DDD 4.2
API Guide
Last Updated: May 10, 2012

Prerequisites:

- DDD 4.2 Installed
- Microsoft Visual Studio 2010 (or higher) Installed
- A working knowledge of the C# language

Overview:

The DDD API consists of a set of libraries used for accomplishing the following tasks:

- Establishing a network connection with the DDD Server
- Receiving/sending events from/to the DDD Server
- Using/constructing DDD Events

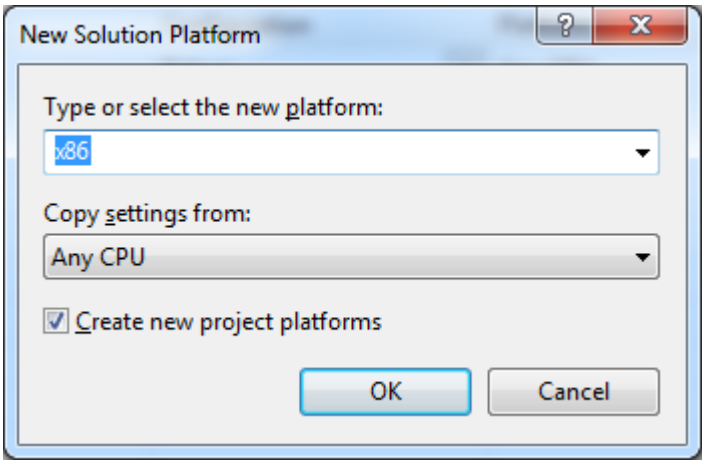
External References:

- DDD_API_Reference.html – To see detailed API documentation for the software libraries used in this guide
- DDD_Event_Reference.html – To see detailed documentation about the DDD events that you will have access to through using this API.
- DDDAPITestProject directory

Getting started:

1. Navigate to the "DDDAPITestProject" directory (created when you unzipped DDD_API_Docs.zip)
2. Double click on "DDDAPITest.sln" to open the solution file in Visual Studio 2010.
    a. NOTE: If you are using a more recent version of Visual Studio than 2010, it will ask you to convert the project.
3. Go the project References tree item, and select Add Reference. In the Dialog that comes up, select the Browse tab and then navigate to C:\\Program Files\Aptima\DDD 4.2\Server\ and then select and add the following DLLs:
    a. Aptima.Asim.DDD.CommonComponents.DataTypeTools.dll
    b. Aptima.Asim.DDD.CommonComponents.NetworkTools.dll
    c. Aptima.Asim.DDD.CommonComponents.SimulationEventTools.dll
    d. Aptima.Asim.DDD.CommonComponents.SimulationModelTools.dll
    e. Aptima.Asim.DDD.CommonComponents.ErrorLogTools.dll
    f. Aptima.Asim.DDD.CommonComponents.ServerOptionsTools.dll
4. Now right click on the solution in the Solution Explorer window of Visual Studio, and select Configuration Manager.
5. If under Active Solution Platform, it says anything besides "x86", click the dropdown and select <New…>.
    a. Select x86 from the first drop down

b. Ensure that "Create new project platforms" is checked off
c. Click OK



NOTE: The DDD libraries are build against the x86 target platform. This means that if you want to use these libraries for your own applications, that your application will need to target the same platform. If for some reason you need libraries targeting the x64 platform, please contact Aptima Support – support@aptima.com

6. Start the DDD Server application.
7. Within Visual Studio, open "Program.cs". Change "HOSTNAME" and Port to match the hostname and port on the DDD Server. (Note: Port is an integer and not enclosed in quotes). The DDD Server hostname and port information is displayed within the DDD Server application's main GUI.
8. Build the solution via the Build->Build Solution menu item or <F6>.

9. Click “Load Scenario” on the DDD Server.
10. Once the scenario has finished loading, begin running the DDD Agent Application by hitting <F5>
11. Click “Start Simulation” on the server.

Reference

Connecting to the DDD Server:

1. Create a NetworkClient object:

```
NetworkClient nc = new NetworkClient();
```

2. Connect to the DDD Server with the Connect method. The HOSTNAME and PORT will depend on your DDD Server installation. An easy way to find the correct values is to run the DDD Server. Hostname and port are displayed in the upper-left section of the interface. Connect will return a Boolean value, which can be used to determine the success of the connect call.

```
nc.Connect(hostname, port);
```

3. Subscribe to events. For a comprehensive list of events, see the DDD_Event_Reference.html document. The RevealObject event will be sent when a new object is present on the map. The MoveObject event is sent when an object starts to move, either at the request of a player or because the object was scripted in the scenario file.

```
nc.Subscribe("RevealObject");
nc.Subscribe("MoveObject");
nc.Subscribe("TimeTick");
```

Constructing and sending an event:

1. Read the simulation model, so that you know what event types are available, and what their parameters are.

```
SimulationModelReader smr = new SimulationModelReader();
SimulationModelInfo simModel =
smr.readModel("C:\\Users\\Public\\DDDClient\\SimulationModel.xml");
```

2. Build an event using the SimulationEventFactory utility class

```
SimulationEvent ev = SimulationEventFactory.BuildEvent(ref simModel,
"ExternalApp_Log");
```

3. Populate the event parameters

```
((StringValue)ev["AppName"]).value = "DDDAPITest";
((StringValue)ev["LogEntry"]).value = "Test app connected to the
server";
```

4. Send the event using the NetworkClient

```
nc.PutEvent(ev);
```

Receiving and using events:

1. Create a list for storing received events

```
List<SimulationEvent> events = null;
```

2. The program must poll for events periodically. The rest of the steps in this section should be contained in a while loop that continues until the DDD Server shuts down the network connection. At the end of the loop, the application "Sleeps" for 50 milliseconds. This is key to ensure that the application gives the rest of the applications on your machine a chance to use some processor cycles, otherwise that loop would run indefinitely just consuming your entire processing power.

```
while (nc.IsConnected())
```

3. Receive all available events

```
events = nc.GetEvents();
```

4. Do something with the event. You can break the event apart and inspect the parameters, or you can print out the whole event using the SimulationEventFactory XMLSerialize mothod.

```
foreach (SimulationEvent e in events)
{
	switch (e.eventType)
	{
		case "RevealObject":
			Console.WriteLine(String.Format("Object {0}
		revealed",((StringValue)e["ObjectID"]).value));
			break;
		case "MoveObject":
			Console.WriteLine(String.Format("Object {0} moved",
		((StringValue)e["ObjectID"]).value));
			break;
		case "TimeTick":
			Console.WriteLine(String.Format("Received: {0}",
		SimulationEventFactory.XMLSerialize(e)));
			break;
	}
}
```

Time For Something More Complex:
There are a few functions at the bottom of the code which do some more complex capabilities with the events.