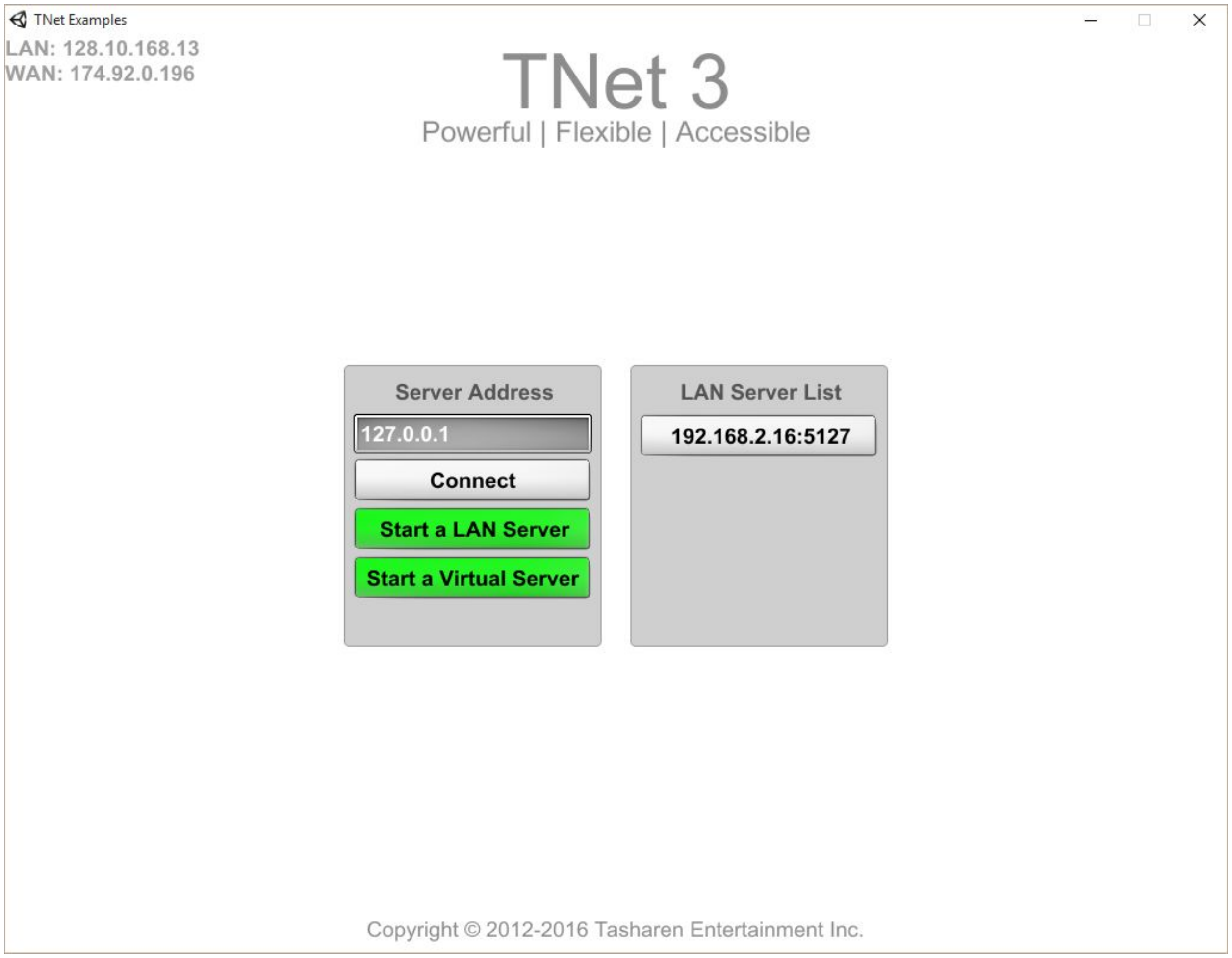
TNet 3: Main Menu In-Depth



The Main menu is not an example per se, but it demonstrates creating a lobby server nicely. Take a look at the OnGUI in `ExampleMenu.cs`. It's on the "Main Menu" GameObject in the scene.

```
void OnGUI ()
{
	if (!TNManager.isConnected)
	{
		DrawConnectMenu();
	}
	else
```

If there is no connection yet, `DrawConnectMenu` is called. This displays the rectangle containing the IP address as well as the connect and start server buttons. Let's take a look at this function from the bottom up.

```
// Start a local server that doesn't use sockets. It's ideal for testing and
for single player gameplay.
if (GUILayout.Button("Start a Virtual Server", button))
{
	mMessage = "Server started";
	TNServerInstance.Start("server.dat");
	TNManager.Connect();
}
```

When pressed this button will create a new local `TNServerInstance`. It behaves exactly the same as a full fledged network connection, but it doesn't actually open any ports. This is ideal for making a single player game or doing the single player campaign in a game which also has multiplayer.

```
if (GUILayout.Button("Start a LAN Server", button))
{
#if UNITY_WEBPLAYER
	mMessage = "Can't host from the Web Player due to Unity's security
	restrictions";
#else
	// Start a local server, loading the saved data if possible
	// The UDP port of the server doesn't matter much as it's optional,
	// and the clients get notified of it via Packet.ResponseSetUDP.
	int udpPort = Random.Range(10000, 40000);
	TNLobbyClient lobby = GetComponent<TNLobbyClient>();

	if (lobby == null)
	{
		if (TNServerInstance.Start(serverTcpPort, udpPort, "server.dat"))
			TNManager.Connect();
	}
	else
	{
		TNServerInstance.Type type = (lobby is TNUdpLobbyClient) ?
		TNServerInstance.Type.Udp : TNServerInstance.Type.Tcp;

		if (TNServerInstance.Start(serverTcpPort, udpPort,
		lobby.remotePort, "server.dat", type))
			TNManager.Connect();
	}
	mMessage = "Server started";
#endif
}
```

Moving up in the OnGUI function the “Start a LAN Server” button will create a full fledged server.

When pressed it will open up a server on a specific port. If the `TNLobbyClient` component exists (which it does in this example) it not only creates a server and connects to it, but it also informs the lobby that it exists.

A Server Lobby allows for a server list to be displayed to the user and in this example it will show up as a list of IP addresses to the right of the Connect and Start Server buttons. This example uses a UDP based lobby, which is good for searching the Local Area Network for fellow gamers, but in other types of games, such as Windward, you would want to use a TCP based lobby on a central server. TNet comes with the script `TNTcpLobbyClient` script to do this.

Near the very bottom of `DrawConnectMenu` there is a call to `DrawServerList`. This is what draws the various servers that the Lobby server knows about.

```
// List of discovered servers
List<ServerList.Entry> list = TNLobbyClient.knownServers.list;

// Server list example script automatically collects servers that have recently
announced themselves
for (int i = 0; i < list.size; ++i)
{
      ServerList.Entry ent = list[i];

      // NOTE: I am using 'internalAddress' here because I know all servers are
hosted on LAN.
      // If you are hosting outside of your LAN, you should probably use
'externalAddress' instead.
      if (GUILayout.Button(ent.internalAddress.ToString(), button))
      {
            TNManager.Connect(ent.internalAddress, ent.internalAddress);
            mMessage = "Connecting...";
      }
}
```

The `TNUdpLobbyClient` and `TNTcpLobbyClient` scripts both inherit from a class named `TNLobbyClient`. TNLobbyClient contains a series of static functions which can be used to find out information on the servers it knows about.

`DrawServerList` starts off by taking advantage of those static functions and queries the list of known servers. It then loops through them and creates buttons using the IP address as the text. When clicked, it also uses the IP address and port number to connect to that server.

Back in `DrawConnectMenu` near the top is where the regular connect button is located. It connects in much the same way as the server list buttons, but reads a text field for the IP address instead of an internal memory structure.

```
GUILayout.Label("Server Address", text);
mAddress = GUILayout.TextField(mAddress, input, GUILayout.Width(200f));

if (GUILayout.Button("Connect", button))
{
	// We want to connect to the specified destination when the button is 
clicked on.
	// "OnConnect" function will be called sometime later with the result.
	TNManager.Connect(mAddress);
	mMessage = "Connecting...";
}
```

If pressed, `TNManager.Connect` is called with the value of the text field. A few seconds the `TNManager.onConnect` event will fire. It has been registered in the parent class `TNEventReceiver` which is overridden by the `OnConnect()` function.

```
protected override void OnConnect (bool success, string message)
{
	Debug.Log("Connected: " + success + " " + message + " (Player ID #" + 
TNManager.playerID + ")");
	mMessage = message;

	// Make it possible to use UDP using a random port
	if (!TNServerInstance.isLocal) TNManager.StartUDP(Random.Range(10000, 
50000));
}
```

If the connection is not a local connection then it will also open up a series of UDP based ports for the example scenes to use. This way multiple clients on the same computer won't try to use the same port.

That's the basics of connecting and starting servers as well as how to start up a lobby.