Tutorial 1
Creating an Agent

In this tutorial, you'll learn how to create an agent, as well as an Actionary. The Actionary is the connection between the database and your code, and stores and manages all the agents, objects, and actions within your program.

Before We Begin:

It's expected that you have the code installed. If you do not, please do that first (see the installation Readme).

Let's Begin:

Create a new program. For this tutorial, you will be exclusively writing in *c++*. To create a new project in Visual Studio, go to *File-¿New Project*. Since this is a basic example, a main.cpp is created. At the top of our code, we need to import a few libraries, specifically, the agent process and Actionary. Place this at the top of your code:

```
#include "agentproc.h"
```

This is the include files needed to get a basic agent up and running. agentproc.h holds the AgentProc class that builds the agent and itself includes all of the other files needed.

Next, we create some global variables to control the agent and Actionary.

```
extern Actionary *actionary; // global pointer to the Actionary
AgentProc* agent;  // global pointer to an agent
parTime *partime;  // global pointer to PAR time class (So we can manipulate time)
char* actionLocation=strdup("PATH TO FOLDER");  // Path to PAR folder
```

The Actionary and agent variables just point to the Actionary and agent. The Actionary pointer is defined in AgentProc.cpp and is required for any PAR application. Every entity in the virtual world that will be performing actions needs to be associated with an AgentProc. We would recommend creating a subclass of this class to store any additional information required for your application. This would include a pointer to the graphical representation of the agent in animated applications and specific AI. You can create lists of these agents and call their update methods systematically.

Partime allows us to set up timing information for the system. You can have simulations begin at different times of day and speed up the rate of time as well.

The last variable, actionLocation, points to the place PAR where is installed. It's used to give a starting location for the location of the Python action scripts (which is talked about in more detail in tutorial 2). **Do not forgot to change this to your action filepath**. Note that the action location requires the end slash of the path. So, if your

actions are on in a folder called *actions* on your root path, the path should be *C:*
actions
.

Next we move onto the main function. Within this function, we'll set the environment for the agent, and create a basic agent. The main function appears as:

```
int main(void){
        partime = new parTime();  // setup the timing info for the simulation
        partime->setTimeOffset(8,30,30);  // hrs., min., sec. from midnight
        partime->setTimeRate(1);          // how fast should time change

//Creates an actionary
        actionary=new Actionary();
        actionary->init();

        agent=new AgentProc("Agent_0");
        agent->setCapability("ROOT");

        system("PAUSE"); // just holds the output window
}
```

First, we create and set the partime variable to 8:30 am, and have it run at a constant rate of one per time step. Next, we create an Actionary. Finally, an agent is created with the name Agent_0. This agent is given the base capability 'ROOT'. It is important to give agents capabilities; otherwise the system won't know what the agent can and cannot do. By giving the highest level capability to the agent, it can perform all actions that are children of 'ROOT' (see the action tree diagram in the general documentation).

Before compiling and running this code, you'll need to add some project settings. To change these settings, please refer to Figure 1. Note that `${CONNECTOR_ROOT}/include`, `${PYTHON_ROOT}/include`, and `${MYSQL_ROOT}/include` use the environmental variables we set up earlier. The other include path, *PAR*, refers to the directory that PAR's codebase is installed in.

Additional Include Directories should include paths (using the environmental variables) to:

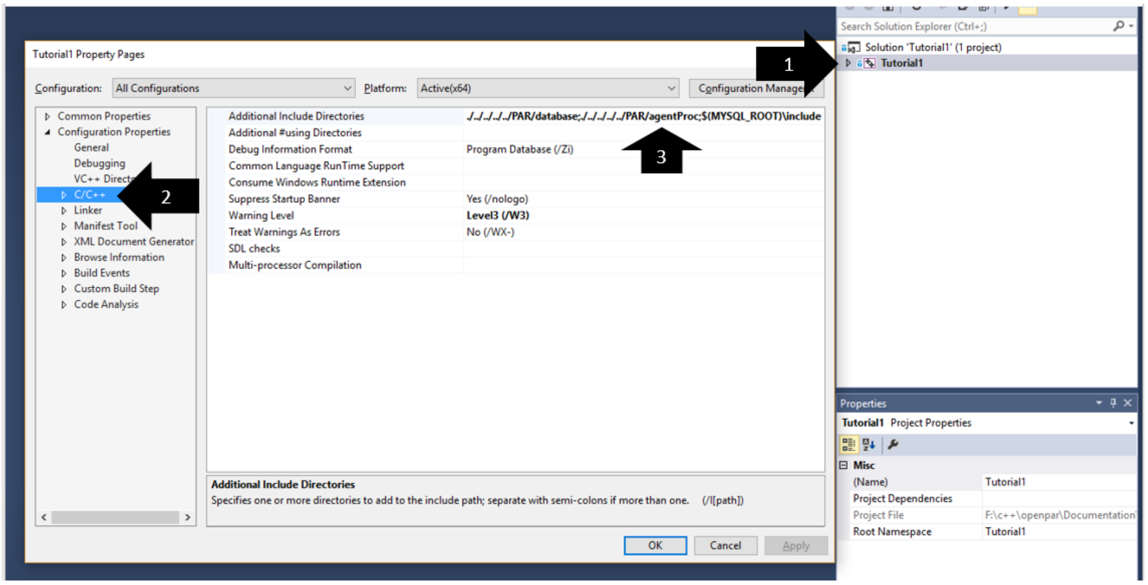
```
PAR/database
PAR/agentProc
PAR/lwnets
${CONNECTOR_ROOT}/include
${PYTHON_ROOT}/include
${MYSQL_ROOT}/include
```

Futhermore, we need to set up additional linker libraries. First, we need to set the paths. Please refer to Figure 2 to see where to add linker paths. *Additional Library Directories* should include paths to:

`PAR/libs`
`${CONNECTOR_ROOT}/lib/debug` (or `${CONNECTOR_ROOT}/lib/opt` if linking to the release libraries)

Finally, the actual dependencies need to be added. Please refer to Figure 3 for that.

Additional Dependencies (under Linker) should include:



Configuration: All Configurations
Platform: Active(x64)

- Common Properties
- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
  - Linker
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step
  - Code Analysis

Additional Include Directories	**./../../../../PAR/database;./../../../../PAR/agentProc;$(MYSQL_ROOT)\include**
Additional #using Directories	
Debug Information Format	Program Database (/Zi)
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	**Level3 (/W3)**
Treat Warnings As Errors	No (/WX-)
SDL checks	
Multi-processor Compilation	

Additional Include Directories
Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])

Solution 'Tutorial1' (1 project)
Tutorial1

Tutorial1 Project Properties

Misc

(Name)	Tutorial1
Project Dependencies	
Project File	F:\c++\openpar\Documentation
Root Namespace	Tutorial1

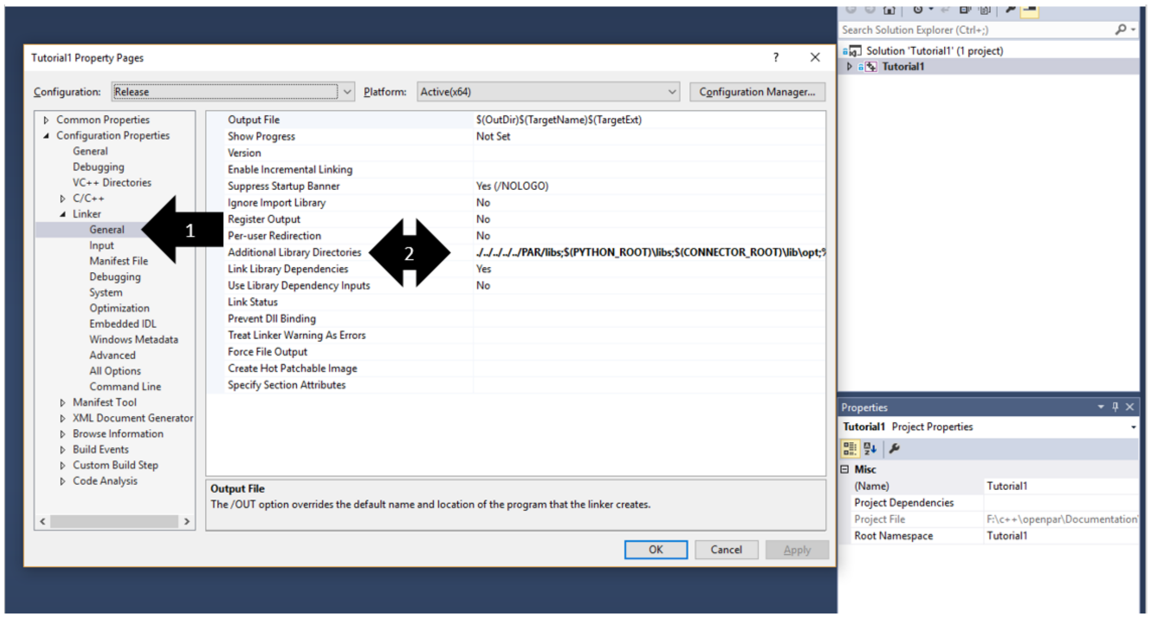
Figure 1: A screenshot of how to get to add includes

databased.lib
agentProcd.lib
lwnetd.lib
mysqlcppconn.lib
python27_d.lib
winmm.lib

Or the release versions, as appropriate. The release versions are:

database.lib
agentProc.lib
lwnet.lib
mysqlcppconn.lib
python27.lib
winmm.lib

You should now be able to compile and run this first tutorial. Note that we've provided libs and dlls for mysqlpp and python. However, these may not be compatible with different operating systems. If this tutorial cannot be run because of a dll conflict, try compiling your own versions of these files. (Also, remember that the three PAR solutions need to be compiled to generate the .lib files before this tutorial can be run). Furthermore, if mysql's



Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	./../../../../PAR/libs;$(PYTHON_ROOT)\libs;$(CONNECTOR_ROOT)\lib\opt;%
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Link Status	
Prevent Dll Binding	
Treat Linker Warning As Errors	
Force File Output	
Create Hot Patchable Image	
Specify Section Attributes	

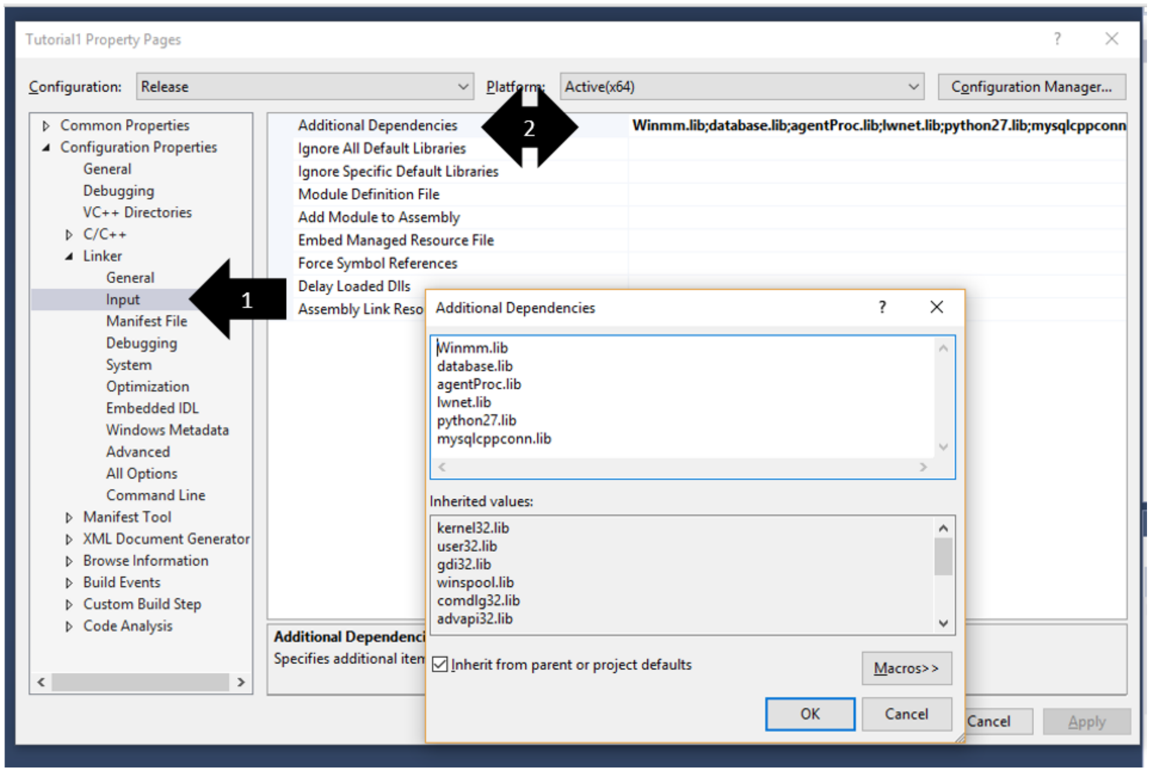
Output File
The /OUT option overrides the default name and location of the program that the linker creates.

(Name)	Tutorial1
Project Dependencies	
Project File	F:\c++\openpar\Documentation
Root Namespace	Tutorial1

Figure 2: A screenshot of adding linker library paths

c++ connector was installed using the community install, it will not have built debug libraries. So you will have to run your code in release mode.

This is a very simple start to help make sure that your setup is correct. Additional tutorials will highlight the functionality of PAR.



Configuration: Release

Platform: Active(x64)

Additional Dependencies: Winmm.lib;database.lib;agentProc.lib;lwnet.lib;python27.lib;mysqlcppconn

Additional Dependencies

```
Winmm.lib
database.lib
agentProc.lib
lwnet.lib
python27.lib
mysqlcppconn.lib
```

Inherited values:

```
kernel32.lib
user32.lib
gdi32.lib
winspool.lib
comdlg32.lib
advapi32.lib
```

☑ Inherit from parent or project defaults

Figure 3: A screenshot of adding linker library files