DeinoMPI

Deino Software © 2006

Contents

Introduction

DeinoMPI is an implementation of the MPI-2 standard for Microsoft Windows originally derived from the MPICH2 distribution from Argonne National Laboratory.
System requirements:
Windows 2000/XP/Server 2003
.NET Framework 2.0

New Features

- UNICODE support. All the functions that take char* arguments now have a second version that takes wchar_t*. DeinoMPI is implemented using wide characters and provides wrapper functions for ASCII char * strings. The name conversion is taken care of automatically with macros in mpi.h so all the user program has to do is define UNICODE and compile. These are the dual implementation functions (The C functions are listed and the C++ equivalents also have wide character versions):
  - MPI_Add_error_string
    MPI_Close_port
    MPI_Comm_accept
    MPI_Comm_connect
    MPI_Comm_get_name
    MPI_Comm_set_name
    MPI_Comm_spawn
    MPI_Comm_spawn_multiple
    MPI_Error_string
    MPI_File_delete
    MPI_File_get_view
    MPI_File_open
    MPI_File_set_view
    MPI_Get_processor_name
    MPI_Info_delete
    MPI_Info_get
    MPI_Info_get_nthkey
    MPI_Info_get_valuelen
    MPI_Info_set
    MPI_Init
    MPI_Init_thread
    MPI_Lookup_name
    MPI_Open_port
    MPI_Pack_external
    MPI_Pack_external_size
    MPI_Publish_name
    MPI_Register_datarep
    MPI_Type_get_name
    MPI_Type_set_name

MPI_Unpack_external
MPI_Unpublish_name
MPI_Win_get_name
MPI_Win_set_name

- Collective operations have been optimized for clusters of SMP machines. The collective operations have been optimized to minimize network traffic when multiple processes reside on each node. The new functions only affect MPI_COMM_WORLD but in the future this support will be extended to derived communicators. For the beta release this functionality has to be turned on by an environment variable. If you want to try it out add "-env MPICH_USE_SMP_OPTIMIZATIONS 1" to your mpiexec command line.

- A new startup mechanism and process manager. The DeinoPM (process manager) uses public and private keys to establish secure connections between machines in the cluster. All traffic between the process managers is encrypted. Each user controls their own keys (similar to the way they would for ssh).

- An abort callback function has been added. This allows for a function to be called asynchronously when a job is about to be aborted. The current implementation uses this function to write out logging buffers to disk before the process is killed. The subsequent log files contain more data as a result.

- Romio has been fully integrated into the source tree and obeys all the same coding practices as the rest of the code base. This means that MPI_File is allocated the same way MPI_Comm and other MPI objects are allocated. It also means the error handling and output is consistent across functions that take MPI_File, MPI_Comm or any other MPI object.

- The process manager can launch processes that have been compiled for either DeinoMPI or MPICH2 1.0.3. If you have MPICH2 1.0.3 from Argonne installed you can run these applications using mpiexec and DeinoMPIwin from DeinoMPI without modification. The process manager knows how to set up the environment for binaries compiled against MPICH2 1.0.3. This allows you to use one process manager to start all your DeinoMPI and MPICH2 jobs.

- Singleton init supports MPI-2 spawn functions. DeinoMPI allows single processes started without the process manager to call the spawn functions just as if it had been started by mpiexec. In other words, if you start your application like this, "mpiexec –n 1 myapp.exe", or like this, "myapp.exe", both applications can call MPI_Comm_spawn.

- DeinoMPI has a callback function that can print out the state of the MPI message queues and the recent MPI function call history of a running job. This can be useful for developers when a job hangs and they want to see what MPI messages the processes are waiting on and what MPI function calls they have made.

- DeinoMPI supports directory staging. DeinoMPI can copy a directory and its sub-directories out to the worker nodes before starting a job. Then after the job is finished any modified files in the directories can be copied back to the source. This can be useful to push out executables and data files for a job before it starts.

Installation

Download DeinoMPI.msi and run it on each machine in your cluster. The installation is identical on all machines and you must be an administrator to install Deino MPI on a machine. In addition to the software libraries and tools a process manager is installed as a Windows service. This action requires administrator privileges. Once the package is installed on the machines all users can use DeinoMPI even if they don't have administrator privileges.

Note on Windows Vista:
Currently on a Windows Vista machine the installer cannot start the Deino Process Manger service because the installer runs commands with limited privileges even when the user has Administrator privileges. So this means that you have to go to the Services management console after installing DeinoMPI and start the DeinoMPI Process Manager service on each node before Deino MPI jobs can be run (or you can reboot the machines and the service will start automatically). The management console can be started by right-clicking Computer from the Start menu and selecting "manage". Navigate to the DeinoPM service, right-click and select Start.

User setup

After you install the software on all the machines each user will need to create a Credential store. This store will be used to securely launch jobs. Mpiexec will not run a job without a Credential Store and the GUI tool will prompt the user to create a store the first time it is run. Users can use the DeinoMPIwin tool to create a Credential store or they can use the command line tool, create_credential_store.exe, to create the store.
      By default all users have access to all machines where DeinoMPI is installed and can launch jobs on any machines where they have user accounts. If you want to restrict user access to machines you can use the manage_public_keys.exe command line tool. Have an administrator run the following two commands on each machine that you want restricted access to:
      1) manage_public_keys.exe /auto_keys false
      2) manage_public_keys.exe /clear_all
At this point no users will have access to the machines. Then the administrator can add users one by one using the public key from each user's Credential store. Have each user export their public key from their Credential store using the GUI tool or the manage_public_keys.exe command line tool. The command line would be something like this:
      manage_public_keys.exe /export mypubkey.txt
Then send this file to or point the administrator to this file. The administrator can add access to the user using the following command:
      manage_public_keys.exe /import mypubkey.txt
This command is local to a machine and must be executed by a user with administrator privileges on the machine. Since the command is local it must be executed on each machine in the cluster that you want the user to have access to. Once the user's public key has been imported on one or more machines in the cluster then the user can use mpiexec or the GUI tool to launch jobs on those machines.

Tools

The following tools are installed:
- Mpiexe.exe – The MPI job launcher
- DeinoMPIwin.exe – The GUI interface which includes mpiexec job launching functionality and setup and management tools
- DeinoPM.exe – The process manager service
- Command line tools:
  - create_credential_store.exe
  - manage_credentials.exe
  - manage_public_keys.exe
- Jumpshot – A Java tool to view log files created when MPI jobs are profiled

Security

All communication used during process startup and management is encrypted. The public and private keys in each user's Credential store are used to establish secure connections between machines in the cluster. The keys are used to encrypt a session key which is then used to encrypt and decrypt session data. The private keys currently are 2048 bit and the session encryption uses the Rijndael (AES) algorithm.

Files

The following sub-directories are created in the location you choose to install DeinoMPI: bin, include, lib, examples and java. The bin directory contains the executable files mpiexec.exe and DeinoPM.exe and Jumpshot.jar and the command line tools. The include directory contains the header files required to compile MPI applications. The lib directory contains the libraries required to link MPI applications. The examples directory contains pre-compiled example applications and source code for each. Read the examples.pdf file for a description of all the examples. The java directory contains the manual for the Jumpshot log viewing application. The dlls used to implement DeinoMPI are installed in the Windows system directory. Two assemblies used are installed in the GAC (Global Assembly Cache), DeinoMPI.dll and DeinoLP.dll.

Compiling Applications

Compile your MPI applications using the following information. Set your include path to include the DeinoMPI\include directory. Set your library path to include the DeinoMPI\lib directory. Link your C applications with mpi.lib from the lib directory. Link C++ applications with the cxx.lib and mpi.lib libraries. Link Fortran applications with fmpich2.lib.

There are several versions of the Fortran link libraries to support common name mangling and calling conventions used by various Fortran compilers. The following libraries contain the specified formats:

- fmpich2.lib – MPI_INIT and mpi_init_. All capital external names and all lower case external names with a trailing underscore. Functions use the C calling convention. There is a Unicode version of this library also – fmpich2u.lib. The Intel compiler uses this format as of the 8.x series. The Portland Group compiler uses the lower case format.
- fmpich2s.lib – MPI_INIT@4. All capital external names. Functions use the stdcall calling convention. Visual Fortran 6.x uses this format.
- fmpich2g.lib – mpi_init__. All lower case external names with two trailing underscores. Functions use the C calling convention. g77 uses this format.

For examples using specific compilers see the Compiler support section.

The libraries have the same names as in the MPICH2 distribution so users with existing scripts can easily port them to DeinoMPI.

Running Applications

You can use the graphical DeinoMPIwin.exe or mpiexec.exe from the command line to start MPI jobs.

DeinoMPIWin.exe

This multi-purpose tool can be used to start MPI jobs, manage the user's Credential Store, search the local network for machines with DeinoMPI installed, verify the mpiexec inputs to diagnose common job problems, and view the DeinoMPI web page for information such as help pages on all the MPI functions.

Mpiexec tab

The main page is used to start and interact with MPI jobs.
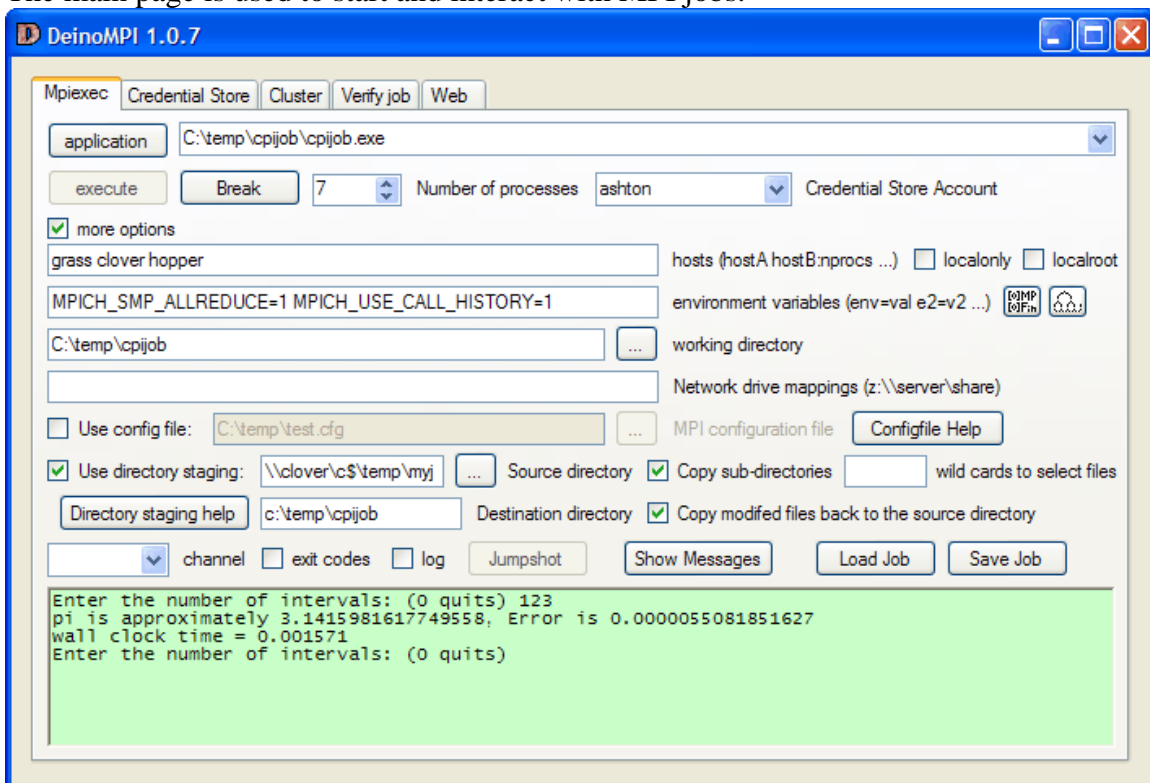
The mpiexec tab contains the following sections:

- Application button and text box
  - Enter the full path to your MPI application here. The path must be valid on all machines in the job. If you specify a local path like, C:\temp\myapp.exe, then you must copy myapp.exe to all the nodes in the job before executing the job. DeinoMPI does not push local files out to the nodes for you
  - If you enter a network path to an executable you need to make sure you have enough licenses on the server to host the executable. For example if the application is specified like this, \\myserver\myshare\myapp.exe, then you need to make sure that the host myserver has enough licenses to serve

myapp.exe to all the nodes in the cluster. Windows XP only allows 10 network connections to a file. If you plan on running jobs this way with more than 5 processes then you probably need to be running Windows Server on the myserver machine.
  - o Click the Application button to browse to the location of your executable.
- Execute button
  - o Click this button to start the job. The output area will turn green while the job is running and output will be displayed there. You can also type into the output area and the characters will be sent to the root process in the job each time you hit enter.
- Break button
  - o This button can be used to abort a job and kill all the processes.
- Number of processes spinner
  - o Use this control to enter how many processes you want to launch.
- Credential Store Account selection box
  - o Select the user credential to launch the job under. If your Credential store does not contain any user credentials then you will be prompted to enter a user account when you click the Execute button.
- More options check box
  - o This box expands or contracts the extra controls area.
- Hosts edit box
  - o Enter the hosts here where the job is to be launched. If this is left blank then the default hosts are used. If it is blank and there are no default hosts then the local host is used. Hosts are selected from this list in a round robin fashion from left to right. If you want more than one process to be deposited on a host before moving on to the next host you can specify this by adding a colon and a number to the host name. So if your list looked like this:
    - ▪ hostA hostB:2 hostC
  - o Then hosts would be selected like this: hostA hostB hostB hostC hostA hostB hostB hostC hostA …
- Localroot check box
  - o Check this box to have the root process in the job (process 0) launched directly from the current process bypassing the process manager. This means that the process will run under the same credentials as the current process which may or may not be the one selected from the Credential store. It also means that the process will have access to the console so it can do things like bring up windows and interact with the user.
- Localonly check box
  - o Check this box to cause all processes to be launched on the local machine. All host information will be ignored and all the processes will be launched locally.
- Environment variables edit box
  - o Enter a list of environment variables and their values here to be set in the environment of each process in the job before each is launched. The format is var=val var2=val2 …

- o If either the variable name or value has spaces in it then they must be quoted like this: "my name"="John Doe"
- Add call history check button
  - o This helper button adds or removes the environment variable used to control the MPI call history debugging option. With this environment variable set all MPI function calls are printed to an internal ring buffer while the job is running. Then you can click the Show Messages button and the call history will be printed out for each process. The default is to save 32 calls per process. If you want to reduce this number you can set the variable MPICH_CALL_HISTORY_SIZE=num_entries. You can set this value to a number between 1 and 32.
- Add SMP optimizations check button
  - o This helper button adds or removes the environment variable used to control the SMP optimizations for the collective operations. With this environment variable set the communication patterns for the collective operations are modified to minimize the network traffic on clusters of SMP machines. When jobs are run with multiple processes per host node the collective operations can be broken into two steps, a node phase and a network phase, thus minimizing the network traffic and speeding up the operations. As of the 1.0.7 release the optimized functions are MPI_Bcast, MPI_Barrier, MPI_Allreduce and MPI_Gather and they only apply to MPI_COMM_WORLD. To turn on all optimizations use the variable MPICH_USE_SMP_OPTIMIZATIONS=1. To turn on individual options use MPICH_SMP_BCAST=1, MPICH_SMP_BARRIER=1, MPICH_SMP_GATHER=1 or MPICH_SMP_ALLREDUCE=1.
- Working directory edit box and browse button
  - o Enter the path to the working directory to be set before each process is launched. Use the browse button to select a directory using a GUI.
- Network mapped drives edit box
  - o Drive mappings entered here will be mapped before each process is started and removed after the processes exit. The format is <drive>:<share path>. For multiple mappings separate them with semi-colons. The following example shows two mappings: z:\\myserver\myhome;y:\\myserver\mydata
- Channel selection box
  - o Select the MPICH2 channel to use for inter-process communication. The default is the sock channel and is valid for all cases.
  - o Sock – TCP/IP socket channel
    - ▪ This channel uses TCP/IP sockets to communicate between processes.
  - o Shm – shared memory channel
    - ▪ This channel can only be used on a single machine and only scales up to about 8 processes
  - o Sshm – scalable shared memory channel
    - ▪ This channel can only be used on a single machine
  - o Mt – mutli-threaded socket channel

- This channel is the same as the sock channel but the thread safe locks have been turned on so that MPI applications can be multi-threaded.
- Exitcodes check box
  - Select this option to print the exit codes of all the processes when the job exits.
- Log check box
  - Select this option to use the MPE wrappers to create a log file of the MPI job. When the job completes a clog2 file will be created in the same location as the MPI application and will have the same name as the application with .clog2 appended to the end. Then you can use Jumpshot to view this file. Jumpshot will convert the .clog2 file to a .slog2 file before viewing.
- Jumpshot button
  - If you have Java installed on your machine this button will launch the Jumpshot viewer and pass it the name of the .clog2 log file created from the last MPI job.
- Show Messages button
  - While a job is running this button is available. When you click it a message is sent to all the processes in the job and they reply with information about the state of their internal MPI message queues. This information is then appended to the output area. You can use this information to see what processes are waiting for MPI messages and a small amount of information about each message. The MPI function call history will also be printed for each process if the environment variable, MPICH_USE_CALL_HISTORY, is set.
- Load Job button
  - You can load job information into the dialog from a file previously saved using the Save Job button.
- Save Job button
  - You can save all the information entered in the dialog to a file.
- Use config file checkbox and input box
  - If you select the "Use config file" checkbox you can specify a standard MPI-2 configuration file to describe the job. When this option is selected all the other members of the dialog are disabled because all the job description comes from the specified configuration file. The format of the file is the same as the command line arguments to mpiexec. When specifying arguments to mpiexec sections are separated by the colon character. These sections are placed on independent lines in the configuration file and colons don't have special meaning. Blank lines are ignored and lines starting with the # character are considered comments. The following is an example of a simple configuration file that starts five processes:
  - #sample job
  - –n 1 –host HostA C:\temp\master.exe
 - –n 4 C:\temp\worker.exe

- Use Directory staging checkbox and associated input fields.
    - If you select the "Use directory staging" check box then you can enter the information necessary so you can stage a directory on all the nodes in your job. Enter the source directory where you want files to be copied from. Select the "Copy sub-directories" check box if you want sub directories under the source directory to be copied also. Enter the destination directory where you want the files copied to on all the nodes. If you leave this field blank then the destination will be the same as the source. You can limit which files will be copied by entering a wildcard specification. For example if you enter *.txt then only files that end in .txt will be copied (the default is all files *.*). The files and directories are copied out to the nodes before the job starts. After the job ends you can copy any modified files back to the source directory if you select the "Copy modified files …" checkbox. When files are copied back to the source they will overwrite files in the source directory. But if there are files created on the nodes with the same names they will be renamed when copied back to the source so they don't clobber each other.
    - For example: If you had a directory on a server with your executable and data files you could enter the directory in the source edit box like this: \\myserver\myshare\myjob1. Then you could enter a local location in the destination directory field like this: c:\temp\job1. Then you could enter into the application field the name of your job executable relative to the local directory like this: c:\temp\job1\myapp.exe. When you click execute to start the job the first thing that happens is the myjob1 directory is copied to all the nodes. Then the job will run and after the job finishes if you selected to copy the modified files then any modified or new files in the c:\temp\job1 directory will be copied back to the source directory, \\myserver\myshare\myjob1.
    - The source doesn't have to be on a server. You could enter a local directory to be copied to all the nodes like: c:\data\cfd\parallel\sim123.
- Output area
    - The output area displays output from the currently running job. It turns green while the job is active as a visual queue to indicate that the job is running. You can type into this area and the input will be forwarded to the root process in the job each time the enter key is pressed.

Credential Store tab

The Credential Store tab is used to manage the current user's Credential Store.
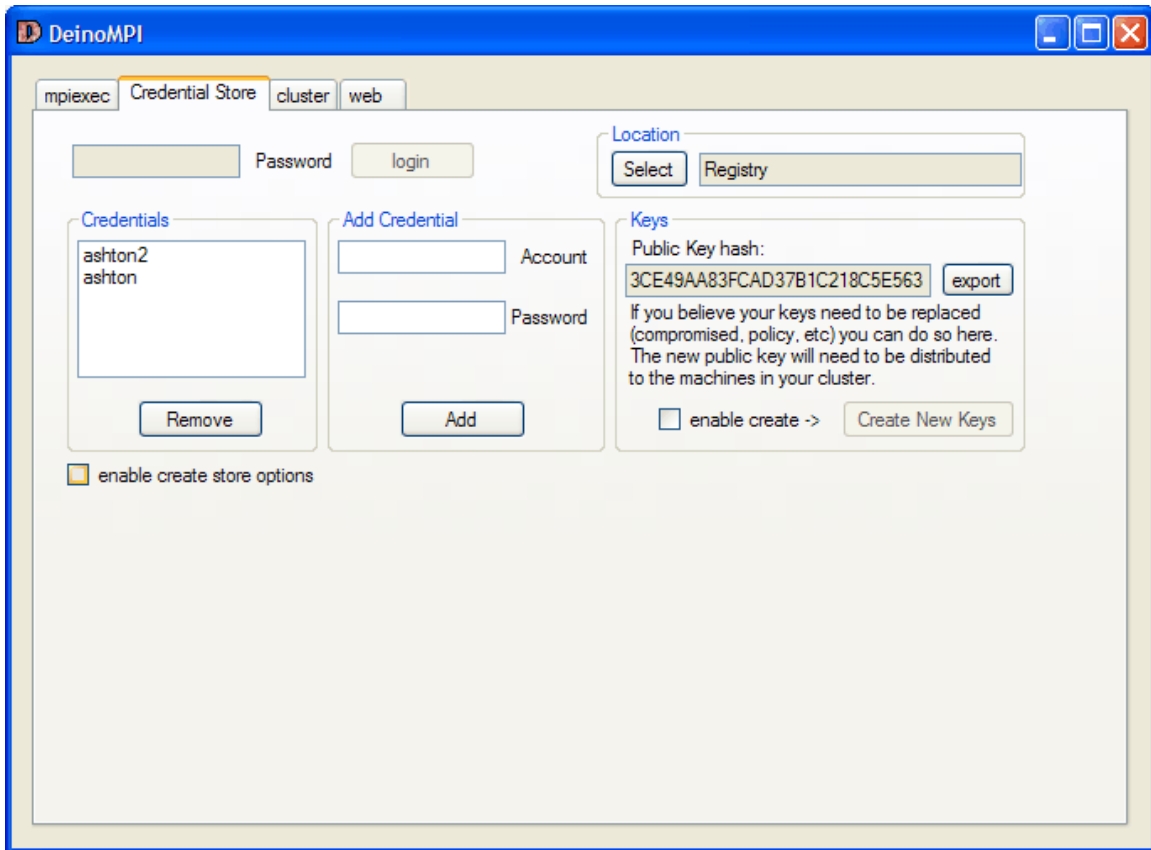
If you do not have a Credential Store created then select the "enable create store options" check box and more options will be available. These options are hidden by the check box because they are usually only needed once by the current user the first time DeinoMPI is run.
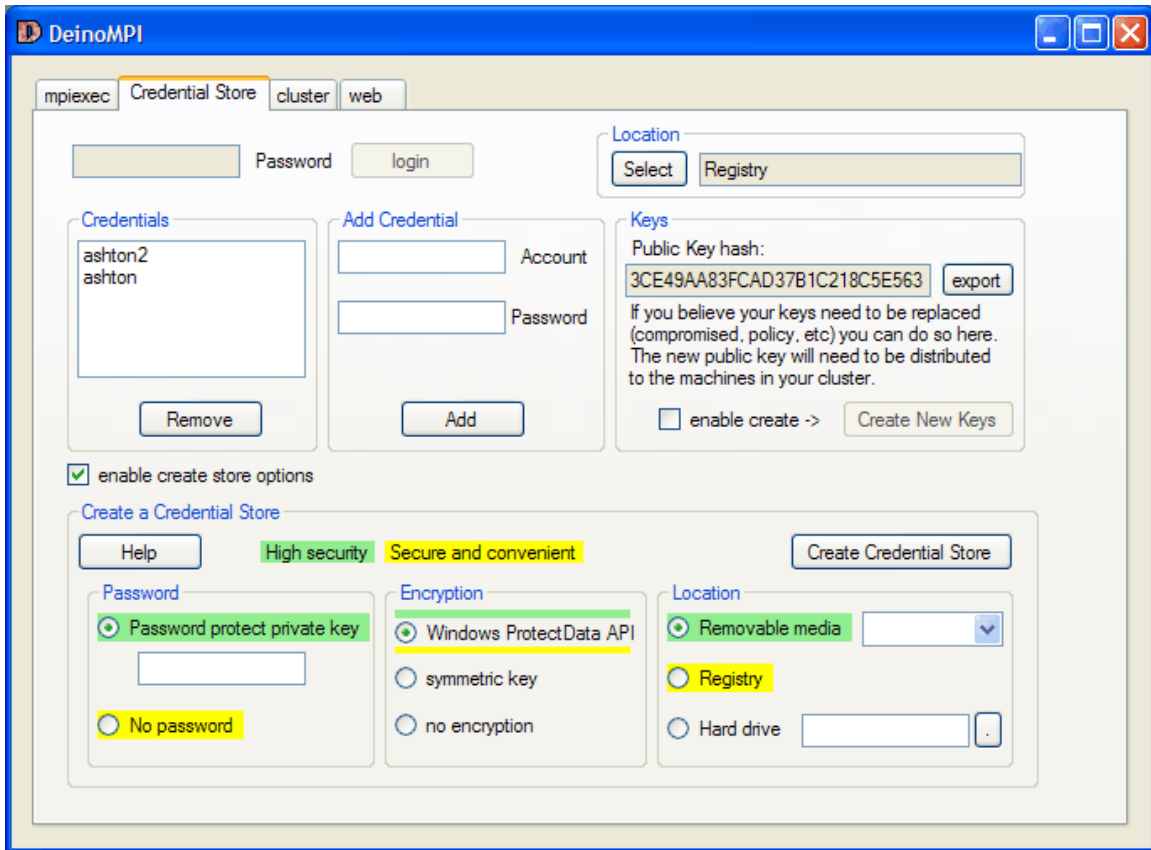
Creating a credential store

Click the "enable create store options" check box to enable the creation fields
Select your choice from the three sections and then click Create Credential Store
The three sections are

1. Password
   a. If you select a password then the Credential store can only be accessed by entering this password. This is the most secure option but it requires that you enter this password every time an MPI job is run. The password is not stored anywhere so you have to enter it for every job.
   b. Selecting no password protection makes using MPI easier to use but slightly more vulnerable. With no password, any program run by the current user can access the Credential store. This usually isn't a problem if you know you are not running any malicious software.
   c. Even with no password your Credential store is still not available to other users if you select encryption.
2. Encryption
   a. Select "Windows ProtectData API" to encrypt your Credential store using the current user's encryption scheme provided by Windows. This ensures that your Credential store can only be accessed by you when you are logged in.
   b. If you selected a password for your store then you can select the symmetric key encryption format. This encryption uses the password to

create a symmetric key to encrypt the Credential store. This encryption is not user specific so any user who knows the password can access the store.

    c. The "no encryption" option is not recommended. This option stores the data in the store in plain-text that anyone can access. With this option it is your responsibility to protect the Credential store file.

3. Location
    a. Select the Removable media option to save the store on a removable device like a USB thumb drive. If you select this option then MPI jobs can only be started by the current user when this drive is inserted in the machine. This can be a security enhancement since the user controls window of time when the Credential store is available. If this is combined with a password and the Windows ProtectData API encryption then the device is also quite safe in case it is lost or misplaced.
    b. You can use the hard drive option to save the store to a specific location on your machine. The browse button lets you graphically navigate to a directory. This option also allows you to navigate to any location including a removable media device.
    c. Select the Registry option if you want to save the store in the current user's Windows Registry hive.

Recommendations:

If you want the highest security option it is recommended that you select a password, the Windows ProtectData API and save the store to a removable USB thumb drive. Click the green High Security label to automatically select these settings.

If you want high security and convenience it is recommended that you select no password, the Windows ProtectData API and save the store to the Windows Registry. Click the yellow Secure and Convenient label to automatically select these settings.

Once you have created a Credential store you can use the rest of the fields in the Credential store tab to manage the store. You can add and remove user names and passwords from the store using the Add and Remove buttons.

If your store is password protected then you will need to enter the password into the password box and click login before you will have access to the store.

You can change the location of your Credential store using the location section. This is useful if you have created multiple stores. For example you may want to switch between a store saved to the Windows Registry and another store saved on a USB thumb drive.

The Keys section shows the hash of the public key in your store. You can export the public key to a file by clicking the export button. This file can then be imported into the PublicKey store on other machines in your cluster. This is required if you select the option to deny unknown public keys when connecting to machines in your cluster. The default behavior is to automatically distribute public keys when connecting to machines for the first time. If you disable the automatic option then you will need to import the public key of each user that is allowed to run jobs on a machine. Public keys can be imported into the local PublicKey store on each machine by an administrator using the manage_public_keys.exe command line tool.

You can also change the public and private keys in your store. Since this is an uncommon option it is protected by two clicks. First select the enable create check box to enable the button to create new keys. Then click the Create New Keys button and the keys will be replaced in your store with new ones. This option is provided in case your keys are compromised for some reason. You may also have security requirements that require you to change the keys at certain intervals. How you decide to manage your keys and Credential stores is up to you.

Cluster tab

The cluster tab is used to view the machines on your local network and see what version of DeinoMPI you have installed.
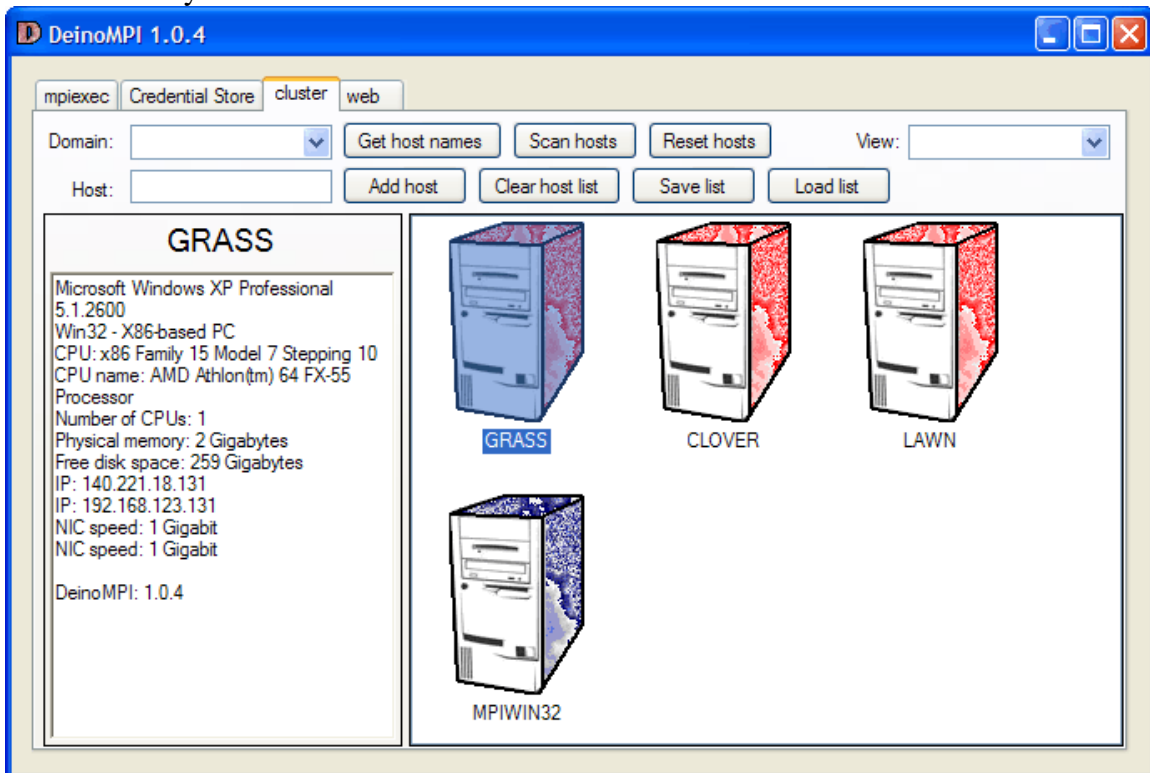

The cluster tab contains the following items:

- Domain dropdown input box. If your network is part of a domain you can input the domain name here and then click the get host names button to retrieve the host names from the domain controller. When you select the drop down arrow the tool queries the network for available domains and adds them to the list. This operation can take a while.
- Get host names button. After entering a domain name in the domain box click this button to query the domain for host names and add them to the host list.
- Scan hosts button. Click this button to query all the hosts in the host list to see if they have DeinoMPI installed. This operation takes a long time and processing goes on in a background thread. Hosts where DeinoMPI is installed turn red or blue. Blue hosts have DeinoMPI installed but it is a different version from the one installed on the local machine. The version installed must be the same on all

the nodes in order to run jobs so only red machines can participate in jobs. Hosts where DeinoMPI is not installed turn grey. If you add more hosts to the list and then click the scan button again, only the hosts that have not been previously scanned are scanned.

- Reset hosts button. Click this button to reset all the hosts in the list to an un-scanned state. After clicking the reset button, if you click the scan button all the hosts will be scanned.
- Host input text box. Use this box to enter host names.
- Add host button. After entering a host name into the host input text box click this button to add the host name to the list.
- Clear host list button. The clear button removes all hosts from the host list.
- Save list button. Click this button to save all the hosts in the list to a file. Any scanned information is also stored in the file.
- Load list button. Click this button to load a host list from a file that was previously created using the save list button. This is useful for large lists of hosts since the scan operation is quite time consuming and hosts usually don't change their configuration often.
- View dropdown selection box. Use this option box to determine how the hosts are presented in the host list area. The two options are Large icons and Details. The large icons option shows icons and host names. The details option shows the hosts in a list with small icons and columns for each piece of host information. Clicking the column headers will sort the list based on the selected column.
- Hosts list. The host list area shows all the hosts and any scanned information about the hosts in various views. Red hosts have DeinoMPI installed on them. Blue hosts have a version of DeinoMPI installed on them that does not match the version installed on the local host. Grey hosts do not have DeinoMPI installed and white hosts have not been scanned yet. When you click an individual host it will be scanned and the resulting information will be presented in the host info area. While a host is being scanned a question mark appears on the machine icon. If you select one or more hosts then a context menu becomes available using the right mouse button. Right click the selected hosts and you can perform any of the following actions:
  o Send the selected host names to the mpiexec tab. This action puts the selected hosts in the hosts text box of the mpiexec tab. Then when you launch a job these hosts will be used to host the processes.
  o Scan the selected hosts. This option will scan all the selected hosts. If the hosts have already been scanned they will be reset and scanned again.
  o Remove the selected hosts. This option will remove all the selected hosts from the host list.
  o Reset the selected hosts. This option will remove any scanned information and return the selected hosts to a pre-scan unknown state.
- Host info area. When an individual host is selected the information about that host is presented in the host info area. If the host has DeinoMPI installed on it then there will be a list of resources available on the host presented here. If the version of DeinoMPI installed on the host does not match the version installed locally the tool will not be able to retrieve any information about the host other
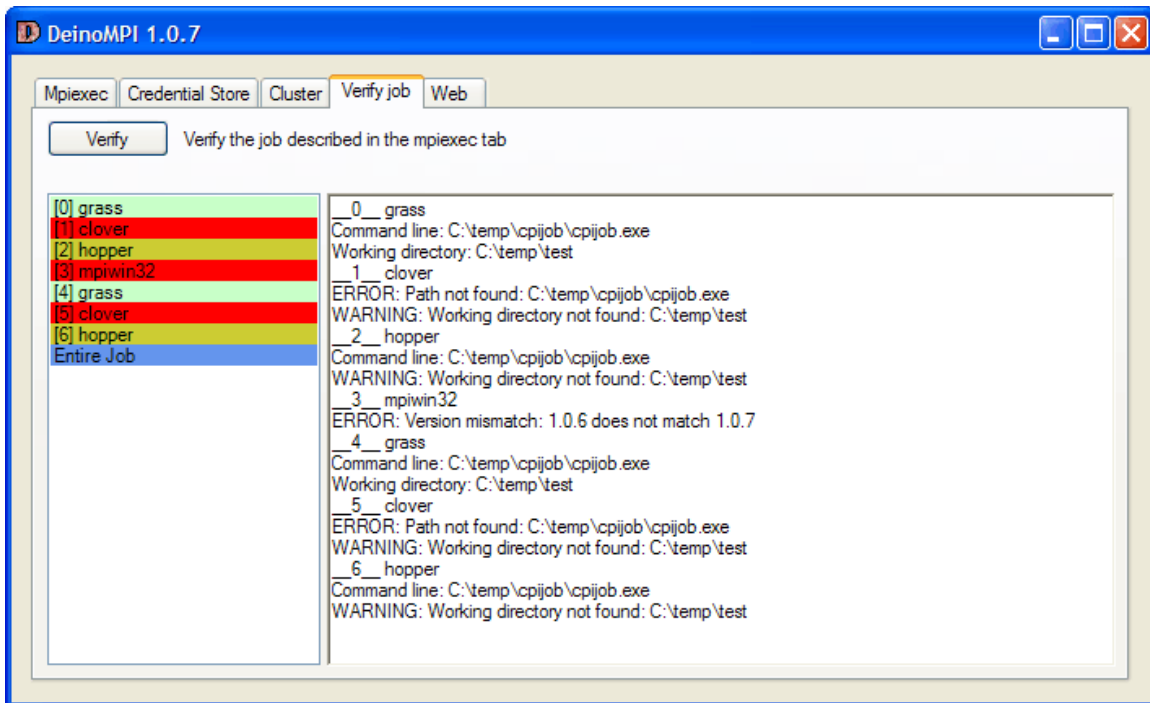
than the DeinoMPI version string. If DeinoMPI is not installed on the host then
only the host name will be displayed and a message indicating that DeinoMPI is
not installed.
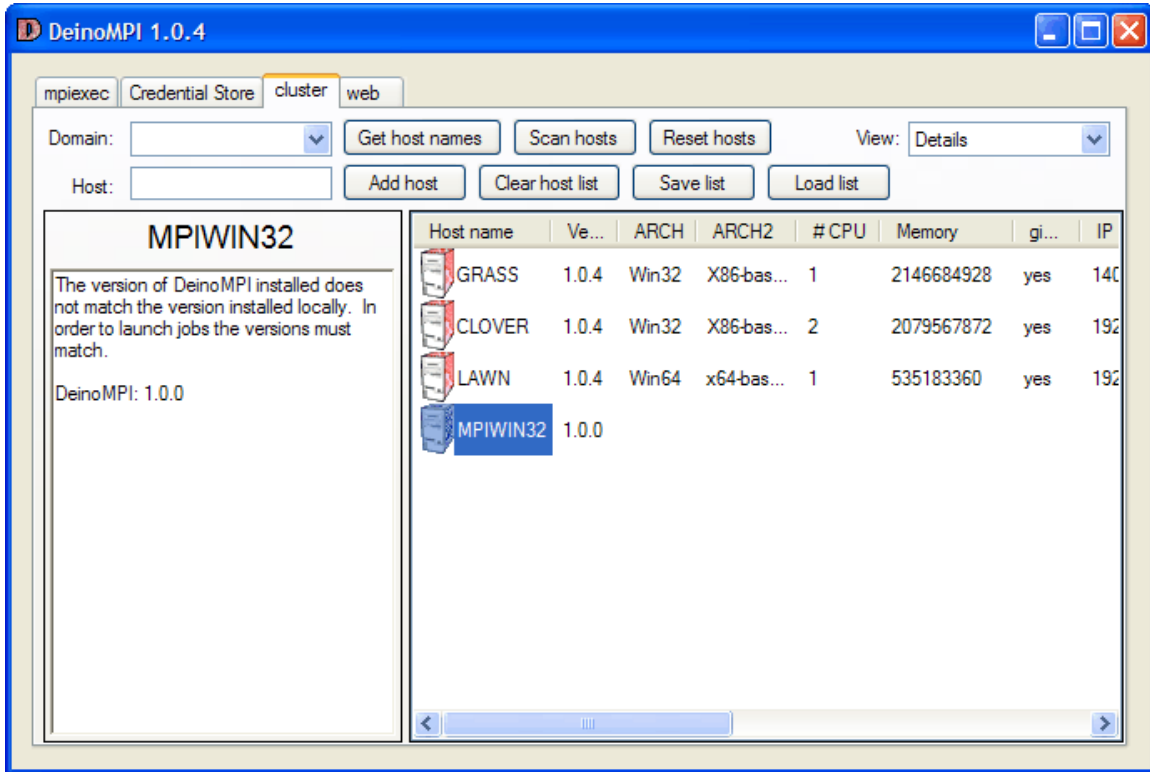
Verify Job tab

The Verify Job tab is a diagnostic tool to check to make sure the information provided in
the Mpiexec tab describes a valid job that will run on your cluster. It does not run the job
but it parses all the information, contacts the hosts and verifies that the job will be able to
run. It can find the following problems with a job:

- Missing executables.
- Bad user name or password.
- Missing dynamic link libraries.
- Invalid host names.
- Hosts where DeinoMPI is not installed or the versions don't match.

```
D DeinoMPI 1.0.7                                                        _ □ X

  Mpiexec   Credential Store   Cluster   Verify job   Web

     Verify        Verify the job described in the mpiexec tab

  [0] grass                    __0__ grass
  [1] clover                   Command line: C:\temp\cpijob\cpijob.exe
  [2] hopper                   Working directory: C:\temp\test
  [3] mpiwin32                 __1__ clover
  [4] grass                    ERROR: Path not found: C:\temp\cpijob\cpijob.exe
  [5] clover                   WARNING: Working directory not found: C:\temp\test
  [6] hopper                   __2__ hopper
  Entire Job                   Command line: C:\temp\cpijob\cpijob.exe
                               WARNING: Working directory not found: C:\temp\test
                               __3__ mpiwin32
                               ERROR: Version mismatch: 1.0.6 does not match 1.0.7
                               __4__ grass
                               Command line: C:\temp\cpijob\cpijob.exe
                               Working directory: C:\temp\test
                               __5__ clover
                               ERROR: Path not found: C:\temp\cpijob\cpijob.exe
                               WARNING: Working directory not found: C:\temp\test
                               __6__ hopper
                               Command line: C:\temp\cpijob\cpijob.exe
                               WARNING: Working directory not found: C:\temp\test
```

When you click the Verify button the information in the Mpiexec tab is parsed as if the job were to be run and then the hosts are contacted and information is collected. The results for each process in the job are provided in a list with the last entry used to display all the information for all processes. The processes are color coded to quickly ascertain the results. Green processes will run without any problems. Red processes will not run due to one of the problems mentioned in the above bulleted list. Yellow processes will probably run but have warnings. For example if a working directory does not exist on a host a warning will be displayed but the process will still run. If a delay loaded dll dependency is detected in the executable and that dll is not found on a host then a warning is displayed because the process may fail at runtime. This is a warning and not an error because many applications have libraries that load dlls based on the capabilities of the system they run on and do not fail when a dll is missing.

There are some problems that can cause a job to fail when run that the Verify Job tab is not able to check. If the process explicitly loads a dll by name using the LoadLibrary function call and this dll does not exist this error will not be detected. If the application tries to open a file that does not exist on a host this error will not be detected. If there is a fault in the application that causes it to crash or if the architecture doesn't match these errors will not be detected. There may be other run-time errors that can occur that this tool does not detect. It is provided as a tool to diagnose the most common errors that can be detected without running the job.

If the "Use directory staging" check box from the Mpiexec tab is selected a warning is displayed under the Verify button. If directory staging is selected then there may be files required by the job that will be copied out to the nodes before the job starts. The verify command does not copy any files so it may display errors even though the job may run just fine after the directories are copied to the nodes.

Web tab

The last tab is the web tab and it shows the web page for DeinoMPI. You must be connected to the internet to view this tab.
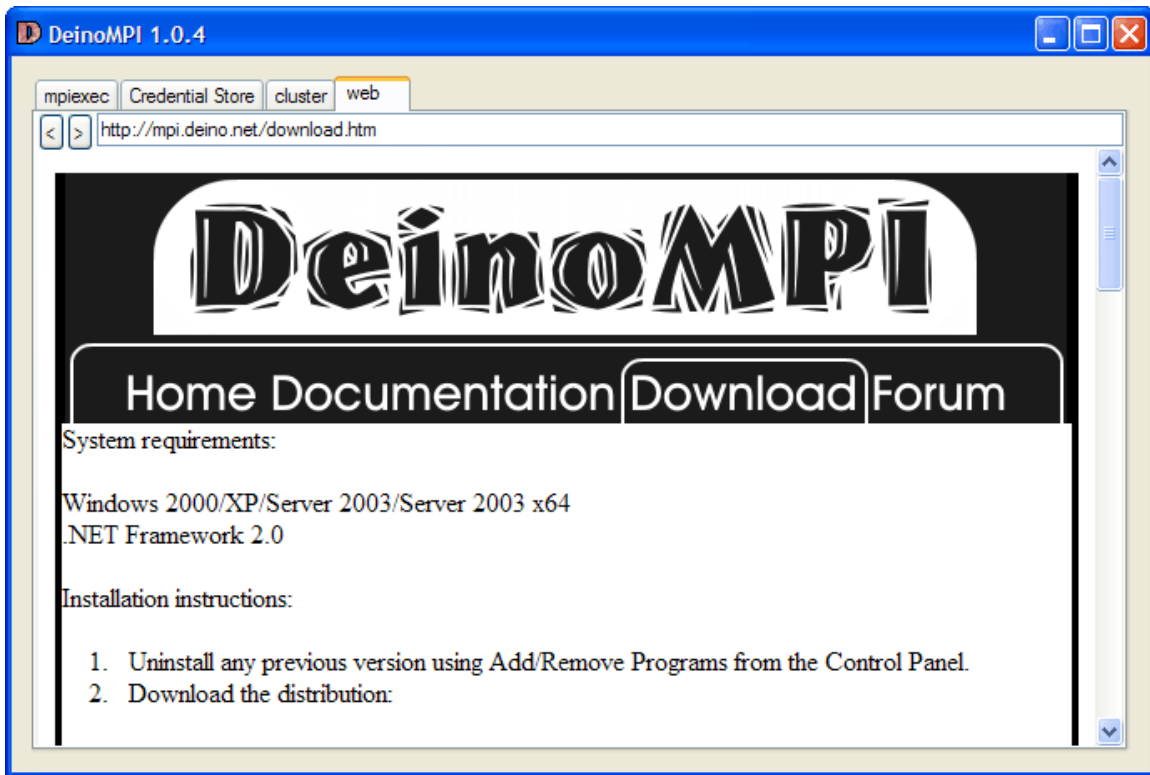
Mpiexe.exe

This command line tool can be used to start MPI jobs.

Usage

mpiexec -n <maxprocs> [options] executable [args ...]
mpiexec [options] executable [args ...] : [options] exe [args] : ...
mpiexec -configfile <configfile>

Examples

mpiexec -n 4 cpi
mpiexec -n 1 -host foo master : -n 8 worker

Standard options

-n <maxprocs>
-wdir <working directory>
-configfile <filename> -
     each line contains a complete set of mpiexec options including the executable and
     arguments
-host <hostname>
-path <search path for executable, ; separated>

All options

-n x
-np x
     Launch x processes.
-localonly x
-n x -localonly
     Launch x processes on the local machine.
-machinefile filename
     Use a file to list the names of machines to launch on. Each line in the file should
     contain one host name. Lines beginning with # are ignored. If you want multiple
     processes to be deposited on a single host you can add a colon and a number to
     the host name like this: hostA:4
-host hostname
-hosts n host1 host2 ... hostn
-hosts n host1 m1 host2 m2 ... hostn mn
     Launch on the specified hosts.
     In the second version the number of processes = m1 + m2 + ... + mn.
     Example: mpiexec –hosts 3 hostA hostB 2 hostC myapp.exe
     The example would produce 4 processes, 1 on hostA, 2 on hostB and 1 on hostC.
-map drive:\\host\share
     Map a drive on all the nodes. This mapping will be removed when the processes
     exit. Example: mpiexec –map z:\\myserver\myhome –n 4 z:\myapp.exe
-dir drive:\my\working\directory
-wdir drive:\my\working\directory

Launch processes in the specified directory. –wdir and –dir are synonyms.
Example: mpiexec –dir c:\temp –n 4 myapp.exe

-env var val
Set environment variable before launching the processes.
Example: mpiexec –env VERBOSE 1 –n 4 myapp.exe

-logon
Prompt for user account and password.

-pwdfile filename
Read the account and password from the file specified. Put the account on the
first line and the password on the second.

-user <user name>
Select the username to be used from the Credential store. With this option you
can specify different users for different hosts. This is useful if you have different
user accounts and passwords for different machines. The user name must match
an account saved in the current user's Credential store. If you have the same user
name on different machines but different passwords you will have to specify the
machine name in the user account name like this: hostA\userA and hostB\userA.
Example: mpiexec –user joe –host Foo –n 2 myapp.exe : -user john –host Bar –n
3 myapp.exe

-exitcodes
Print the process exit codes when each process exits.

-noprompt
Prevent mpiexec from prompting for user credentials.

-priority class[:level]
Set the process startup priority class and optionally level.
class = 0,1,2,3,4   = idle, below, normal, above, high
level = 0,1,2,3,4,5 = idle, lowest, below, normal, above, highest
The default is -priority 2:3.

-localroot
Launch the root process directly from mpiexec if the host is local. (This allows
the root process to create windows and be debugged.)

-path search_path
Search path for executable. Separate paths with the ; character.
Example: mpiexec –path c:\bin;c:\temp –n 4 myapp.exe

-timeout seconds
Specify a timeout for the job in seconds. If the job takes longer than the specified
number of seconds mpiexec will kill the processes.
Example: mpiexec –timeout 180 –n 4 myapp.exe

-set_hosts hostA hostB:N hostC ...
Set the default hosts and optionally the number of processes per host. If you want
to specify the number of processes to be deposited on a host put a colon at the end
of the host name and then put the number, like this: foo.host.com:2. This is useful
for multi-CPU machines where you want processes deposited on hosts one per
CPU.

-set_myhosts hostA hostB:N hostC …

Set the default hosts for the current user only. Use the same format as the –set_hosts option.

-log

Use the MPE logging library to log the MPI job. A clog2 file is produced after the job finishes which can be converted and viewed using Jumpshot (requires Java to be installed on the host).

-validate

Validate that the job can run without actually starting the job. Add this flag along with all the rest of the mpiexec command line. The hosts are contacted to determine if the job can run but executables are not started.
Example: mpiexec –validate –n 4 cpi.exe

-stage path

Specify the source directory you want to copy out to all the nodes before the job is started.
Example: -stage c:\data\myjob or -stage \\myserver\myshare\data

-stage_dest path

Specify the destination directory where you want the staged directory to be copied to. If this option is not specified then the source directory is used as the destination.
Example: mpiexec -stage \\myserver\myshare\data -stage_dest c:\temp\data ...

-stage_recursive

Specify this flag to indicate that you want the source directory to be copied and all of its sub-directories also.

-stage_wildcards pattern

Specify a wild-card pattern to limit which files are copied out to the worker nodes when staging a directory. Only files that match this pattern will be copied.
Example: mpiexec -stage_wildcards *.txt -stage c:\temp\data …

-stage_nopull

Specify this flag to indicate that you do not want any modified files copied back to the source directory after the job has finished. The default behavior is for new or modified files in the staging destination directory to be copied back to the source directory after the job has completed. This flag overrides that option.

Debugging Applications

Debugging parallel applications is hard but there are a few options provided by DeinoMPI that can help.

This simplest and sometimes most effective way to debug MPI applications is to add printf statements to your code. Whenever you add a printf statement, make sure to add a call to fflush(stdout); after the printf statement or statements. It is important to do so because the output of applications is buffered by default and must be flushed if you want to see the output immediately. This is true for stderr also because unlike on UNIX systems stderr is buffered on Windows machines.

The DeinoMPIwin.exe GUI tool has a Show Messages button that can provide useful information.

First, the Show Messages button will display the current state of the internal MPI message queues. This can be useful if your application hangs and you are not sure why. You can click the Show Messages button and see what messages are in the internal message queues for each process. Two message types can show up. There can be posted messages where a process is expecting a message but has not received it yet. And there can be messages that have been received and buffered but not matched yet. Note: some messages may not correspond directly to user code MPI calls since the implementation uses internal messages to implement the collective, file and win MPI functions. But it is easy to understand these messages because they are marked with type information.

Second, the Show Messages button will display the MPI function call history for each process if you have specified the MPICH_USE_CALL_HISTORY environment variable. With this option turned on each MPI function call is logged to an internal ring buffer. When you click the Show Messages button the last N calls will be printed out for each process. The default is to save 32 calls per process but this can be reduced using the environment variable, MPICH_CALL_HISTORY_SIZE. Set this value to a number between 1 and 32 to limit the depth of MPI calls per process saved.

Jumpshot is a java tool that can be used to profile applications. It is limited in its debugging ability though because applications must run to completion in order to generate log files. The primary goal of Jumpshot is to view the runtime patterns of an MPI application and then analyze these patterns to see if changes can be made to the code to optimize the application. Jumpshot has very well developed tool options for this kind of work. See the Jumpshot manual for information.

Command line tools

The following command line tools are provided to support DeinoMPI. create_credential_store.exe and manage_credentials.exe provide command line options equivalent to those found in the GUI DeinoMPIwin.exe tool. manage_public_keys.exe is an administrative tool that provides additional functionality not replicated in the GUI tool.

create_credential_store.exe

This tool creates a Credential store for the current user. Run it with no parameters and it will prompt you for the information necessary to create a store.

1. First it displays an introductory message and asks if you want to create a Credential store. If you specify yes it continues otherwise it exits.
2. Then it prompts for a password to protect the private key in the Credential store. If you do not enter anything by simply hitting enter it prompts you to confirm that you do not want the key to be password protected.
3. Then it prompts you for the type of encryption to protect the keys with. If you select "none" the keys will not be encrypted. If you select "protect" it will use the Windows API protect data functions to encrypt the keys. This type of encryption is user specific and can only be decrypted by the current user. Stores protected this way cannot be used by other users even if they know the password protecting the store. If you entered a password in step 2 then a third option is available: symmetric encryption. This option uses the password to create a symmetric key to encrypt the store. This type of encryption can be decrypted by any user who knows the password.
4. Then it prompts for the location to save the Credential store. The options are to save it in the Windows registry for the current user, on a removable device like a USB drive, or on the hard drive.
5. Then the Credential store is created and afterwards the user is ready to use manage_credentials.exe to add user credentials to the store and mpiexec to start MPI jobs.

manage_credentials.exe

This tool manages user credentials stored in the current user's Credential store. Run it with no parameters and it will output a usage message briefly explaining the options.

- manage_credentials /add [username] [password]
  - o Add a credential to the store.
  - o If you do not enter a username and or password you will be prompted to enter these parameters. If a credential with the same username already exists in the store it will be replaced.
- manage_credentials /remove [username]
  - o Remove the credential from the store.
- manage_credentials /list
  - o List all the usernames in the store.
- manage_credentials /replace_keys
  - o Create a new private and public key and re-encrypt all the existing credentials in the store.
- manage_credentials /set_location [path or "registry"]
  - o Point the current user to the Credential store specified.
  - o Provide either the full path to the Credential store file or registry to refer to the Windows Registry.

manage_public_keys.exe

This tool manages the Credential store public keys for the current user and the current machine. Some options to this tool require administrator privileges to execute. Run it with no parameters and it will output a usage message briefly explaining the options.

User level options:
- manage_public_keys /export <filename>
  - Export the public key from the current user's Credential Store and save it to the specified file.
- manage_public_keys /export <filename> <CredentialStore filename>
  - Export the public key from the specified Credential Store and save it to the specified file.

Administrator level options:
- manage_public_keys /import <filename>
  - Import the public key from the specified file and save it in the PublicKey store for the local machine.
- manage_public_keys /list
  - List the public keys in the PublicKey store for the local machine.
- manage_public_keys /remove <public key hash>
  - Remove the public key in the PublicKey store for the local machine that corresponds to the specified key hash
- manage_public_keys /clear_all
  - Remove all public keys in the PublicKey store for the local machine
- manage_public_keys /auto_keys <true or false>
  - Set the policy on the local machine to either accept new public keys from incoming connections automatically or reject them. If auto_keys is set to false then all user keys must be individually imported into the PublicKey store using the /import option.

Support

Send bug reports and other correspondence to support@deino.net

Compiler support

This section describes in detail how to compile MPI applications for DeinoMPI using various compilers.

Developer Studio .NET 2003

For Developer Studio .NET 2003 or newer you can use the example projects provided in the examples directory as a guide to creating your own projects.

1. Create a project and add your source files. It is recommended that you select a command line application but if you choose a GUI application you should make sure it can run unattended and that only the root process creates windows.
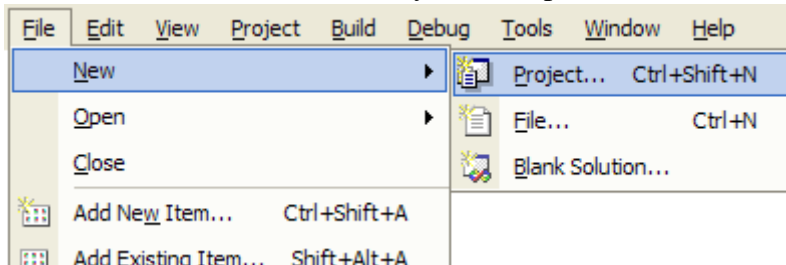
2. Bring up the properties dialog for your project by right clicking the project name and selecting Properties.
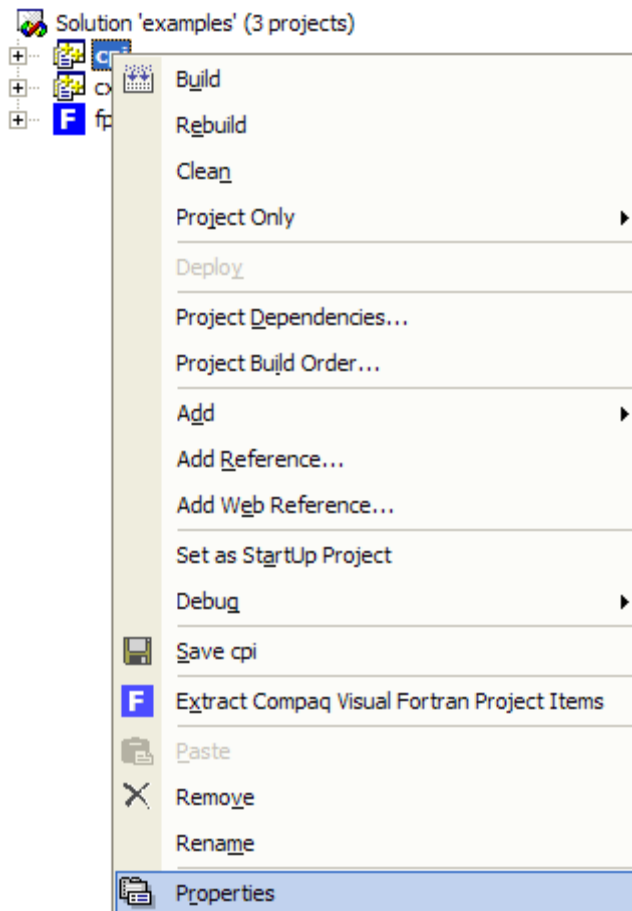
3. Navigate to Configuration Properties::C/C++::General
4. Add C:\Program Files\DeinoMPI\include to the "Additional Include Directories" box.
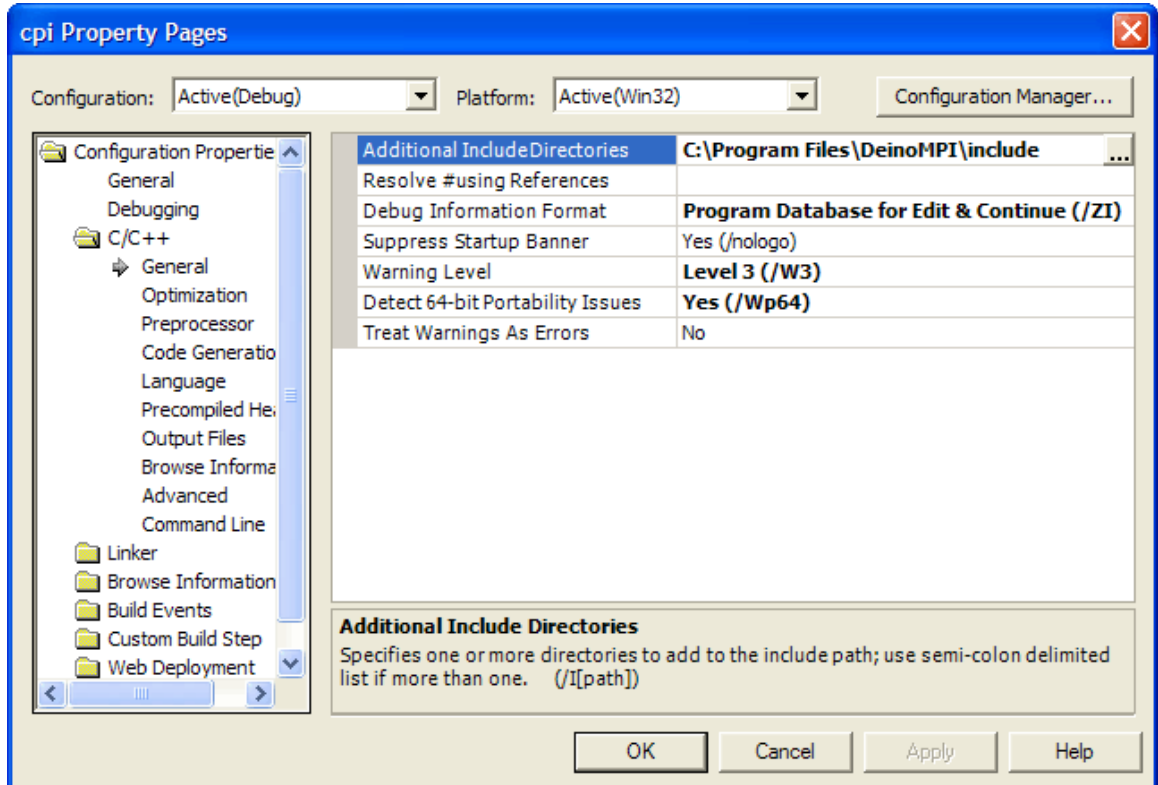
5. Navigate to Configuration Properties::Linker::General

6. Add C:\Program Files\DeinoMPI\lib to the "Aditional Library Directories" box.
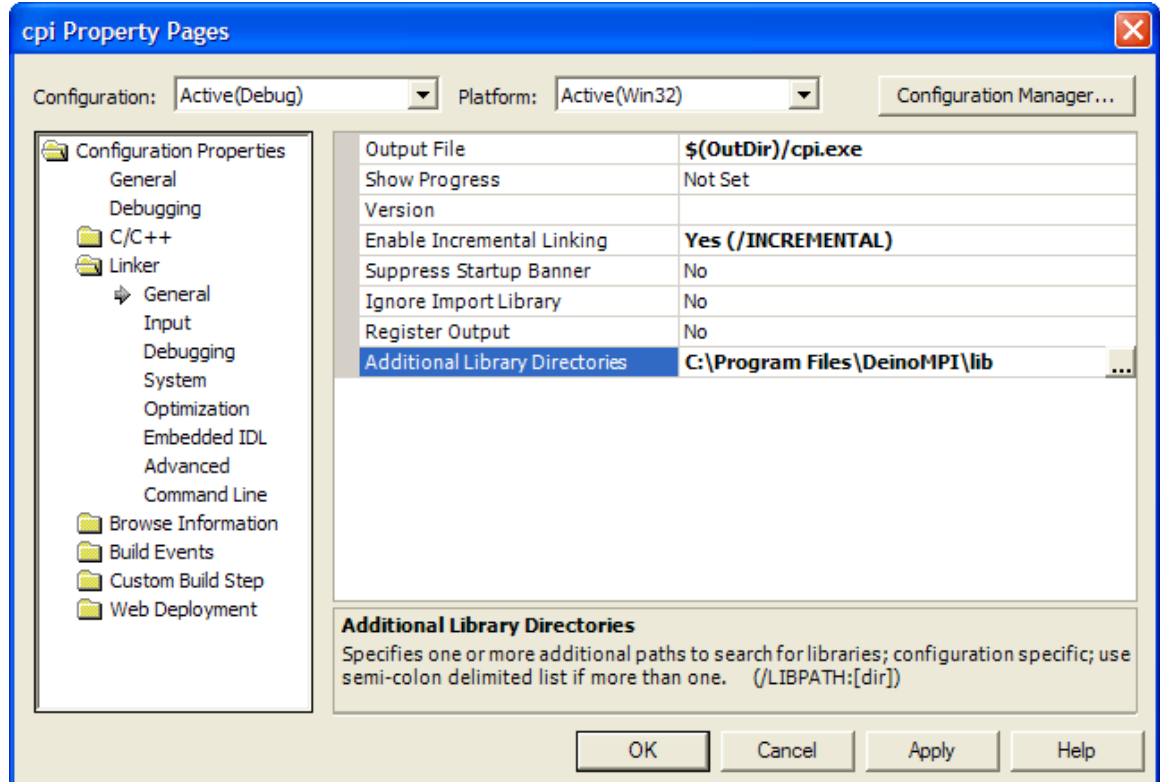
7. Navigate to Configuration Properties::Linker::Input
8. Add cxx.lib and mpi.lib and fmpich2.lib to the "Additional Dependencies" box. If your application is a C application then it only needs mpi.lib. If it is a C++ application then it needs both cxx.lib and mpi.lib. If it is a Fortran application then it only needs one of the fmpich2[s,g].lib libraries. The fortran library comes in several flavors namely fmpich2.lib, fmpich2u.lib, fmpich2s.lib and fmpich2g.lib. Add the library that matches your Fortran compiler.
    a. fmpich2.lib contains two sets of symbols. One set is all uppercase symbols and the other is all lowercase with one trailing underscore. Both interfaces use the C calling convention. The symbols look like this: MPI_INIT or mpi_init_. The Intel compiler uses this format as of the 8.x series. The Portland Group compiler uses the lower case format.
    b. fmpich2u.lib is a Unicode version of fmpich2.lib
    c. fmpich2s.lib contains all uppercase symbols and uses the stdcall calling convention like this: MPI_INIT@4. Visual Fortran 6.x uses this format and so does the Intel compiler when combined with the –cvf flag.
    d. fmpich2g.lib contains all lowercase symbols with double underscores and the C calling convention like this: mpi_init__. g77 uses this

format.
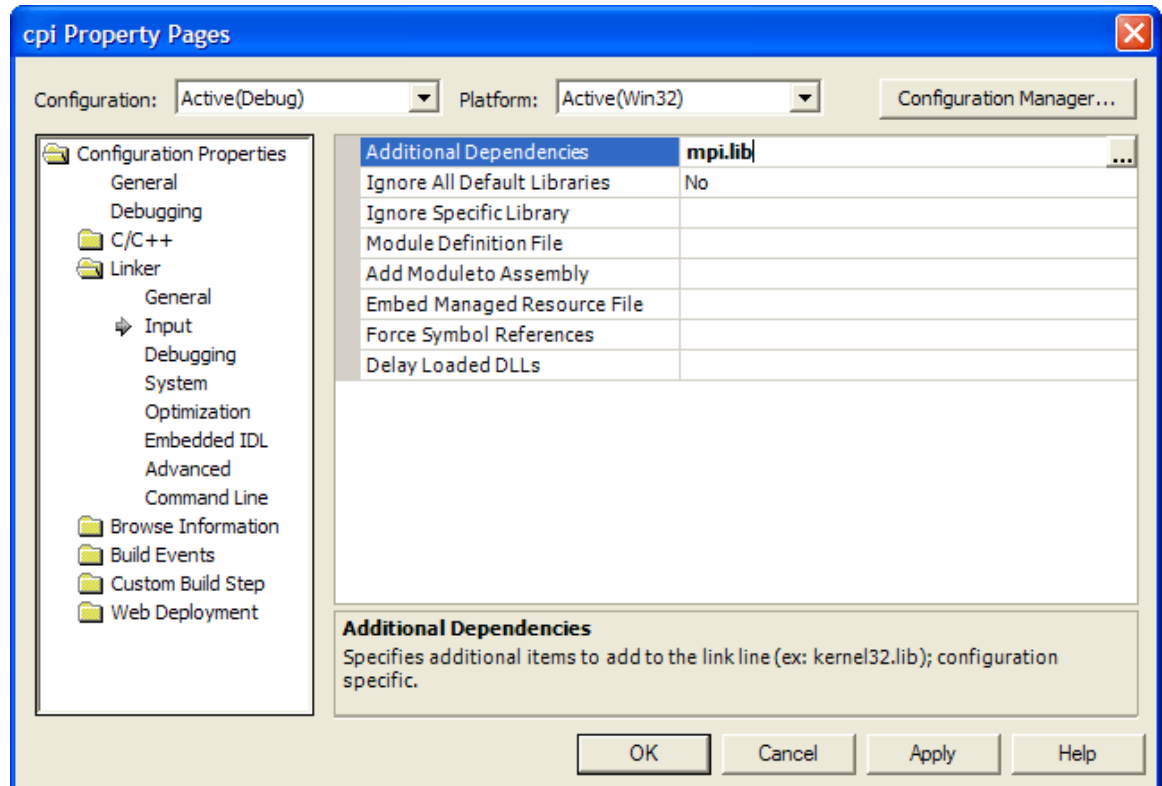
9. Compile your application.

Developer Studio 2005

For creating C/C++ MPI applications using Developer Studio 2005 the instructions are the same as in the section for Developer Studio .NET 2003. The dialog boxes are identical for setting the include and library paths. There are a couple things to note if you choose to create a console application using the default wizard settings. The wizard creates an application using UNICODE and pre-compiled headers. You will need to add "include <mpi.h>" to the generated stdafx.h file instead of your source file otherwise you will get unresolved symbol errors. Also the include of mpi.h must go before stdio.h because of the known conflict between the MPI standard and the standard include files.

Intel C

The Intel compiler can be integrated into Developer Studio .NET 2003 and it accepts the same command line arguments as the Microsoft Visual C/C++ compiler. So if you have the Intel compiler integrated into the developer studio suite then you can follow the instructions from the Developer Studio section to create DeinoMPI applications.

Intel Fortran

The Intel compiler can be integrated into Developer Studio .NET 2003. So if you have the Intel compiler integrated into the developer studio suite then you can follow the instructions from the Developer Studio section to create DeinoMPI applications. Set up the include and library directories as described in the Developer Studio section. For the
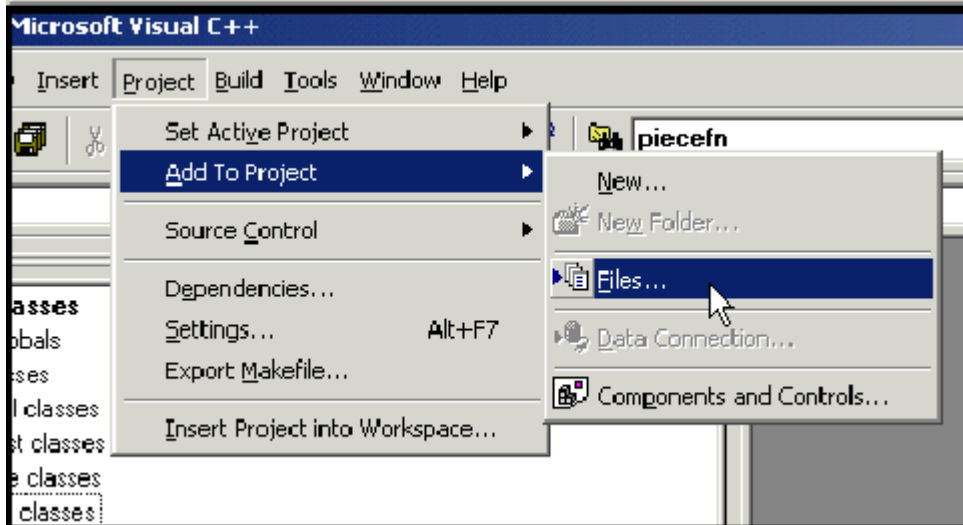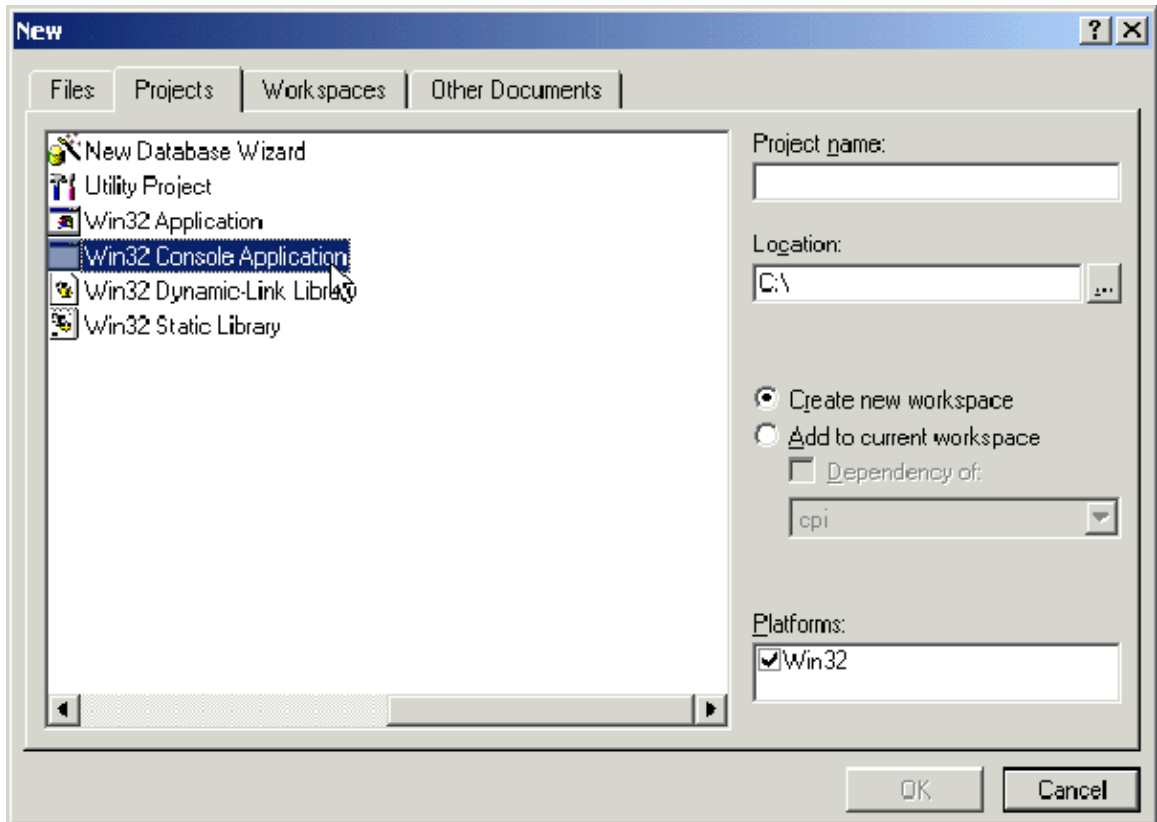
default Intel Fortran settings use fmpich2.lib in the link dialog. If you use the –cvf option then link with the fmpich2s.lib library. The one difference between the instructions for Visual Studio C/C++ applications and Intel Fortran applications is the location of the include directory setting. For Fortran applications it is located under the fortran::general folder:
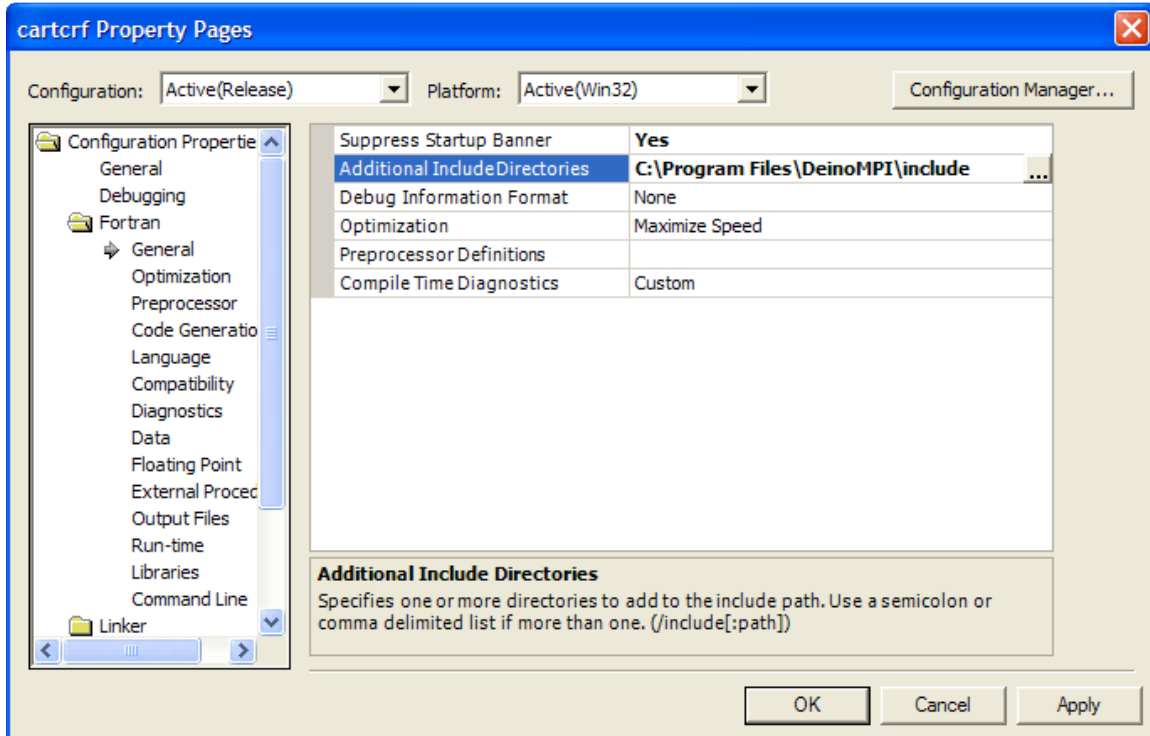

Visual Studio 6.0

Visual C++ 6.0 cannot handle multiple functions with the same type signature that only differ in their return type. So you must define HAVE_NO_VARIABLE_RETURN_TYPE_SUPPORT in your project.
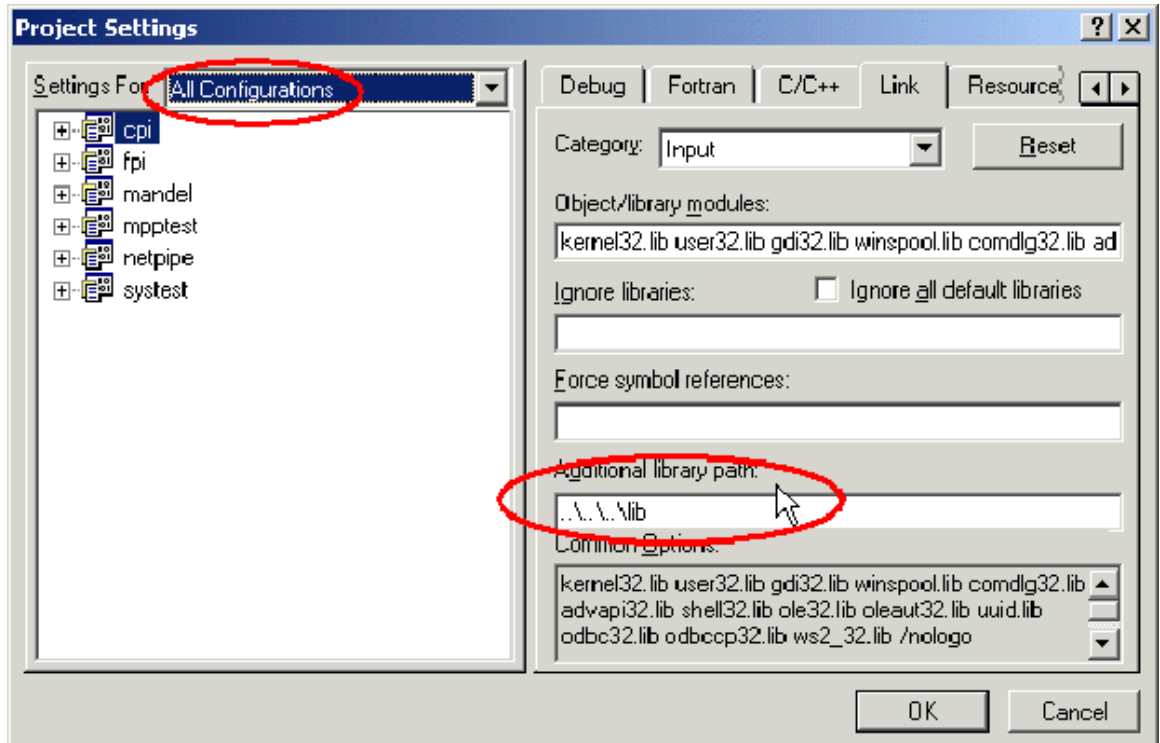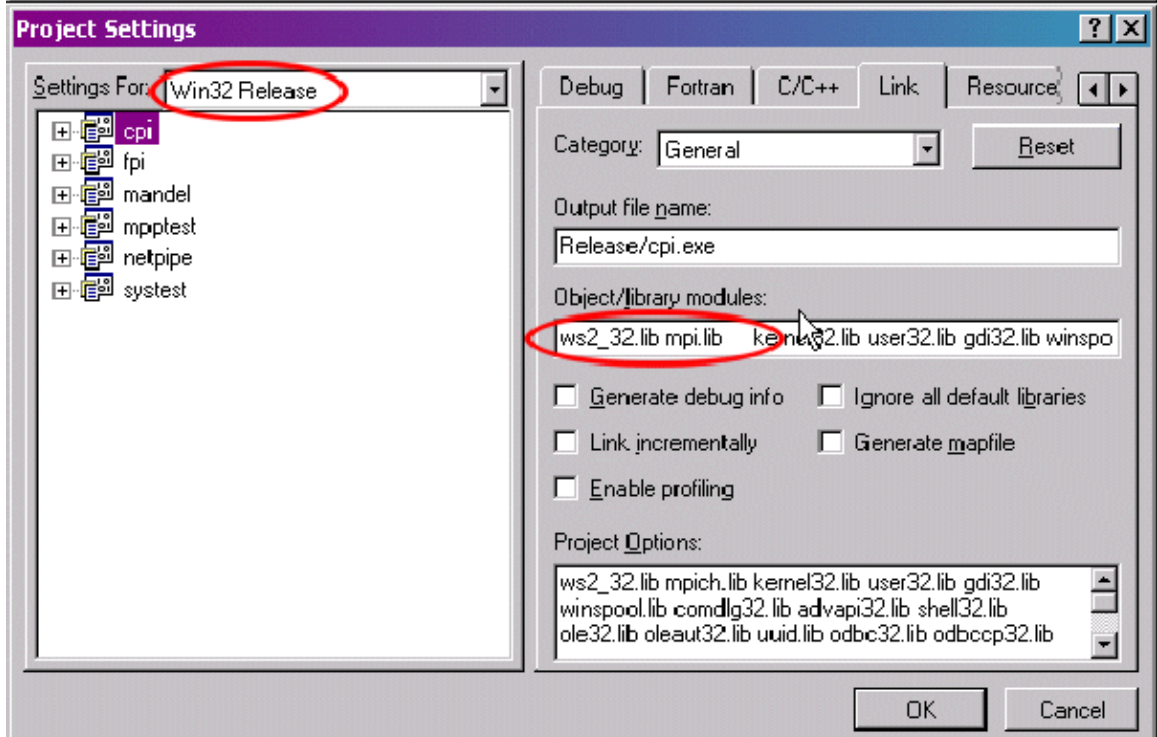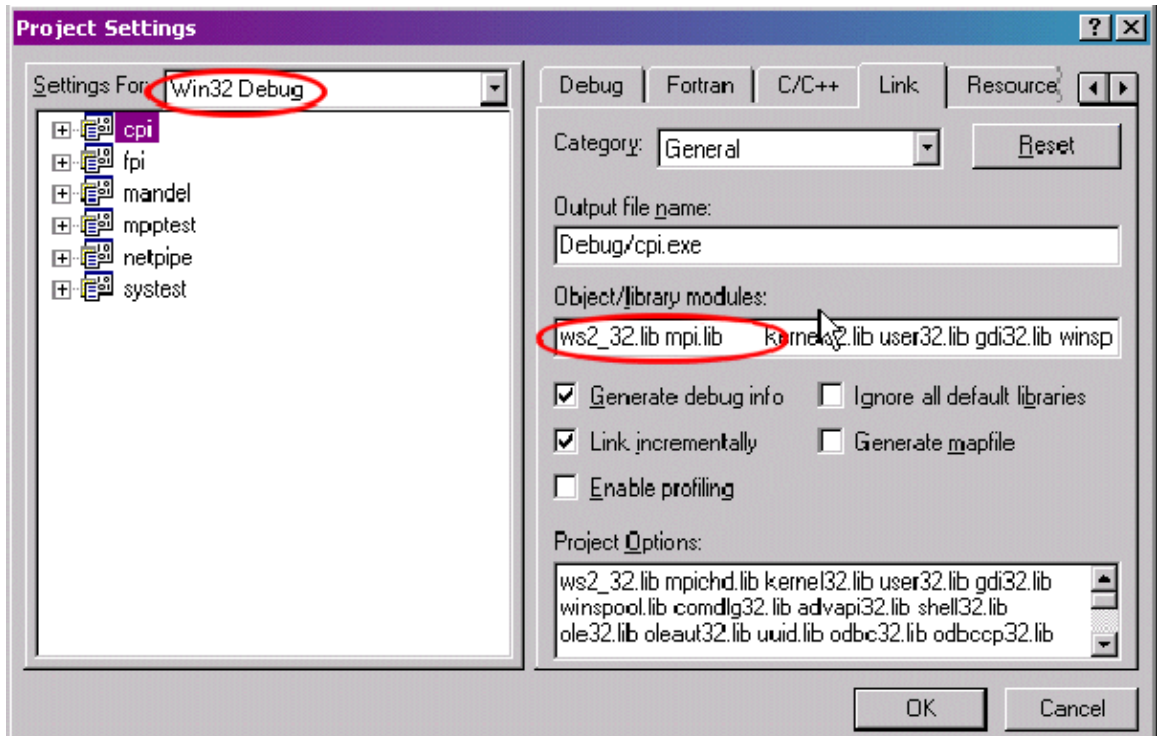
1. Create a project and add your source files.

New

| Files | Projects | Workspaces | Other Documents |

- New Database Wizard
- Utility Project
- Win32 Application
- Win32 Console Application
- Win32 Dynamic-Link Library
- Win32 Static Library

Project name:

Location:

C:\

- ● Create new workspace
- ○ Add to current workspace
  - ☐ Dependency of:
  - cpi

Platforms:

☑ Win32

OK   Cancel

Microsoft Visual C++

Insert   Project   Build   Tools   Window   Help

Set Active Project   ▶   piecefn
Add To Project   ▶   New...
Source Control   ▶   New Folder...
Dependencies...   Files...
Settings...   Alt+F7   Data Connection...
Export Makefile...   Components and Controls...
Insert Project into Workspace...

asses
obals
ses
l classes
t classes
e classes
classes

2. Bring up the settings for the project by hitting Alt F7. Select the Preprocessor Category from the C/C++ tab. Enter HAVE_NO_VARIABLE_RETURN_TYPE_SUPPORT into the Preprocessor box. Enter C:\Program Files\DeinoMPI\include into the "Additional include directories"
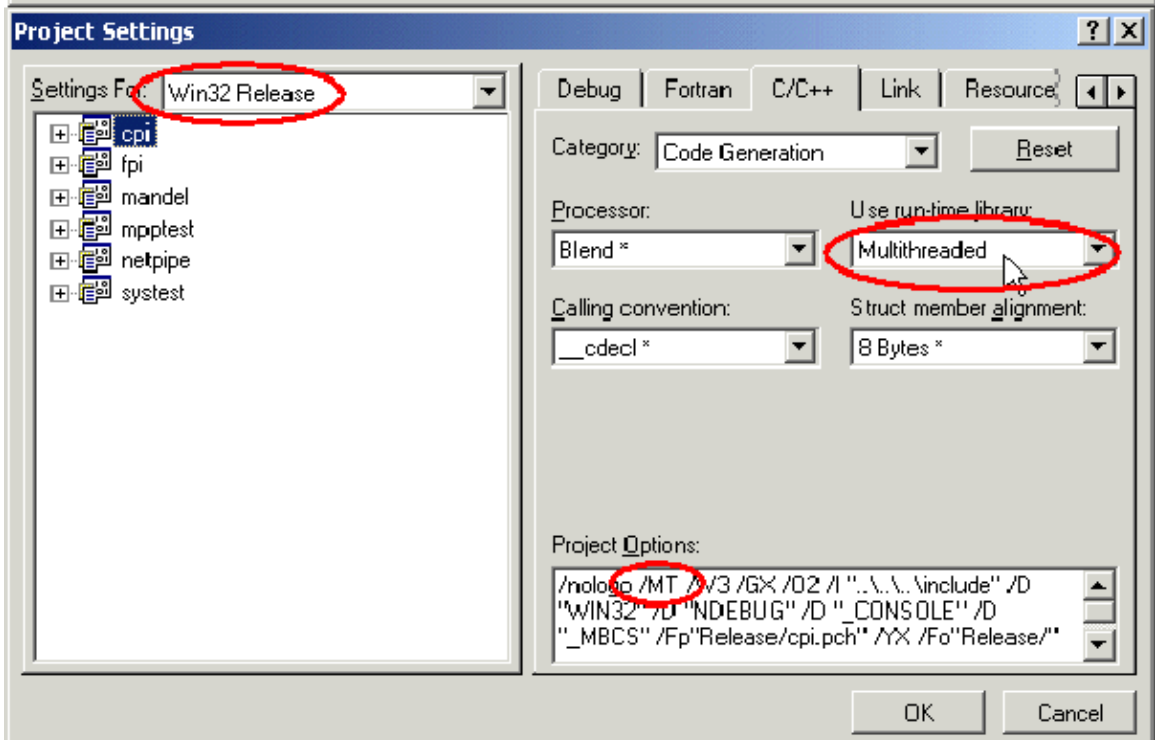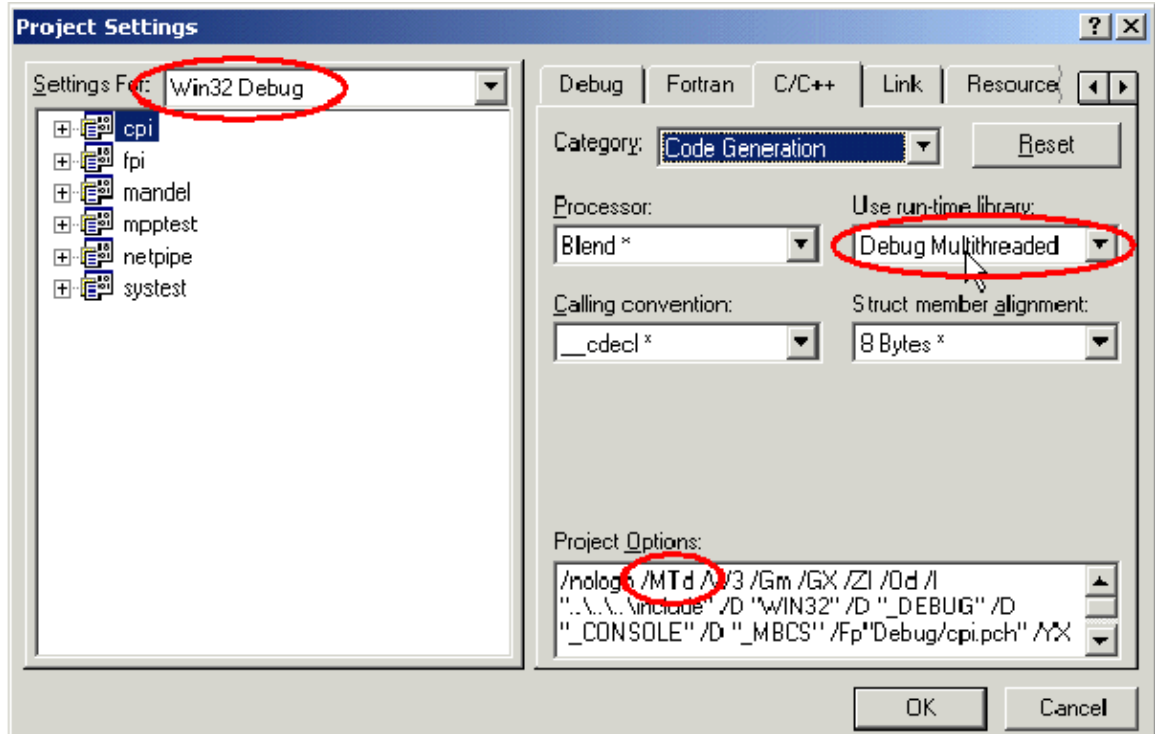
box.
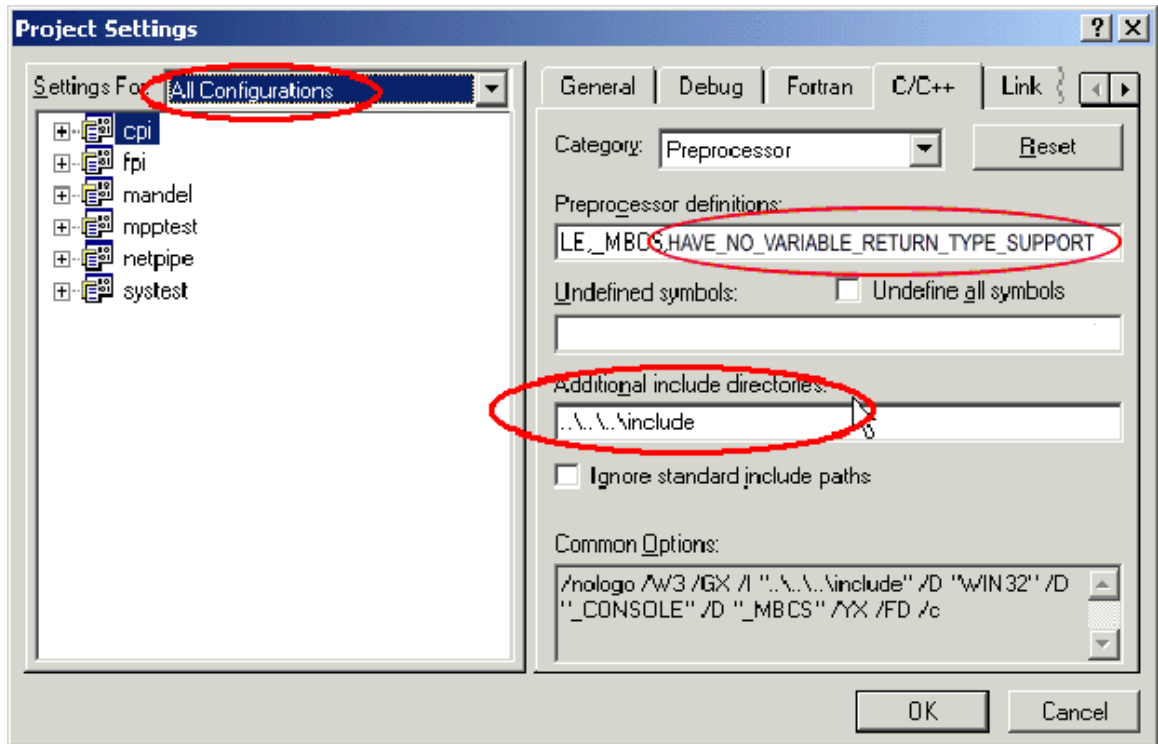
3. Select the Input Category from the Link tab. Add cxx.lib and mpi.lib to the Object/library modules box. Add C:\Program Files\DeinoMPI\lib to the "Additional library path" box.
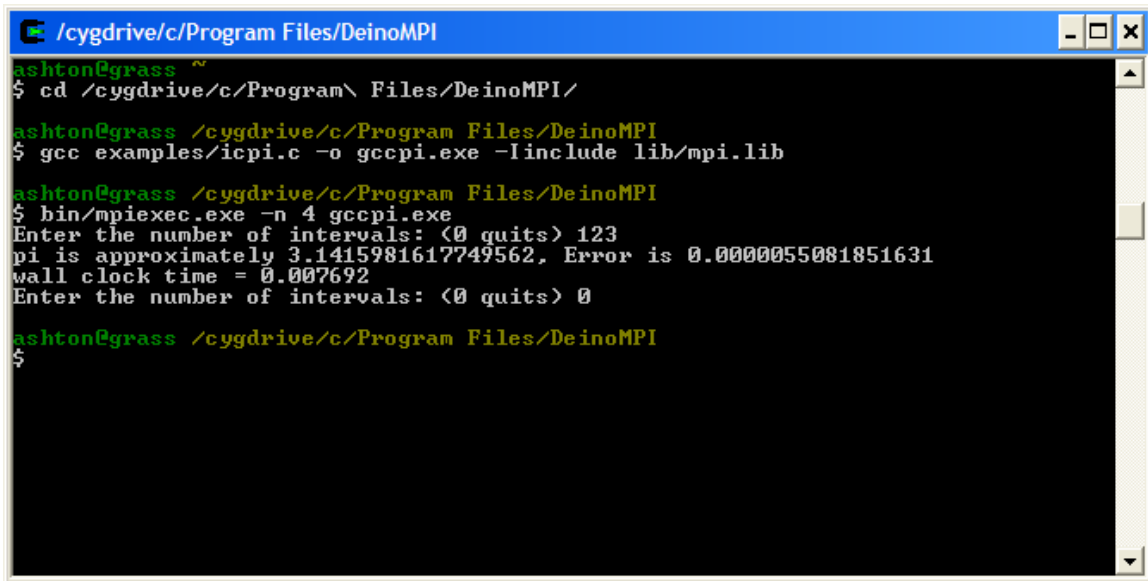
Project Settings (Win32 Debug)

Settings For: Win32 Debug

- cpi
- fpi
- mandel
- mpptest
- netpipe
- systest

Tabs: Debug | Fortran | C/C++ | Link | Resource

Category: General   Reset

Output file name:
Debug/cpi.exe

Object/library modules:
ws2_32.lib mpi.lib   kernel32.lib user32.lib gdi32.lib winsp

☑ Generate debug info   ☐ Ignore all default libraries
☑ Link incrementally   ☐ Generate mapfile
☐ Enable profiling

Project Options:
ws2_32.lib mpichd.lib kernel32.lib user32.lib gdi32.lib
winspool.lib comdlg32.lib advapi32.lib shell32.lib
ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib

OK   Cancel

Project Settings (Win32 Release)

Settings For: Win32 Release

- cpi
- fpi
- mandel
- mpptest
- netpipe
- systest

Tabs: Debug | Fortran | C/C++ | Link | Resource

Category: General   Reset

Output file name:
Release/cpi.exe

Object/library modules:
ws2_32.lib mpi.lib   kernel32.lib user32.lib gdi32.lib winspo

☐ Generate debug info   ☐ Ignore all default libraries
☐ Link incrementally   ☐ Generate mapfile
☐ Enable profiling

Project Options:
ws2_32.lib mpich.lib kernel32.lib user32.lib gdi32.lib
winspool.lib comdlg32.lib advapi32.lib shell32.lib
ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib

OK   Cancel

4. Change the settings to use the thread safe libraries

5. Compile your application.

Cygwin

http://www.cygwin.com. Cygwin users can use gcc and g77 to compile DeinoMPI applications. Link C applications with mpi.lib and Fortran applications with fmpich2g.lib. In order to run your application you must have the cygwin dll in the system path or in the same location as your executable. Otherwise the application will not load. This is important because the process manager does not use the user path so even though you can run your application as a single process within the bash shell it will fail when you try to run it with mpiexec. Either copy cygwin1.dll to the Windows system32 directory or put it in the same location as your executable.

The C++ interface to MPI is not available under cygwin because g++ does not create the same symbols as Visual C++ which was used to create the C++ library cxx.lib.

MINGW
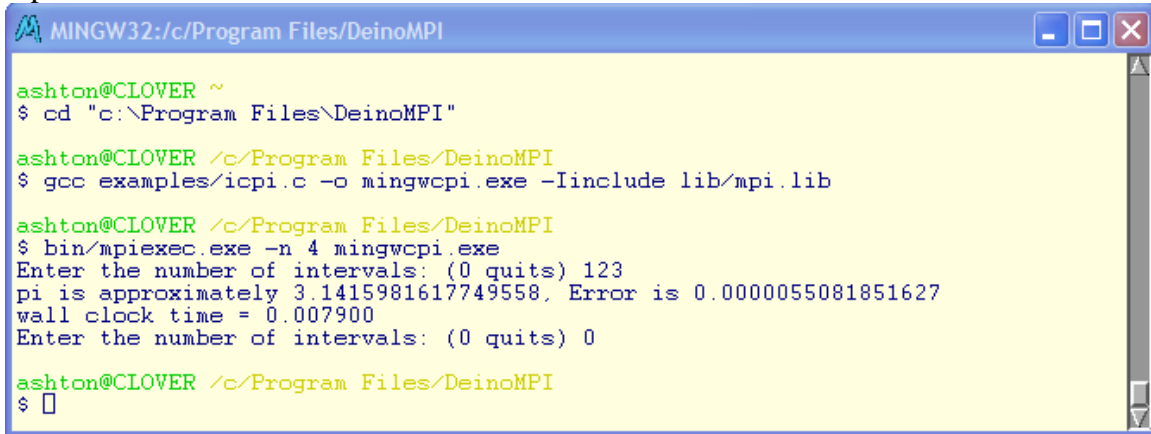
The Minimalist GNU for Windows environment allows you to create applications that do not depend on any dynamic libraries other than the Windows system libraries. This is an advantage over the cygwin environment because in order to get your cygwin DeinoMPI application to run you have to copy the cygwin1.dll to all your nodes. MINGW applications do not depend on any companion dlls.

In order to test DeinoMPI under MINGW the project files were downloaded from SourceForge: http://sourceforge.net/projects/mingw. It is recommended that you download the latest version but the names of the packages used to test DeinoMPI are provided here for reference.

1. binutils-2.16.91-20060119-1.tar.gz
2. gcc-core-3.4.2-20040916-1.tar.gz
3. gcc-g77-3.4.2-20040916-1.tar.gz
4. mingw32-make-3.81-1.tar.gz
5. mingw-runtime-3.10.tar.gz
6. mingw-utils-0.3.tar.gz
7. w32api-3.7.tar.gz

8. MSYS-1.0.10.exe

On the test machine all the files were unzipped to C:\bin\mingw. After unzipping the MINGW files MSYS was installed and pointed to C:\bin\mingw. From within the MSYS shell the example cpi application was compiled and run using mpiexec:
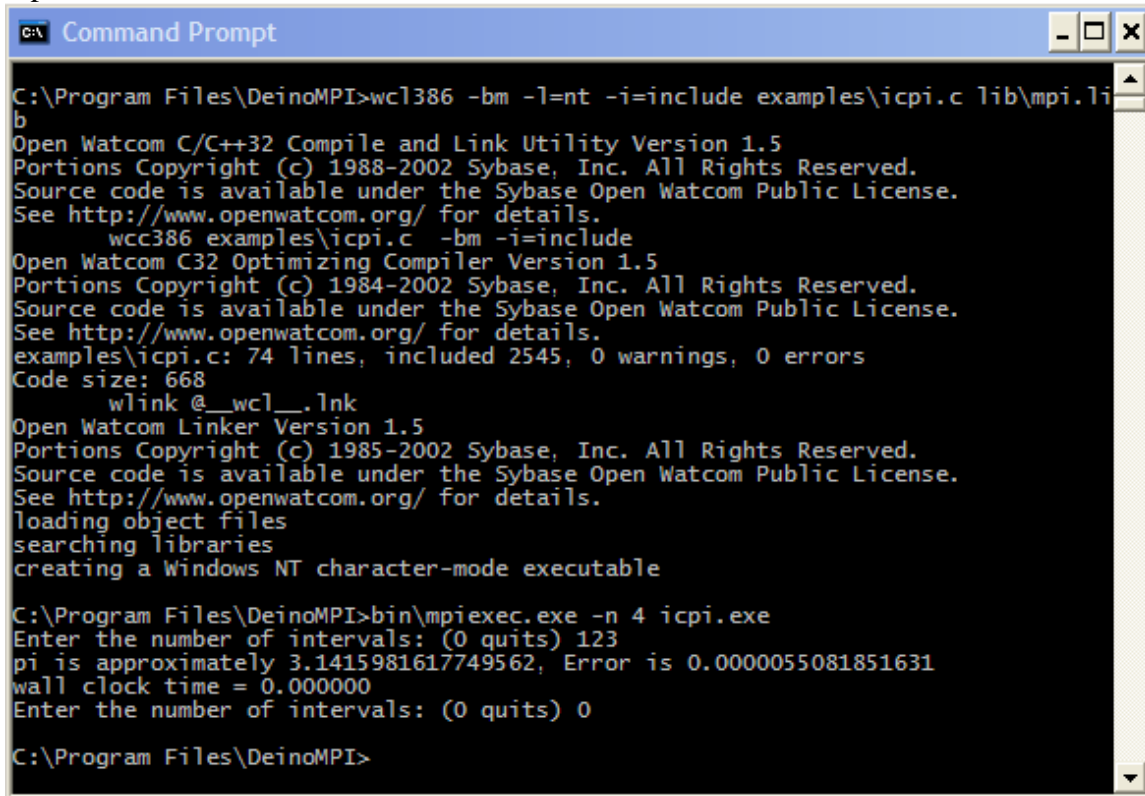
Watcom

The C compiler from OpenWatcom was tested. Version 1.5 was downloaded from http://www.openwatcom.org. The sample cpi program was compiled from the command line. Bring up a command prompt and set the environment variables defined in CHANGES.ENV in the root Watcom directory. Change to the DeinoMPI directory and execute the following command:

```
wcl386 –bm –l=nt –i=include examples\icpi.c lib\mpi.lib
```

Then execute the example using
mpiexec.

```
C:\Program Files\DeinoMPI>wcl386 -bm -l=nt -i=include examples\icpi.c lib\mpi.lib
Open Watcom C/C++32 Compile and Link Utility Version 1.5
Portions Copyright (c) 1988-2002 Sybase, Inc. All Rights Reserved.
Source code is available under the Sybase Open Watcom Public License.
See http://www.openwatcom.org/ for details.
        wcc386 examples\icpi.c  -bm -i=include
Open Watcom C32 Optimizing Compiler Version 1.5
Portions Copyright (c) 1984-2002 Sybase, Inc. All Rights Reserved.
Source code is available under the Sybase Open Watcom Public License.
See http://www.openwatcom.org/ for details.
examples\icpi.c: 74 lines, included 2545, 0 warnings, 0 errors
Code size: 668
        wlink @__wcl__.lnk
Open Watcom Linker Version 1.5
Portions Copyright (c) 1985-2002 Sybase, Inc. All Rights Reserved.
Source code is available under the Sybase Open Watcom Public License.
See http://www.openwatcom.org/ for details.
loading object files
searching libraries
creating a Windows NT character-mode executable

C:\Program Files\DeinoMPI>bin\mpiexec.exe -n 4 icpi.exe
Enter the number of intervals: (0 quits) 123
pi is approximately 3.1415981617749562, Error is 0.0000055081851631
wall clock time = 0.000000
Enter the number of intervals: (0 quits) 0

C:\Program Files\DeinoMPI>
```

Tiny C Compiler

This compiler almost worked. Version 0.9.23 was downloaded from
http://fabrice.bellard.free.fr/tcc. tcc doesn't understand the dll macros so mpi.h had to be
modified to remove those definitions by simply defining them to nothing:
```
#define MPI_CALL
#define MPI_DLL_SPEC
```
Then it complained about the extern declarations so these two lines had to be removed:
```
extern MPI_DLL_SPEC MPI_Fint * MPI_F_STATUS_IGNORE;
extern MPI_DLL_SPEC MPI_Fint * MPI_F_STATUSES_IGNORE;
```
In order to link with the MPI library an export file had to be created. The file
deino_mpich2mpi.def was created by executing this command:
```
c:\tcc-0.9.23\tcc\tiny_impdef.exe
c:\windows\system32\deino_mpich2mpi.dll
```
Then the cpi example was compiled using this command:
```
c:\tcc-0.9.23\tcc\tcc.exe -o tccpi.exe -DUSE_GCC -Iinclude -
Ic:\tcc-0.9.23\include examples\icpi.c deino_mpich2mpi.def
c:\tcc-0.9.23\lib\msvcrt.def
```
The code compiled but link failed with an unresolved symbol "_start".
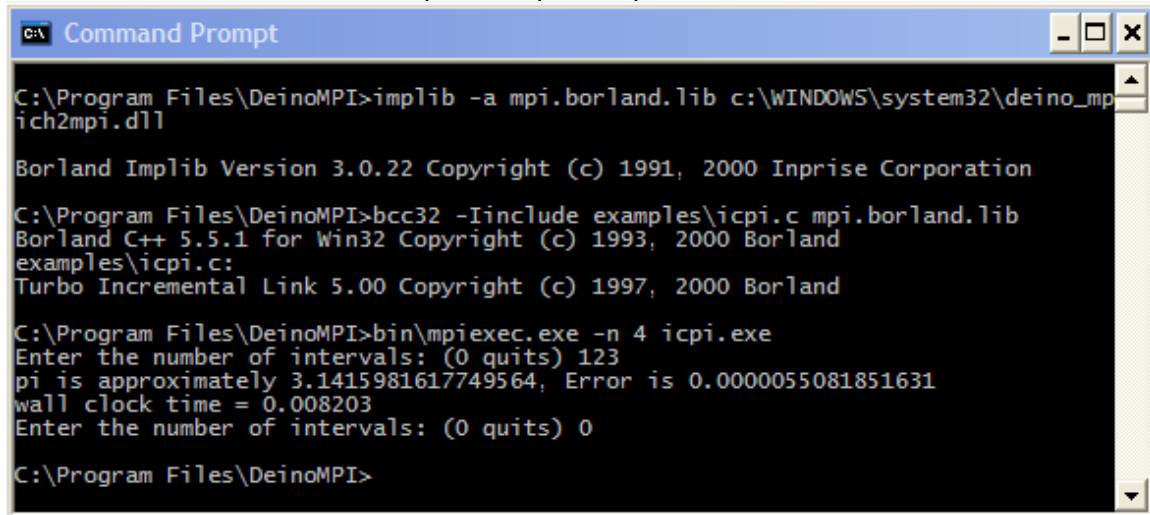
Borland C

The Borland C/C++ compiler command line tools version 5.5 package was downloaded
from http://www.borland.com/downloads/download_cbuilder.html (If the link has
changed then you can always start at the root website and navigate from there
http://www.borland.com). The compiler was installed in the default location and the

configuration files were created in accordance with the readme.txt file. There was a mistake in the readme in that the configuration files need to be place in the bin directory and not the root directory. The Borland compiler doesn't know how to read the Microsoft import library format so an import library needed to be created before any DeinoMPI applications could be compiled. This command created the import library:

```
c:\Borland\bcc55\bin\implib -a mpi.borland.lib
c:\Windows\system32\deino_mpich2mpi.dll
```

Then you can compile the example cpi application from a command prompt with this command:

```
bcc32 -Iinclude examples\icpi.c mpi.borland.lib
```


Digital Mars

The Digital Mars C/C++ compiler version 8.48 was downloaded from http://www.digitalmars.com. The C/C++ compiler (dm848c.zip) and Basic Utilities (bup.zip) packages were unzipped to C:\bin and a dm subdirectory was created by doing so. Then the icpi sample application was created from a command prompt. The Digital Mars C compiler cannot read the mpi.lib import library so a new one was created from deino_mpich2mpi.dll before the sample application could be compiled. Here is the command to create an import library called mpi_dm.lib:

```
c:\bin\dm\bin\implib.exe /s mpi_dm.lib
c:\Windows\system32\deino_mpich2mpi.dll
```

With the import library created icpi could be compiled with the following command:

```
c:\bin\dm\bin\cl.exe /Iinclude examples\icpi.c mpi_dm.lib
```

```
C:\Program Files\DeinoMPI>set PATH=c:\bin\dm\bin;%PATH%

C:\Program Files\DeinoMPI>\bin\dm\bin\implib.exe /s mpi_dm.lib c:\WINDOWS\system
32\deino_mpich2mpi.dll
Digital Mars Import Library Manager Version 7.6B1n
Copyright (C) Digital Mars 2000.  All Rights Reserved.
Input is a Windows NT DLL file 'deino_mpich2mpi.dll'.
Output is a Windows NT import library.
Digital Mars Import Library Creator complete.

C:\Program Files\DeinoMPI>\bin\dm\bin\cl.exe /Iinclude examples\icpi.c mpi_dm.li
b
dmc -Jm -w /Iinclude examples\icpi.c mpi_dm.lib -mn
link icpi,,,mpi_dm+user32+kernel32/noi;



C:\Program Files\DeinoMPI>bin\mpiexec.exe -n 4 icpi.exe
Enter the number of intervals: (0 quits) 123
pi is approximately 3.1415981617749562, Error is 0.0000055081851630
wall clock time = 0.008339
Enter the number of intervals: (0 quits) 0

C:\Program Files\DeinoMPI>
```

Lcc-win32

The lcc-win32 compiler package version 4.0 July 30, 2006 was downloaded from
http://www.cs.virginia.edu/~lcc-win32. It was installed in the default location C:\lcc.
The compiler cannot link with the DeinoMPI import library, mpi.lib, so a new one was
created using the tools provided. Creating the import library took two steps:

```
pedump /EXP lib\mpi.lib > mpi.lcc.exp
buildlib mpi.lcc.exp mpi.lcc.lib
```

Once the import library was created then the sample application could be compiled:

```
lc -Iinclude examples\icpi.c mpi.lcc.lib
```



```
C:\Program Files\DeinoMPI>set PATH=C:\lcc\bin;%PATH%

C:\Program Files\DeinoMPI>pedump /EXP lib\mpi.lib > mpi.lcc.exp

C:\Program Files\DeinoMPI>buildlib mpi.lcc.exp mpi.lcc.lib

C:\Program Files\DeinoMPI>lc -Iinclude examples\icpi.c mpi.lcc.lib

C:\Program Files\DeinoMPI>bin\mpiexec.exe -n 4 icpi.exe
Enter the number of intervals: (0 quits) 123
pi is approximately 3.1415981617749600, Error is 0.0000055081851600
wall clock time = 0.008325
Enter the number of intervals: (0 quits) 0

C:\Program Files\DeinoMPI>
```

Pelles C

The Pelles C compiler can be found in various places,
http://www.smorgasbordet.com/pellesc or http://www.christian-heffner.de. If either of
these mirrors is not available just search the web for "Pelles C for Windows". Version
4.0 was installed to the default location. Compiling is straightforward if the cc compiler

driver is used. Here is the command line used to compile icpi.c from a command prompt after changing into the C:\Program Files\DeinoMPI directory:

```
C:\Program Files\PellesC\bin\cc.exe /Ze -Iinclude examples\icpi.c
lib\mpi.lib
```

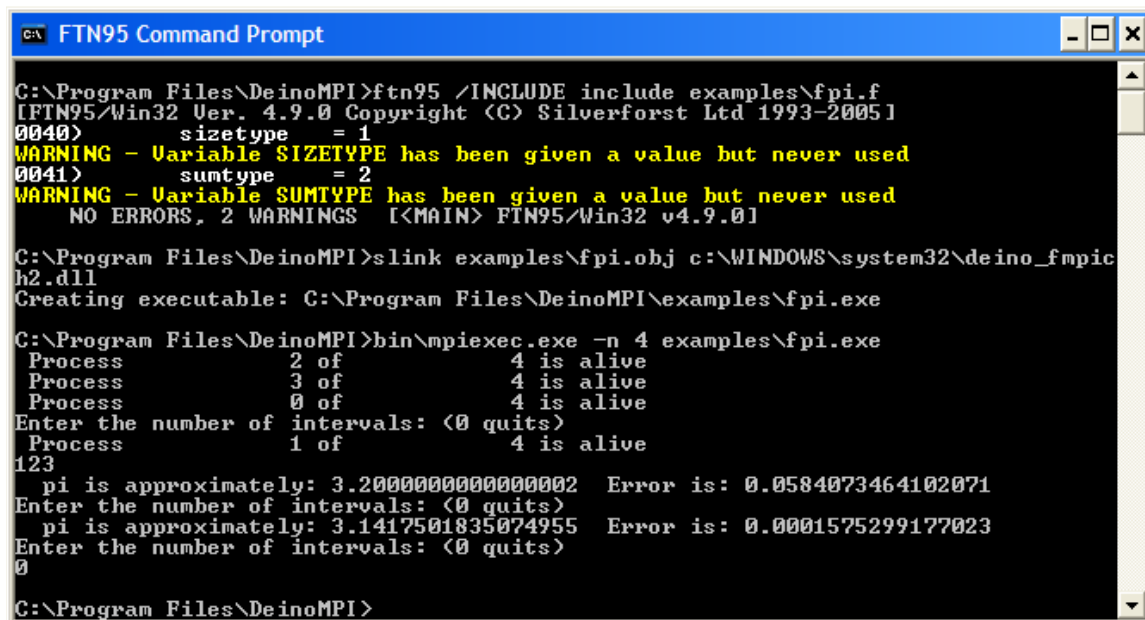The /Ze option tells the compiler to use the Microsoft extensions to C.

Salford FTN95

The single user evaluation package of the Salford Fortran95 compiler version 4.9 was downloaded from http://www.silverfrost.com. The package was installed to the default location and then the redistributable dlls were copied to the Windows system32 directory. This is required because executables generated by FTN95 depend on these dlls and they must be available to the process manager that starts the MPI job. These dlls were found in C:\Program Files\Salford Software\FTN95\redist*.dll. They need to be copied to all the nodes that will participate in the MPI job or they need to be copied to the same location as the MPI application.

The example fpi application was compiled from the command line. First an FTN95 command prompt was brought up from the Start menu link installed with the package. After changing into the DeinoMPI directory the following two commands were issued to compile the sample application:

```
ftn95 /INCLUDE include examples\fpi.f
slink examples\fpi.obj c:\Windows\system32\deino_fmpich2.dll
```

The slink tool creates an import library directly from the Fortran dll from the DeinoMPI distribution. We were not able to get the compiler to link with the import libraries provided with
DeinoMPI.

```
CN  FTN95 Command Prompt                                           _ □ ×

C:\Program Files\DeinoMPI>ftn95 /INCLUDE include examples\fpi.f
[FTN95/Win32 Ver. 4.9.0 Copyright (C) Silverforst Ltd 1993-2005]
0040)      sizetype   = 1
WARNING - Variable SIZETYPE has been given a value but never used
0041)      sumtype    = 2
WARNING - Variable SUMTYPE has been given a value but never used
    NO ERRORS, 2 WARNINGS  [<MAIN> FTN95/Win32 v4.9.0]

C:\Program Files\DeinoMPI>slink examples\fpi.obj c:\WINDOWS\system32\deino_fmpic
h2.dll
Creating executable: C:\Program Files\DeinoMPI\examples\fpi.exe

C:\Program Files\DeinoMPI>bin\mpiexec.exe -n 4 examples\fpi.exe
 Process           2 of           4 is alive
 Process           3 of           4 is alive
 Process           0 of           4 is alive
Enter the number of intervals: (0 quits)
 Process           1 of           4 is alive
123
  pi is approximately: 3.2000000000000002   Error is: 0.0584073464102071
Enter the number of intervals: (0 quits)
  pi is approximately: 3.1417501835074955   Error is: 0.0001575299177023
Enter the number of intervals: (0 quits)
0

C:\Program Files\DeinoMPI>
```

Absoft Fortran

The Absoft Fortran compiler is compatible with the DeinoMPI libraries. The Absoft compiler generates all uppercase external symbols using the C calling convention by default so link with fmpich2.lib.

Portland Group Fortran Win64 compiler

The Portland Group Fortran compiler for Windows was tested on a Windows Server 2003 x64 machine. You can compile and link with fmpich2.lib. To compile the sample fpi application first a PGI Command Prompt (64) was opened using the start menu shortcut under the PGI Workstation tools menu. After changing into the DeinoMPI directory the following command compiled the fpi.f sample application:

```
pgf95 -Iinclude examples\fpi.f lib\fmpich2.lib
```

The pgf95 compiler creates a dependency on pg.dll so you have to copy this dll to the system directory of all the machines or put it in the same location as your MPI application before you can start a job with mpiexec.

Other compilers

If you are using a compiler that cannot link with the Microsoft format mpi.lib or any of the other libraries in the package then you will need to read the documentation for your compiler to find out how to link with existing dlls. You will need to be able to create an import library for deino_mpich2mpi.dll. Most compilers provide tools to strip the export table from existing dlls and then create import libraries from this information. If you are successful in this method please send information on how you did it to support@deino.net so it can be added to the web site.