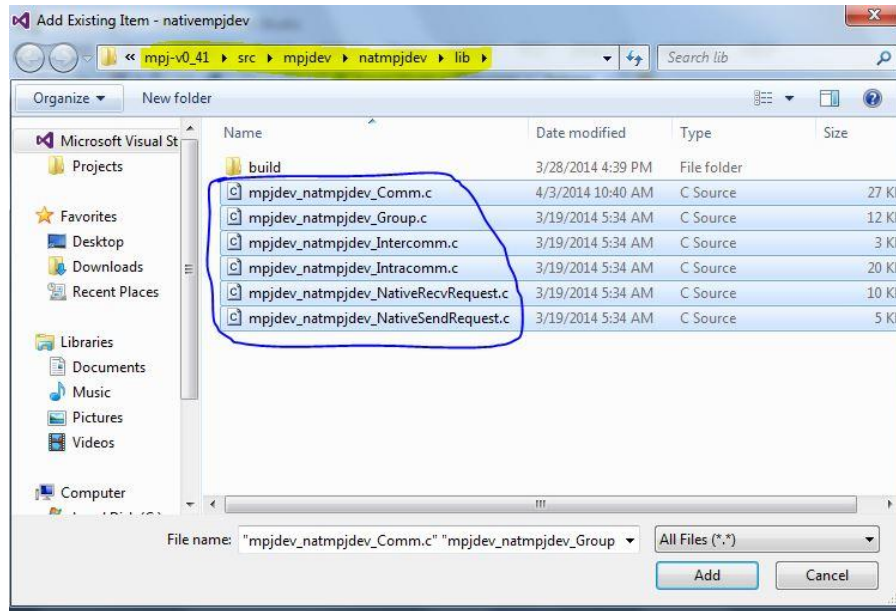


Figure 15: Add source files

- b. Right click on nativempjdev Solution and Build. This creates the dynamic library (nativempjdev.dll)

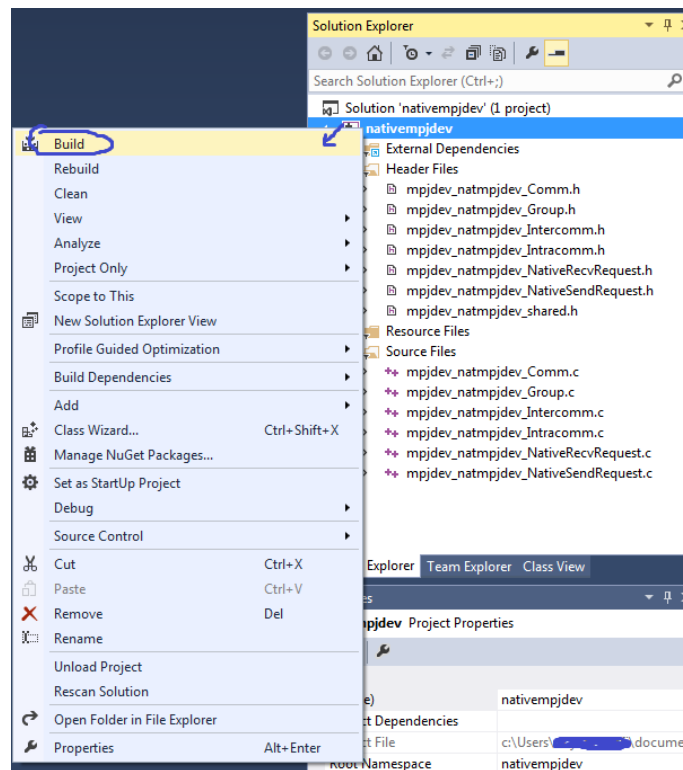


Figure 16: Build

- c. Install the newly created 'nativempjdev.dll'. Copy from your project\folder\x64\Release

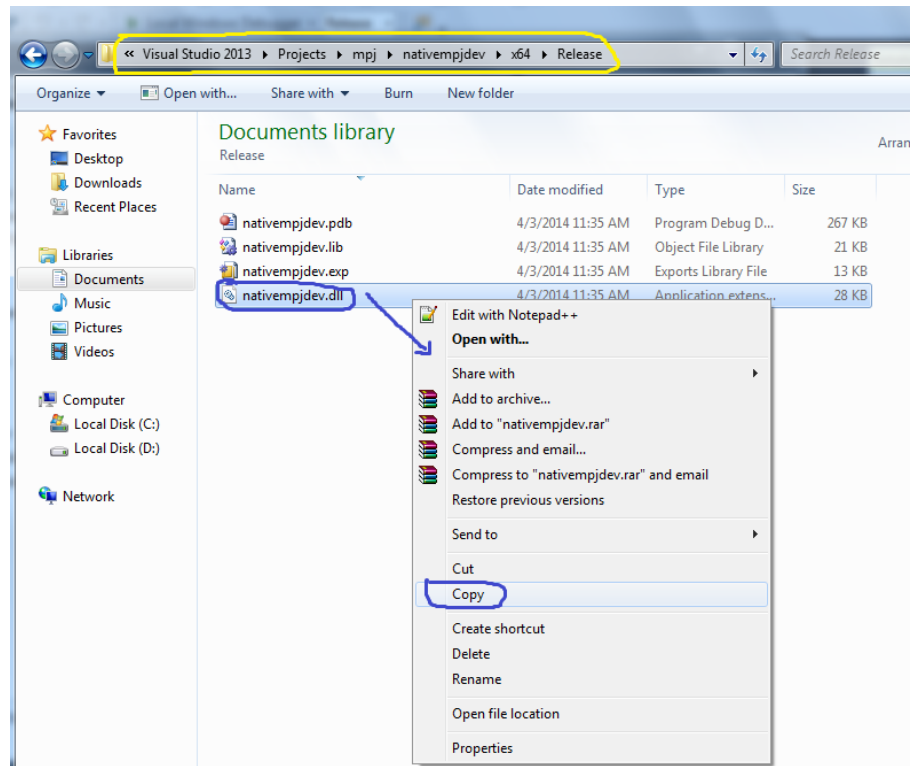


Figure 17: Copy nativempjdev.dll

- d. Paste 'nativempjdev.dll' into %MPJ_HOME%\lib

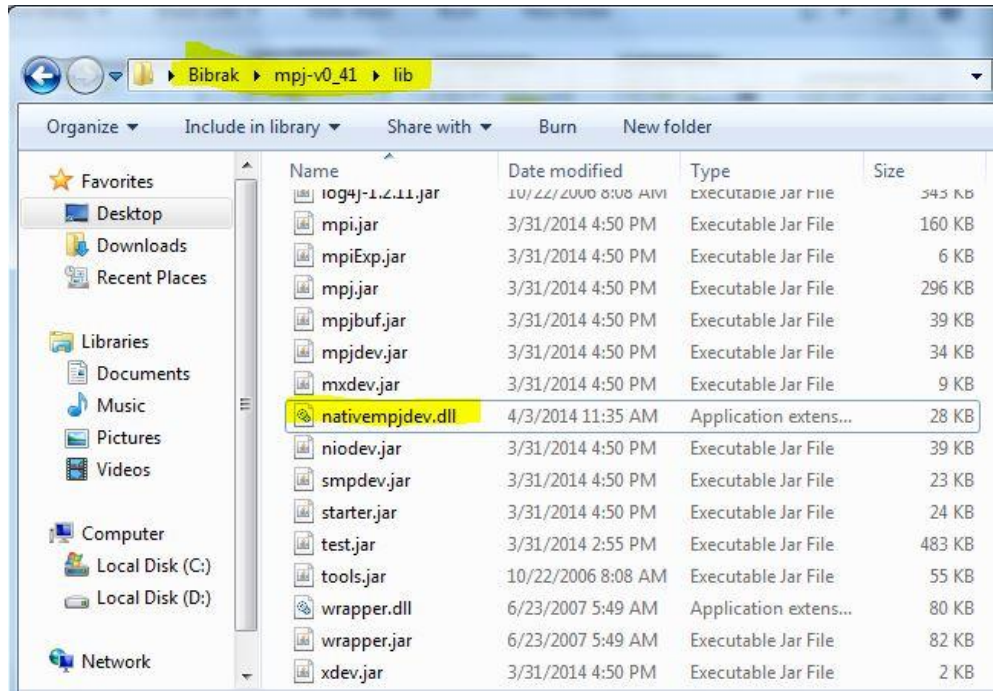


Figure 18: Paste nativempjdev.dll into %MPJ_HOME%\lib

8. Running test cases

a. Compile :

```
cd "%MPJ_HOME%"/test/nativetest
```

```
compile.bat
```

b. Execute:

```
cd "%MPJ_HOME%"/test/nativetest
```

```
runtest.bat
```

- i. To supply a machine file provide full path in the first argument of this script: `runtest.bat /full/path/to/machinefile`

Write Hello World MPJ Express program and save it as HelloWorld.java

```
import mpi.*;

public class HelloWorld {

    public static void main(String args[]) throws Exception {
        MPI.Init(args);
        int me = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        System.out.println("Hi from <"+me+">");
        MPI.Finalize();
    }
}
```

1. Compile: `javac -cp .;"%MPJ_HOME%/lib/mpj.jar HelloWorld.java`
2. Execute: `mpjrun.bat -np 2 -dev native HelloWorld`
3. JVM arguments: Please refer to section 2.3.4.1 for details on how to pass JVM arguments like `-cp`, `-Xms512M` and `-Djava.library.path` and more.
4. Application Arguments: Users may pass arguments to their parallel applications by specifying them after `-jar <jarname>` or `"classname"` in the `mpjrun` script:
 - a. The user may pass three arguments “a”, “b”, “c” to the application as follows:
`mpjrun.bat -np 2 -dev native HelloWorld a b c`
 - a. Application arguments can be accessed in the program by calling the `String[] MPI.Init(String[] args)` method. The returned array stores user arguments [a,b,c].
`String appArgs[] = MPI.Init(args);`

2.3.4.1 Advanced Options:

Running directly with `mpiexec` to use options provided by native MPI library

This is for the advanced user who wants to run parallel Java programs using custom options to the native MPI library.

The `mpjrun.bat` script provides a wrapper to native `mpiexec` (`mpirun`) command. The user can bypass `mpjrun.bat` and directly call `mpiexec` using the following template.

```
mpiexec -np <number of processes> -machinefile </path/to/file/filename> java -cp "%MPJ_HOME%/lib/mpj.jar;. -Djava.library.path=%MPJ_HOME%/lib HelloWorld 0 0 native userarg1 userarg2 userarg3
```

The above template consists of three parts: **mpiexec**, **java** and **user application**. In this way the user has flexibility to supply three kinds of options:

1. `mpiexec`: these are supplied to native MPI library bootstrapping framework (a.k.a `mpirun`), for example `-np` and `-machinefile`
2. `java`: these are supplied to the JVM for example `-cp`, `-Xms512M` and `-Djava.library.path` and more.
3. `user application`: these are supplied to the user application for example `userarg1 userarg2 userarg3` in the above template. The three arguments `0 0 native` following

user application (classname or jar) are reserved for MPJ Express and are to be kept intact. MPJ Express for conventional reasons searches for device name on argument index 3 (i.e `args[2]`).

3 MPJ Express Debugging

This section shows how to debug various modules of the MPJ Express software. It is possible to debug MPJ Express on three levels:

1. The `mpjrun` Script: This script allows bootstrapping MPJ Express programs in cluster of multicore configuration.
2. Core Library: Internals of the MPJ Express Software
3. MPJ Express Daemons: While running the cluster configuration, daemons execute on compute nodes and are responsible for starting and stopping MPJ Express processes when contacted by the `mpjrun` script.

3.1 The `mpjrun` Script

To turn ON debugging for the `mpjrun` script, follow these steps:

1. Edit `%MPJ_HOME%/src/runtime/starter/MPJRun.java` and change the value of `static final boolean DEBUG` flag to `true`
2. Recompile the code: `cd %MPJ_HOME% ; ant`
3. The `mpjrun` script relevant log file is `%MPJ_HOME%/logs/mpjrun.log` file

3.2 Core Library

To turn ON debugging for the core library, follow these steps:

1. Edit `%MPJ_HOME%/src/mpi/MPI.java` and change value of `static final boolean DEBUG` flag to `true`
2. Recompile the code: `cd %MPJ_HOME% ; ant`
3. If the total number of MPJ Express processes is two, then the relevant log files will be `%MPJ_HOME%/logs/mpj0.log` and `%MPJ_HOME%/logs/mpj1.log` for processes 0 and 1 respectively.

3.3 MPJ Express Daemons (Cluster configuration only)

The MPJ Express daemons running on compute nodes can be debugged in two steps outlined below:

3.3.1 Step One: Modifying the wrapper.conf file

1. Edit `%MPJ_HOME%/conf/wrapper.conf` file.
2. Change the value of `wrapper.logfile.loglevel` from "NONE" to "DEBUG".
3. Now the output of `mpjboot`, `mpjhalt`, and other daemon activities can be seen in `$MPJ_HOME/logs/wrapper.log` file. This information is pretty useful for diagnosing and fixing daemons errors.

3.3.2 Step Two: Modifying MPJDaemon.java file

1. Edit `%MPJ_HOME%/src/runtime/daemon/MPJDaemon.java` file.
2. Change the value of static final boolean `DEBUG` flag to `true`
4. Recompile the code: `cd %MPJ_HOME% ; ant`
3. Now log files can be seen in `%MPJ_HOME%/logs/daemon-<machine_name>.log` file.

Additional Optional Information: Running daemons in console mode on compute nodes

For debugging purposes, sometimes it is useful to run the daemons in console mode on compute nodes. This can be achieved in the following way:

1. `cd %MPJ_HOME%/bin`
2. Execute `./mpjdaemon.bat`. Choose the appropriate script for your machine.

4 Known Issues and Limitations

A list of known issues and limitations of the MPJ Express software are listed below.

1. Users of the Windows Vista and 7 might find `installmpjd-windows.bat` script not working directly. In such case, try executing with "Run as Administrator". If the issue still persists, try executing the script `mpjdaemon.bat` directly or turning off the firewall.

2. There is a known (up to some extent) problem on Windows and Solaris that results in hanging MPJ processes. Normally this will be observed when MPJ test-cases will hang, as result, not completing or throwing any error message.

We partially understand the problem but if some user encounters this problem, we would request some more debugging information. The required information can be obtained as follows. Edit `%MPJ_HOME%/src/xdev/niodev/NIODevice.java` and goto line 3693 and uncomment the line `"ioel.printStackTrace() ;"`. The line 3673 is in the MPJ Express release 0.34 and it might change in the future. The general code snippet is like this:

```
catch (Exception ioel) {
    if(mpi.MPI.DEBUG && logger.isDebugEnabled() ) {
        logger.debug(" error in selector thread "
            + ioel.getMessage());
    }
    //ioel.printStackTrace() ;
} //end catch(Exception e) ...
if(mpi.MPI.DEBUG && logger.isDebugEnabled()) {
    logger.debug(" last statement in selector thread");
}
} //end run()
}; //end selectorThread which is an inner class
```

As a result now, when test-cases are executed again, users will see stacks periodically. Most of these are related to socket closed exceptions that are normal. If the code hangs now, the latest stack trace that is not about socket being closed is perhaps the reason of this hanging behavior. We would request the users to kindly email us the output so that we can fix the problem. A stack trace that leaves MPJ Express hanging on Solaris is as follows:

```
java.nio.channels.CancelledKeyException
at sun.nio.ch.SelectionKeyImpl.ensureValid(SelectionKeyImpl.java:55)
at sun.nio.ch.SelectionKeyImpl.readyOps(SelectionKeyImpl.java:69)
at java.nio.channels.SelectionKey.isAcceptable(SelectionKey.java:342)
at xdev.niodev.NIODevice$2.run(NIODevice.java:3330)
at java.lang.Thread.run(Thread.java:595)
```

3. Some users have noticed that it takes a long time to bootstrap MPJ Express processes. For example,

```
user@machine:~/mpj-user> mpjrun.sh -np 6 -jar $MPJ_HOME/lib/test.jar
16:15:43.400 EVENT Starting Jetty/4.2.23
16:15:43.415 EVENT Started HttpContext[/]
16:15:43.419 EVENT Started SocketListener on 0.0.0.0:15000
16:15:43.419 EVENT Started org.mortbay.http.HttpServer@23ac23ac
16:15:43.420 EVENT Starting Jetty/4.2.23
16:15:43.420 EVENT Started HttpContext[/]
16:15:43.421 EVENT Started SocketListener on 0.0.0.0:15001
16:15:43.421 EVENT Started org.mortbay.http.HttpServer@50265026
[ pause for a minute or two ]
```

```
Starting process <0> on Starting process <1> on
```

```
[ pause for a minute or two ]
```

```
Starting process <2> on Starting process <3> on
```

```
[ pause for a minute or two ]  
Starting process <4> on Starting process <5> on  
[ job starts ]
```

Thanks to Andy Botting who is one of the users that identified this problem. This problem is perhaps related to name resolution and we are currently working to fix it.

4. The merge operation is implemented with limited functionality. The processes in local-group and remote-group **have** to specify 'high' argument. Also, the value specified by local-group processes should be opposite to remote-group processes.
5. Any message sent with `MPI.PACK` can only be received by using `MPI.PACK` as the datatype. Later, `MPI.Unpack(..)` can be used to unpack different datatypes
6. Using 'buffered' mode of send with `MPI.PACK` as the datatype really does not use the buffer specified by `MPI.Buffer_attach(..)` method.
7. `Cartcomm.Dims_Create(..)` is implemented with limited functionality. According to the MPI specifications, non-zero elements of 'dims' array argument will not be modified by this method. In this release of MPJ Express, all elements of 'dims' array are modified without taking into account if they are zero or non-zero.
8. `Request.Cancel(..)` is not implemented in this release.
9. MPJ applications should not print more than 500 characters in one line. Some users may use `System.out.print(..)` to print more than 500 characters. This is not a serious problem, because printing 100 characters 5 times with `System.out.println(..)` will have the same effect as printing 500 characters with one `System.out.print(..)`
10. Some users may see this exception while trying to start the `mpjrun` module. This can happen when the users are trying to run `mpjrun.bat` script. The reason for this error is that the `mpjrun` module cannot contact the daemon and it tries to clean up the resources it has. In doing so, it tries to delete a file named 'mpjdev.conf' using `File.deleteOnExit()` method. This method appears not to work on Windows possibly because of permission issues.

```
Exception in thread "main" java.lang.RuntimeException: Another mpjrun module is  
already running on this machine  
at runtime.starter.MPJRun.(MPJRun.java:135)  
at runtime.starter.MPJRun.main(MPJRun.java:925)
```

This issue can be resolved by deleting `mpjdev.conf` file. This file would be present in the directory, where your main class or JAR file is present. So for example, if the users are trying to run `"-jar ../lib/test.jar"`, then this file would be present in `../lib` directory.

11. Permission issues while using MPJ Express runtime with Windows

Problem: The users may run into issues with starting daemons on Windows. The reason is that when MPJ Express processes are started manually, the owner is the user who started them. Thus the log files produced by these processes are owned by the user. On the other hand, the daemon is installed as a SYSTEM service. Thus, while starting the daemon, it may not be able to write to the log file, because logs directory is owned by user, whereas for daemon to be able to write to this directory, it has to be globally accessible. Even when the daemon is started, MPJ Express processes may not be able to write process log files because these log files are owned by the user, whereas now they are required to be globally accessible as MPJ Express processes started by the daemon are also SYSTEM processes. So the problem is caused if the users switch from running their code manually to the runtime, or possibly vice-versa.

Solution: This can be avoided by starting MPJ Express daemons as user process instead of SYSTEM process. To restart the daemons as user process, goto "Control Panel->Administrative Tools->Services", right-click MPJ Daemon service, click Properties, click "Log On" tab, For the option "Log on as:", select This account and put in the user name and password of this account, and restart the service. It should now be started as a user process. To make sure if it is running as a user process, open process manager by pressing "ctrl-alt-delete" and look for processes "wrapper.exe" and "java.exe". The UserName should be the user name of this account instead of SYSTEM. There may be other java processes running on the machine, which may end up showing multiple java.exe on the process list. If this is the case, then "wrapper.exe" is the only process that is representing the MPJ Express daemon

Delete all log files before the first execution

Execute following on cygwin

```
chmod a+w $MPJ_HOME/logs
chmod a+x $MPJ_HOME/lib/*.dll
chmod a+w $MPJ_HOME/logs/wrapper.log (If wrapper.log is present)
```

12. The MPJ Express infrastructure does not deal with security. The MPJ Express daemons could be a security concern, as these are Java applications listening on a port to execute user-code. It is therefore recommended that the daemons run behind a suitably configured firewall, which only listens to trusted machines. In a normal scenario, these daemons would be running on the compute-nodes of a cluster, which are not accessible to outside world. Alternatively, it is also possible to start MPJ Express processes 'manually', which could help avoid runtime daemons. In addition, each MPJ Express process starts at least one server socket, and thus is assumed to be running on machine with configured firewall. Most MPI implementations assume firewalls as protection mechanism from the outside world.

5 Contact and Support

For help and support, join and post on the MPJ Express mailing list (<https://lists.sourceforge.net/lists/listinfo/mpjexpress-users>). Alternatively, you may also contact us directly:

- 1 Aamir Shafi (aamir.shafi@seecs.edu.pk)
- 2 Mohsan Jameel (mohsan.jameel@seecs.edu.pk)
- 3 Bryan Carpenter (bryan.carpenter@port.ac.uk)
- 4 Mark Baker (<http://acet.rdg.ac.uk/~mab>)
- 5 Guillermo Lopez Taboada (<http://www.des.udc.es/~gltaboada>)

6 Appendices

Appendix A: Running MPJ Express without the runtime (manually)

There are two fundamental ways of running MPJ Express applications. The first, and the recommended way is using the MPJ Express runtime infrastructure, alternatively the second way involves the 'manual' start-up of MPJ Express processes. We do not recommend starting programs manually as normal procedure. This section documents the procedure for manual start-up, mainly to allow developers the flexibility to create their own initiation mechanisms for MPJ Express programs. The runmpj.sh script can be considered one example of such a mechanism

1. `cd mpj-user`
2. This document is assuming mpj-user as the working directory for users. The name mpj-user itself has no significance.
3. Write a configuration file called 'mpj.conf' as follows.
 - a. A typical configuration file that would be used to start two MPJ Express processes is as follows. Note the names 'machine1' and 'machine2' would be replaced by aliases/fully-qualified-names/ IP-addresses of the machines where you want to start MPJ Express processes

```
# Number of processes
2
# Protocol switch limit
131072
# Entry in the form of machinename@port@rank
machine1@20000@0
machine2@20000@1
```

- b. The lines starting with '#' are comments. The first entry which is a number ('2' above) represents total number of processes. The second entry, which is again a number ('131072' above) is the protocol switch limit. At this message size, MPJ Express changes its communication protocol from eager-send to rendezvous. There are a couple of entries, one for each MPJ Express process, each in the form of machine name (OR)IP@PORT_NUMBER@RANK. Using this, the users of MPJ Express can control where each MPJ Express process runs, what server port it uses, and what should be the rank of each process. The rank specified here should exactly match the rank argument provided while manually starting MPJ Express processes (using java command). When the users decide to run their code using `mpjrun.bat`, this file is generated programmatically.

- c. Sample configuration files can be found in `%MPJ_HOME%/conf` directory. If you wish to start MPJ processes on `localhost`, see `%MPJ_HOME%/conf/local2.conf` file.
- d. Each MPJ process uses two ports. Thus, do not use consecutive ports if you are trying to execute multiple MPJ Express processes on same node. A sample file for running two MPJ Express processes on same machine would be

```
# Number of processes
2
# Protocol switch limit
131072
# Entry in the form of machinename@port@rank
localhost@20000@0
localhost@20002@1
```

4. Running your MPJ Express program.

- a. The script `runmpj.sh` requires password-less SSH access to machines listed in the configuration file. This script will not work if your machines are not setup for this. You may get some guidance [here](#) regarding setting up SSH so that no password/passphrase is required at login. This is the only script in this software which requires password-less access. An alternative to using `runmpj.sh` is the manual start-up (using `java` command directly -- see directions below)

b. Running class files

For all the machines listed in `mpj.conf`, login to each Windows machine, change directory to `%MPJ_HOME%`

```
java -cp .;%MPJ_HOME%/lib/mpj.jar World <rank> mpj.conf niodev
```

The `<rank>` argument should be 0 for process 0 and 1 for process 1. This should match to what has been written in configuration file (`mpj.conf`). Check the entry format in the configuration file to be sure of the rank

c. Running JAR files

For all the machines listed in `mpj.conf`, login to each Windows or Linux machine

```
java -jar hello.jar <rank> mpj.conf niodev
```

The `<rank>` argument should be 0 for process 0 and 1 for process 1. This should match to what has been written in configuration file (`mpj.conf`). Check the entry format in the configuration file to be sure of the rank.

Appendix B: Changing protocol limit switch

MPJ Express uses two communication protocols: the first is 'eager-send', which is used for transferring small messages. The other protocol is rendezvous protocol useful for transferring large messages. The default protocol switch limit is 128 KBytes. This can be changed prior to execution in following ways depending on whether you are running processes manually or using the runtime.

1. Running MPJ Express applications manually (without using runtime): The users may edit configuration file (for e.g. `%MPJ_HOME%/conf/mpj2.conf`) to change protocol switch limit. Look at the comments in this configuration file. The second entry, which should be 131072 if you have not changed it, represents protocol switch limit
2. Running MPJ Express applications with the runtime: Use `-psl <val>` switch to change the protocol switch limit

Appendix C: MPJ Express Testsuite

MPJ Express contains a comprehensive test suite to test the functionality of almost every MPI function. This test suite consists mainly of mpiJava test cases, MPJ JGF benchmarks, and MPJ microbenchmarks. The mpiJava test cases were originally developed by IBM and later translated to Java. As this software follows the API of mpiJava, these test cases can be used with a little modification. MPJ JGF benchmarks are developed and maintained by [EPCC at the University of Edinburgh](#). MPJ Express is redistributing these benchmarks as part of its test suite. The original copyrights and license remain intact as can be seen in source-files of these benchmarks in `$MPJ_HOME/test/jgf_mpj_benchmarks`. Further details about these benchmarks can be seen [here](#). MPJ Express also redistributes micro-benchmarks developed by [Guillermo Taboada](#). Further details about these benchmarks can be obtained [here](#)

Compiling source code and Testsuite

1. Compiling MPJ Express source code
 - a. Being in `%MPJ_HOME%` directory, execute `ant`

Produces `mpj.jar`, `daemon.jar`, and `starter.jar` in `lib` directory
2. Compiling MPJ Express test-code
 - a. `cd test; ant` This produces `test.jar` in `lib` directory.

Running Testsuite

The suite is located in `%MPJ_HOME%/tests` directory. The test cases have been changed from their original versions, in order to automate testing. `TestSuite.java` is the main class that calls each of the test case present in this directory. The `build.xml` file present in test directory, compiles all test cases, and places `test.jar` into the lib directory. By default, JGF MPJ benchmarks and MPJ micro-benchmarks are disabled. Edit `%MPJ_HOME%/test/TestSuite.java` to uncomment these tests and execute them. Note, after changing `TestSuite.java`, you will have to recompile the testsuite by executing 'ant' in test directory.

1. `cd mpj-user`

With Runtime

1. Write a machines file
2. `mpjrun.bat -np 2 -jar %MPJ_HOME%/lib/test.jar`

Without Runtime

1. Write a configuration file called 'mpj.conf'. Further details about writing configuration file and its format can be found [here](#)
 - a. Start the tests

For all the machines listed in `mpj.conf`, login to each Windows or Linux machine, type,

```
java -jar %MPJ_HOME%/lib/test.jar <rank> mpj.conf niodev
```

The `<rank>` argument should be 0 for process 0 and 1 for process 1. This should match to what has been written in configuration file (`mpj.conf`). Check the entry format in the configuration file to be sure of the rank.