Introduction to the Eclipse IDE + PTP

M. D. Jones, Ph.D.

Center for Computational Research
University at Buffalo
State University of New York

High Performance Computing I, 2014

Eclipse History

- Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and Webgain formed eclipse.org board in 2001
- Open-sourced under Eclipse Public License (EPL), based on IBM Common Public License
- 2004 consortium reorganized into not-for-profit corporation, Eclipse foundation
- Four classes of membership - strategic developers, strategic consumers, add-in providers, open source project leaders
- Currently many projects and subprojects

What is Eclipse?

- Vendor neutral open source development platform
- Platform for tool integration
- Plug-in based framework to create, integrate, and use software tools (called “Equinox,” part of the Open Services Gateway initiative, or OSGi):
    - Plug-ins form a “bundle”
    - bundle (un)installation, update
    - bootstrap, launching, and extension registry

Eclipse Platform

- Core framework for all plug-in extensions
- Common facilities:
  - Workbench (user interface)
  - Portable UI libraries
  - Language independent debugging
  - CVS/subversion/git ... support
  - Dynamic update/install service

Plug-ins

There are a lot of eclipse plug-ins available:

- C/C++ development tools (CDT)
- Fortran development tools (Photran, now part of PTP)
- Parallel tools platform (PTP)
- Java development tools (JDT)
- Plug-in development environment (PDE)
- many more ...

Installing `eclipse`

`www.eclipse.org`

- `Fortran` support is through the **Photran** project (now part of Parallel Tools Platform):

  `www.eclipse.org/photran`
- The Parallel Tools Platform (PTP) is an exciting development, but also can be somewhat tricky to integrate, especially with remote resources.
- Latest release of eclipse (Juno) has a package for parallel application developers that incorporates CDT and PTP

A Few More Installation Comments

- Can install multiple versions of eclipse without any difficulty (just watch out for overlapping workspaces)
- Not really intended as a “server” application - more of a locally installed development tool - that is why the remote tools are interesting/useful
- Remote development tools are still under (very) active development - great idea, though

Installing Eclipse & PTP

Steps for installing and configuring Eclipse plus the PTP, best to follow the latest on the PTP Wiki:

http://wiki.eclipse.org/PTP

The release notes for the current version generally have the most up-to-date instructions. Follow the instructions for installing Eclipse and PTP, running updates, enabling PTP update site, etc.

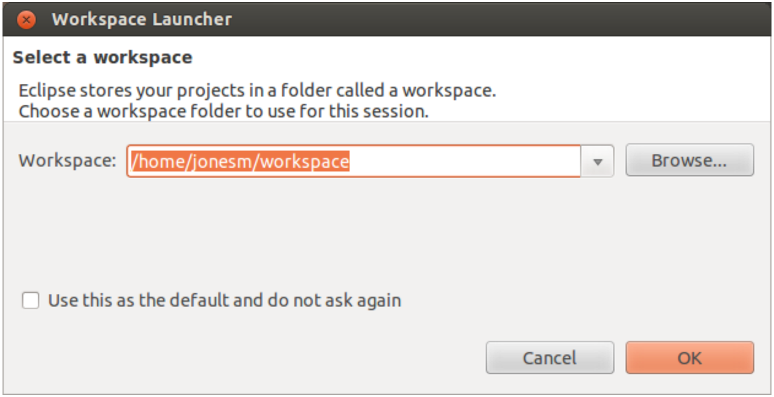
Some Issues

Depending on your operating system and environment, some parts of eclipse and PTP may or may not be available to you - some issues that I have encountered:

- Windows - I can not help you much there, but it is claimed to run under Windows
- Fortran support under Mac OS (and very likely Windows as well) - need to deploy Fortran compiler that you want to use
- MPI support - you need a working MPI to use the MPI features of PTP (depends on your platform)
- Virtual machine installs can be quite handy for dealing with some of these limitations - e.g., use VirtualBox and your favorite Linux distribution inside a VM without messing around with your base machine

Workspace

When you start `eclipse` it asks you for a workspace to use - a directory in which it will store working files. I tend to accept the default (the directory is browsable from the command line)



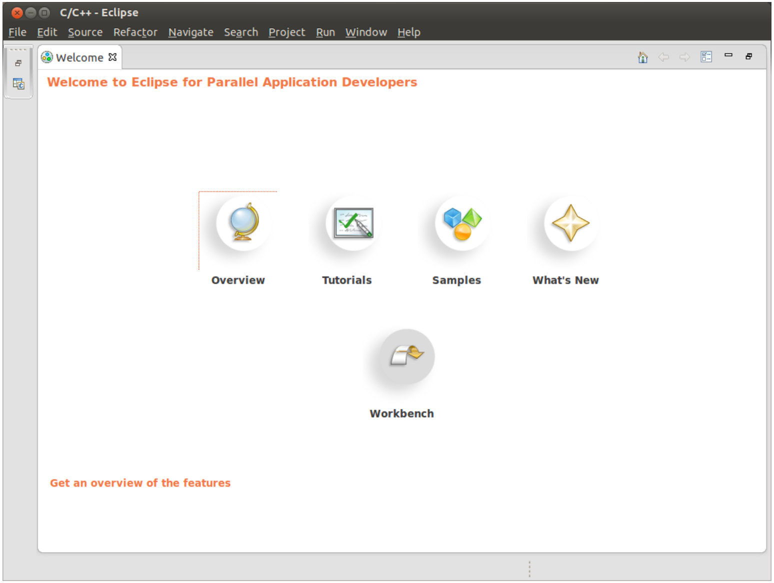
Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: /home/jonesm/workspace

☐ Use this as the default and do not ask again

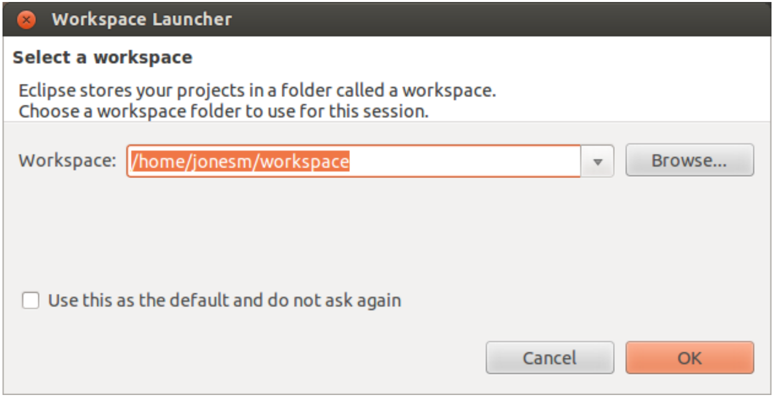

C/C++ - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
Overview
Tutorials
Samples
What's New
Workbench

Welcome to Eclipse for Parallel Application Developers

Get an overview of the features

Start New Project

File->New->Project will start a new project in your workspace.


Workspace Launcher

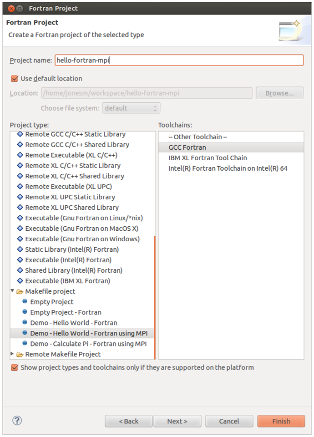
Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: /home/jonesm/workspace

[ ] Use this as the default and do not ask again

Selecting a project type determines the toolchain (compilers, linker, etc.) and tabs that `eclipse` will use and display. Note that you can first open the perspective (`Window->Open Perspective`) and then start a new project (`File->New->Fortran Project->Makefile Project`):



Fortran Project

Fortran Project

Create a Fortran project of the selected type

Project name: hello-fortran-mpi

☑ Use default location

Location: /home/jonesm/workspace/hello-fortran-mpi

Choose file system: default

Project type:

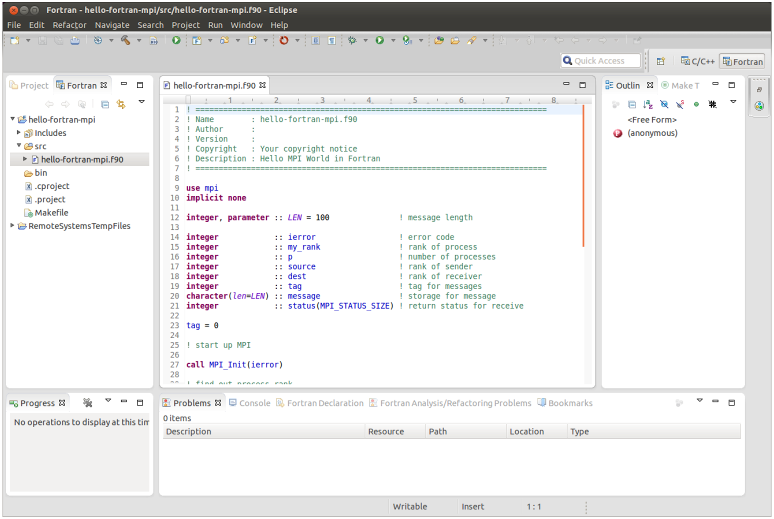
- Remote GCC C/C++ Static Library
- Remote GCC C/C++ Shared Library
- Remote Executable (XL C/C++)
- Remote XL C/C++ Static Library
- Remote XL C/C++ Shared Library
- Remote Executable (XL UPC)
- Remote XL UPC Static Library
- Remote XL UPC Shared Library
- Executable (Gnu Fortran on Linux/*nix)
- Executable (Gnu Fortran on MacOS X)
- Executable (Gnu Fortran on Windows)
- Static Library (Intel(R) Fortran)
- Executable (Intel(R) Fortran)
- Shared Library (Intel(R) Fortran)
- Executable (IBM XL Fortran)
- Makefile project
  - Empty Project
  - Empty Project - Fortran
  - Demo - Hello World - Fortran
  - **Demo - Hello World - Fortran using MPI**
  - Demo - Calculate Pi - Fortran using MPI
- Remote Makefile Project

Toolchains:

- -- Other Toolchain --
- **GCC Fortran**
- IBM XL Fortran Tool Chain
- Intel(R) Fortran Toolchain on Intel(R) 64

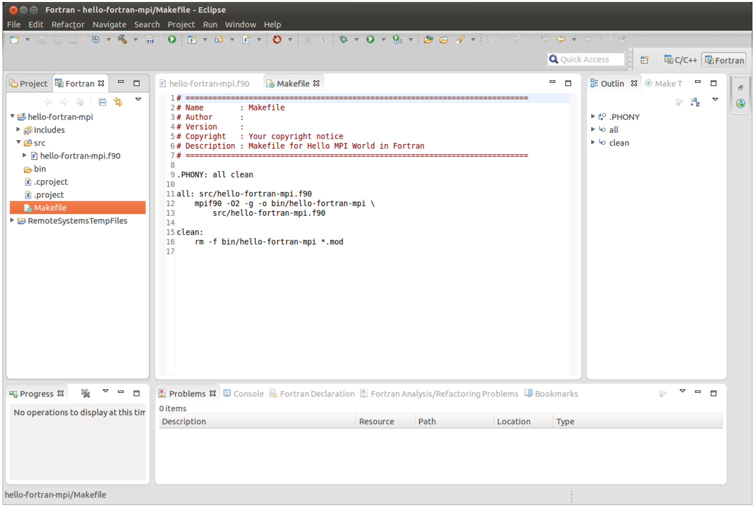
☑ Show project types and toolchains only if they are supported on the platform

In this case I will load up a prepared sample MPI code that should look pretty familiar:



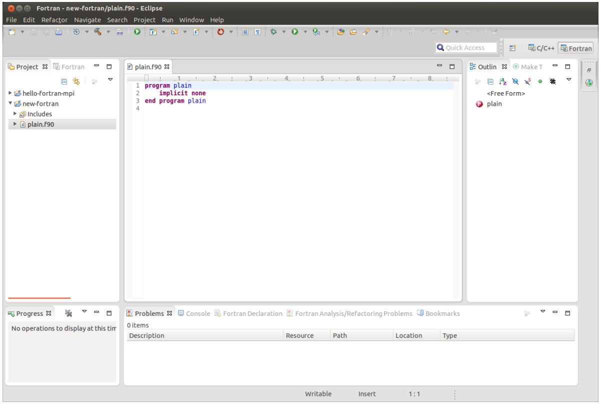
```fortran
! ============================================================================
! Name        : hello-fortran-mpi.f90
! Author      :
! Version     :
! Copyright   : Your copyright notice
! Description : Hello MPI World in Fortran
! ============================================================================

use mpi
implicit none

integer, parameter :: LEN = 100               ! message length

integer             :: ierror                 ! error code
integer             :: my_rank                ! rank of process
integer             :: p                      ! number of processes
integer             :: source                 ! rank of sender
integer             :: dest                   ! rank of receiver
integer             :: tag                    ! tag for messages
character(len=LEN)  :: message                ! storage for message
integer             :: status(MPI_STATUS_SIZE) ! return status for receive

tag = 0

! start up MPI

call MPI_Init(ierror)

! find out process rank
```

and you can edit the Makefile if you want to tweak the various settings (note the built-in editor with syntax highlighting and other nice features):



```
# ============================================================================
# Name        : Makefile
# Author      :
# Version     :
# Copyright   : Your copyright notice
# Description : Makefile for Hello MPI World in Fortran
# ============================================================================

.PHONY: all clean

all: src/hello-fortran-mpi.f90
	mpif90 -O2 -g -o bin/hello-fortran-mpi \
		src/hello-fortran-mpi.f90

clean:
	rm -f bin/hello-fortran-mpi *.mod
```

... and then you can edit a new file in your project by right-clicking on the project folder and selecting **New->File From Template.** You can then type in some new code:

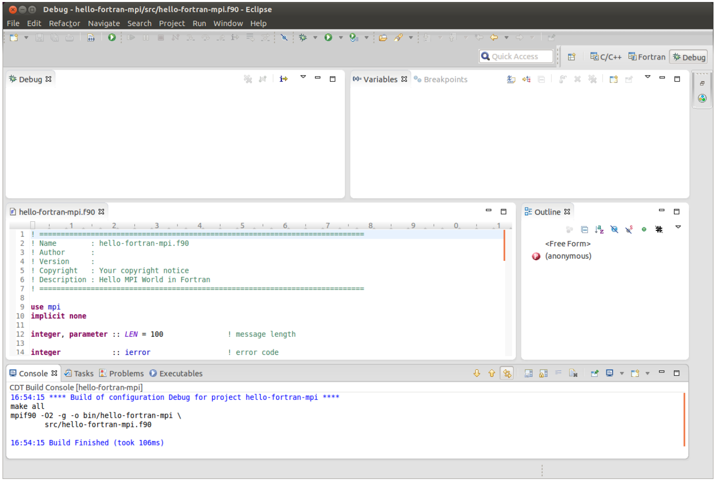


```
program plain
   implicit none
end program plain
```

You can follow a similar path to create a Makefile (new projects can automatically create a Makefile if you select that type of project initially).

Changing to the Debug Perspective

Selecting the **Project->Build Project** should build a binary, and then you can run it (**Run As ...**) and change to the *Debug Perspective*:



```
!  ==========================================================================
!  Name        : hello-fortran-mpi.f90
!  Author      :
!  Version     :
!  Copyright   : Your copyright notice
!  Description : Hello MPI World in Fortran
!  ==========================================================================

use mpi
implicit none

integer, parameter :: LEN = 100              ! message length

integer            :: ierror                 ! error code
```

```
CDT Build Console [hello-fortran-mpi]
16:54:15 **** Build of configuration Debug for project hello-fortran-mpi ****
make all
mpif90 -O2 -g -o bin/hello-fortran-mpi \
        src/hello-fortran-mpi.f90

16:54:15 Build Finished (took 106ms)
```

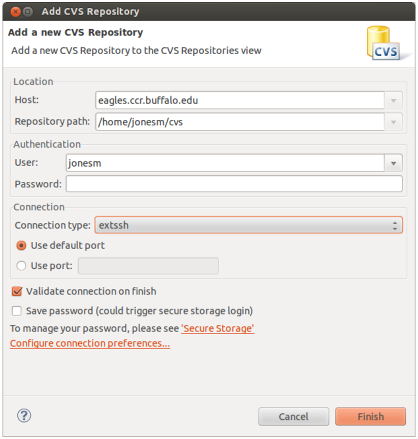
CVS?

If you are not using CVS (or a close relative), you should be:

- Concurrent Version System
- Shared access for multiple developers to develop single code “repository” with version control, branching, etc.
- Popular in open source software development (even “anonymous” read-only access to online CVS repositories
- Git, subversion close competitors (eclipse has git support, is planning on adding subversion support)

Adding CVS Repo

Window->Open Perspective -> Other -> CVS Repository Exploring changes to CVS perspective



Add a new CVS Repository

Add a new CVS Repository to the CVS Repositories view

Location

Host: eagles.ccr.buffalo.edu

Repository path: /home/jonesm/cvs

Authentication

User: jonesm

Password:

Connection

Connection type: extssh

● Use default port

○ Use port:

☑ Validate connection on finish

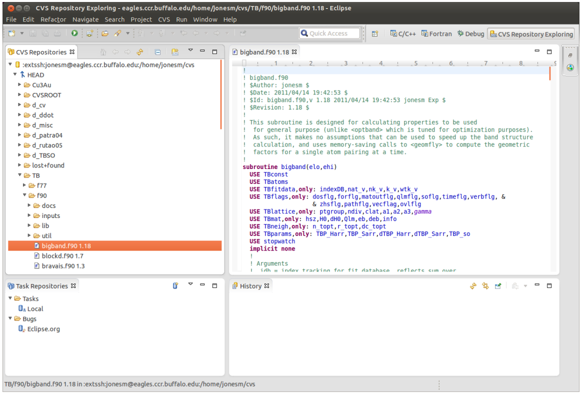
☐ Save password (could trigger secure storage login)

To manage your password, please see 'Secure Storage'

Configure connection preferences...

Hit the **Add CVS Repository** button ...

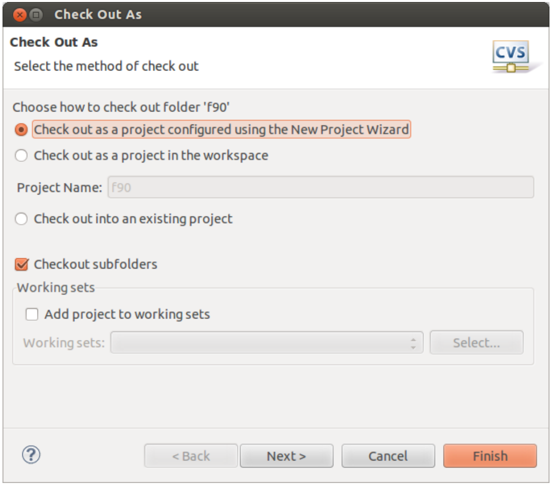
Provided that you authenticate properly (I am using ssh), you can then browse the CVS repo:



```
!
! bigband.f90
! $Author: jonesm $
! $Date: 2011/04/14 19:42:53 $
! $Id: bigband.f90,v 1.18 2011/04/14 19:42:53 jonesm Exp $
! $Revision: 1.18 $
!
! This subroutine is designed for calculating properties to be used
!  for general purpose (unlike <optband> which is tuned for optimization purposes).
!  As such, it makes no assumptions that can be used to speed up the band structure
!  calculation, and uses memory-saving calls to <geomfly> to compute the geometric
!  factors for a single atom pairing at a time.
!
subroutine bigband(elo,ehi)
  USE TBconst
  USE TBatoms
  USE TBfitdata,only: indexDB,nat_v,nk_v,k_v,wtk_v
  USE TBflags,only: dosflg,forflg,matoutflg,qlmflg,soflg,timeflg,verbflg, &
                  & zhsflg,pathflg,vecflag,ovlflg
  USE TBlattice,only: ptgroup,ndiv,clat,a1,a2,a3,gamma
  USE TBmat,only: hsz,H0,dH0,Qlm,eb,deb,info
  USE TBneigh,only: n_topt,r_topt,dc_topt
  USE TBparams,only: TBP_Harr,TBP_Sarr,dTBP_Harr,dTBP_Sarr,TBP_so
  USE stopwatch
  implicit none
  !
  ! Arguments
  !  idb = index tracking for fit database, reflects sum over
```

TB/f90/bigband.f90 1.18 in :extssh:jonesm@eagles.ccr.buffalo.edu:/home/jonesm/cvs

(note that the above is just browsing the files in the repository). Right-click on the project that you want, and select check out to get a copy ...



Check Out As

Check Out As

Select the method of check out

Choose how to check out folder 'f90'

[✓ Check out as a project configured using the New Project Wizard]

Check out as a project in the workspace

Project Name: f90

Check out into an existing project

[✓] Checkout subfolders

Working sets

[ ] Add project to working sets

Working sets:

and you now have a rich set of CVS functionality within `eclipse` (diffs, commits, patching, etc.)

More CVS Information

CVS is a topic in its own right - some links to more CVS documentation:

- http://offog.org/cvsintro.html
- http://cvsbook.red-bean.com/
- http://en.wikipedia.org/wiki/Concurrent_Versions_System

Git, subversion are also integrated with eclipse.

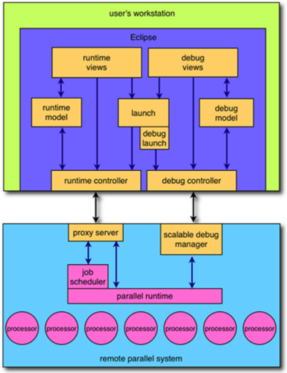
Parallel Tools Platform

Lofty goals:

- Support range of parallel architectures and runtime systems
- Scalable parallel debugger
- Support integration of parallel tools
- Simplify end-user interaction with parallel systems

PTP Architecture

Schematic overview of PTP:



Tricky bit (unsurprisingly) is launching parallel codes (few standards) on remote execution systems (many different job schedulers)...

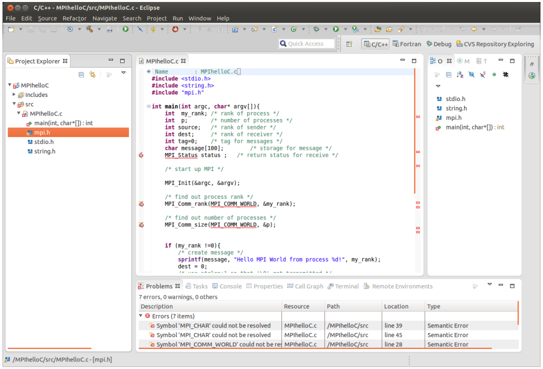
PTP Demonstration

Version 7.0 (officially released mid-2013) is the latest version, and what I will attempt to use for demonstration purposes.

- Best bet - follow carefully the steps outlined in the release notes for the latest PTP release (or pre-release if there is a significant feature that you want that has not yet made it into the released version). There are also frequent tutorials that are worth walking through if you are completely unfamiliar with the interface.

Simple MPI Example

Note that mousing over MPI functions shows calling syntax (also known as “Content Assist” - also can use CTRL-space), but you may need to fix the path settings first ...



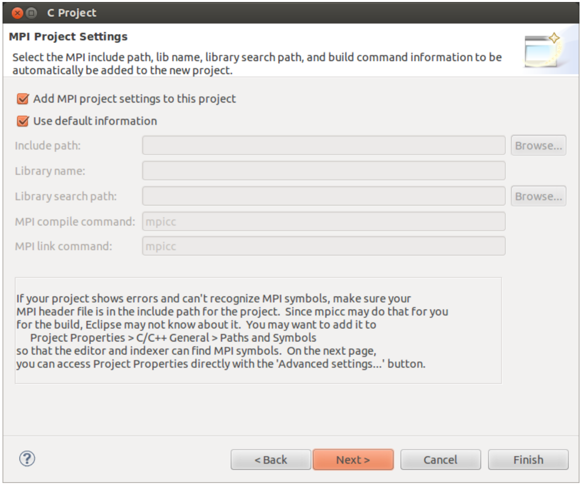
```
 Name        : MPIhelloC.c
#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main(int argc, char* argv[]){
    int  my_rank; /* rank of process */
    int  p;       /* number of processes */
    int source;   /* rank of sender */
    int dest;     /* rank of receiver */
    int tag=0;    /* tag for messages */
    char message[100];        /* storage for message */
    MPI_Status status ;   /* return status for receive */

    /* start up MPI */

    MPI_Init(&argc, &argv);

    /* find out process rank */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* find out number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &p);


    if (my_rank !=0){
        /* create message */
        sprintf(message, "Hello MPI World from process %d!", my_rank);
        dest = 0;
```

7 errors, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (7 items)				
Symbol 'MPI_CHAR' could not be resolved	MPIhelloC.c	/MPIhelloC/src	line 39	Semantic Error
Symbol 'MPI_CHAR' could not be resolved	MPIhelloC.c	/MPIhelloC/src	line 45	Semantic Error
Symbol 'MPI_COMM_WORLD' could not be res	MPIhelloC.c	/MPIhelloC/src	line 28	Semantic Error

MPI Artifacts

Going through the project wizard gives you the instructions for fixing missing include/library paths:



C Project

MPI Project Settings

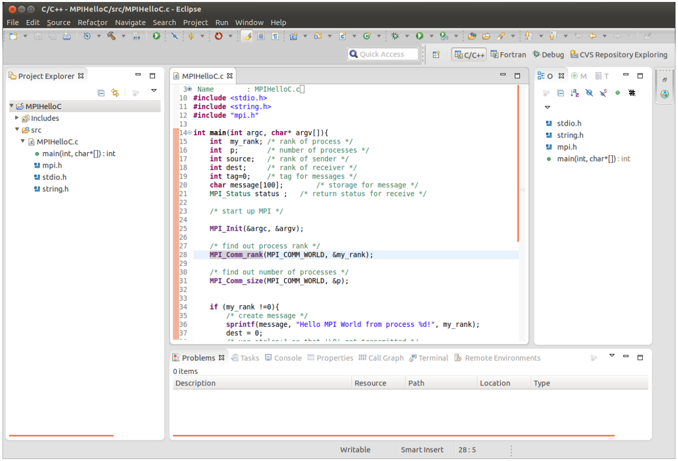
Select the MPI include path, lib name, library search path, and build command information to be automatically be added to the new project.

- [✓] Add MPI project settings to this project
- [✓] Use default information

Include path:	
Library name:	
Library search path:	
MPI compile command:	mpicc
MPI link command:	mpicc

If your project shows errors and can't recognize MPI symbols, make sure your MPI header file is in the include path for the project. Since mpicc may do that for you for the build, Eclipse may not know about it. You may want to add it to
    Project Properties > C/C++ General > Paths and Symbols
so that the editor and indexer can find MPI symbols. On the next page, you can access Project Properties directly with the 'Advanced settings...' button.

Once you have cleaned up the settings:



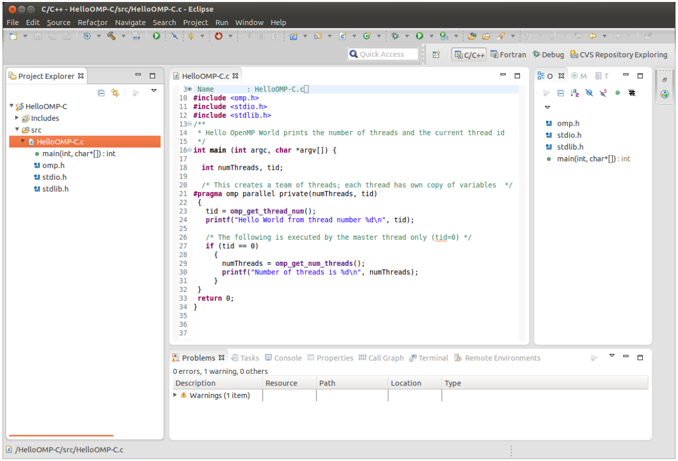
```
 Name          : MPIHelloC.c
#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main(int argc, char* argv[]){
    int  my_rank; /* rank of process */
    int  p;       /* number of processes */
    int source;   /* rank of sender */
    int dest;     /* rank of receiver */
    int tag=0;    /* tag for messages */
    char message[100];         /* storage for message */
    MPI_Status status ;   /* return status for receive */

    /* start up MPI */

    MPI_Init(&argc, &argv);

    /* find out process rank */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* find out number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &p);


    if (my_rank !=0){
        /* create message */
        sprintf(message, "Hello MPI World from process %d!", my_rank);
        dest = 0;
```

MPI Features

- "Mouse over" feature gives calling syntax
- "Control-space" keys give completion assistance in choosing MPI routines
- Press help key (typically F1) when clicking on a routine and help window should appear
- MPI Templates - e.g. type "mpisr <ctrl-space>" to automatically insert a template of MPI_Send and MPI_Recv ("mpi <ctrl-space> <ctrl-space>" should show all templates)

OpenMP Example

Eclipse PTP has gotten a lot better at automatically detecting the environment, but of course an OpenMP compliant compiler is a necessary prerequisite.



```
 Name         : HelloOMP-C.c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
/**
 * Hello OpenMP World prints the number of threads and the current thread id
 */
int main (int argc, char *argv[]) {

  int numThreads, tid;

  /* This creates a team of threads; each thread has own copy of variables  */
#pragma omp parallel private(numThreads, tid)
 {
   tid = omp_get_thread_num();
   printf("Hello World from thread number %d\n", tid);

   /* The following is executed by the master thread only (tid=0) */
   if (tid == 0)
     {
       numThreads = omp_get_num_threads();
       printf("Number of threads is %d\n", numThreads);
     }
 }
 return 0;
}
```

0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				

PTP Runtime and Debugging

- PTP runtime needs an install of MPI in your default environment
- Debugging - typically have to build the debugger (sdm) by hand - see PTP Wiki for steps (usually in the release notes for each version)
- Debug sessions are pretty flexible
- Remote development and debugging also supported through (new) synchronized projects, but setup can be more than a bit tricky (demo?)

More Information on PTP

- PTP Wiki,
  http://wiki.eclipse.org/ptp
- PTP Home,
  http://www.eclipse.org/ptp/
- PTP XSEDE14 Tutorial,
  http://wiki.eclipse.org/PTP/tutorials/XSEDE2014