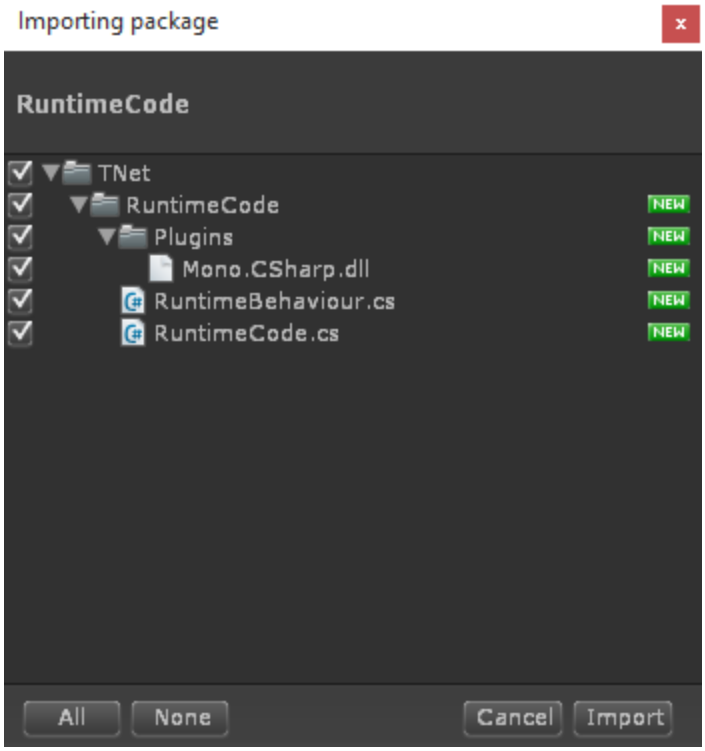
Executing Runtime C# Code

Running code at runtime can be useful for testing features which haven't been pushed live, trolling players by manually spawning enemies around someone in particular, gifting inventory to people, increasing everyone's health on the server, etc. TNet enables this with ease and it's exceptionally powerful.

Being powerful also means that the functionality can be dangerous if you're not careful. A remote user could execute code you never expected ruining the experience. For this reason this feature is included, but is disabled by default.

Installation

- Enable RuntimeCode by double clicking the RuntimeCode unity package in the `\TNet\Extras` folder.
- Import the DLL and all of the C# classes.



Trying it out

The easiest way to try this functionality out is by running the “0. Menu” scene, start up a server, and load the Chat example.

In a chat message type these:

```
/exe Debug.Log("My runtime code");
/exe TNManager.SetServerData("myRuntimeVariable = 5");
```

Debug.Log will write “My runtime code” to your console and the SetServerData will add a variable to the data available to everyone. If you then type /get the list will display all of the variables on the server including `myRuntimeVariable`.

How it works

To understand how it works, open ExampleChat.cs and Scroll down to the `Send()` function.

```
else if (mInput.StartsWith("/exe "))
{
	// Longer version, won't cause compile errors if RuntimeCode is not
imported
	var type = System.Type.GetType("TNet.RuntimeCode");
	if (type != null) type.Invoke("Execute", mInput.Substring(5));
	else Debug.LogError("You need to import the RuntimeCode package first");

	// Shorter version:
	//RuntimeCode.Execute(mInput.Substring(5));
}
```

If the line of text typed starts with `"/exe "` it checks to see if `TNet.RuntimeCode` exists. If it finds the class, that means that RuntimeCode.cs exists in the code base and one of its static functions can be called. The `Execute()` function is called and it parses the C# code as if it were typed in the code.

RuntimeCode.cs also includes another function called `ExecuteFile()`. It opens a local file and executes its contents. This allows you to write more involved code without trying to force it all onto one line.

Keeping it secure

One basic way to keep this functionality secure, is to send the command over the network to the server using a RFC. Once on the server, you can search the code for “bad” keywords. If any are found, ignore the commands. This is by no means the best or only way to do this. It’s just a helpful example.

```
[RFC] void ExecuteCode (string code)
{
	// For security purposes, don't allow reading of files or execution of
more code
	if (code.IndexOf("Directory") != -1) return;
	if (code.IndexOf("File") != -1) return;
	if (code.IndexOf("Reader") != -1) return;
	if (code.IndexOf("Stream") != -1) return;
	if (code.IndexOf("Assembly") != -1) return;
	if (code.IndexOf("Evaluator") != -1) return;
	if (code.IndexOf("Evaluate") != -1) return;
	if (code.IndexOf("SetAdmin") != -1) return;
	if (code.IndexOf("BeginSend") != -1) return;
	if (code.IndexOf("EndSend") != -1) return;
	if (code.IndexOf("BeginPacket") != -1) return;
	if (code.IndexOf("EndPacket") != -1) return;
	RuntimeCode.Execute(code);
}
```

If any file reading or network packet code is found, just drop out silently. This combined with any other security which you program into the game is what can help make it secure.