Windows Test Passes for Console

Overview

Every night, we run a set of automated test passes in the Windows engineering system for the console host code. This process is orchestrated on our working branch, which at the time of this writing is `RS_ONECORE_DEP_ACIDEV` (and will soon switch back to `RS_ONECORE_DEP_ACIOSS` or something of that ilk).

You can find the information about our nightly build, including these test passes, at the website [https://es.microsoft.com], choosing our branch, and then navigating to the `Execution Status` page.

At the bottom of the page will be the test pass runs for our tests that night. You can find out more about how these are set up in the [UniversalTest.md] file.

When a failure occurs in one of these passes, a bug will automatically be generated in the Azure DevOps project for the OS and assigned to our path.

The next step would be investigating one of these failures...

Investigation

A quick overview of investigation... normally you can just attempt to build and reproduce the failure locally with the `OpenConsole` project and it will happen the same way as it did on the nightly build in the lab. However, sometimes the failure will be exclusive to the lab or won't happen in the same way as it does on your local dev machine. At that point, you need to move into setting up the environment as it was during the testpass and figuring out what went wrong.

You can try to do this all manually by pulling down a VM image from the release share for the nightly build, making a VM, deploying the test binaries and TAEF test runner executables to the machine, installing the VS Remote Debugging or WinDBG tools on the VM, and then running the test and figuring out what's going wrong with the debuggers.

Or you can use some of the Engineering Systems tools to make this easier. I'll detail how to do that below.

Prerequisites: - Visual Studio 2017 - Install the TDP (Test Development Platform) plug-in (see: [https://osgwiki.com/wiki/Test_Development_Platform_(TDP)]).

1. Open Visual Studio 2017 and use the TDP drop-down menu to open the `Device Manager`.
2. In the pane that opens to the left, choose `Add` and then `Nebula VM Device`. Nebula is a cloud provider for VMs (like Azure but a more private instance for corporate work usage).
3. Name the machine and choose the build/branch/flavor/SKU from the drop downs at the bottom. It will find the VHD for you from the build shares. Hit `Add Device` to deploy to Nebula.

4. Wait a few minutes. It took 5-10 for it to be deployed.
5. Right click the machine name in the `Device Manager` list and choose `Launch T-Shell`. You can also use `Connect via Console` to get a "remote desktop"-like session to the KVM port on the VM.
6. In T-shell, use `testd Microsoft.Console.TestLab.Desktop.testlist` or a command of that format with a different TESTLIST or TESTMD name from our project (see the [UniversalTest.md] documentation). The `testd` utility will automatically resolve the build/branch/flavor information, dig through the build shares for the matching TESTLIST/TESTMD metadata, and attempt to deploy all relevant packages and dependencies on the device. When it's successful, it will move onto running all the tests and giving you the results. On conclusion, the test results should pop up in the web browser or the `Hubble - Log Viewer` tool provided by the Engineering Systems team.

If some of the above things do not work, go to [https://osgwiki.com] and type them into the Search bar. For instance, if T-Shell isn't found or working, you can find out where to get it or download it on `OSGWiki`. The same goes for the other commands besides `testd` to use in T-shell and more information on what `Hubble` or `Nebula` are.

Presumably now you have a failure. Or a success. You can attempt to spelunk the logs in `Hubble` and you might come to a conclusion. Or you can move onto debugging directly.

Now that you've relied on `testd` to get everything deployed and orchestrated and run once on the device, you can use `execd` to run things again or to run a smaller subset of things on the remote device through `T-Shell`.

By default, in the `Universal Test` world, everything will be deployed onto the remote machine at `C:\data\test\bin`. In T-Shell, use `cdd C:\data\test\bin` to change to that directory and then `execd te.exe Microsoft.Console.Host.FeatureTests.dll /name:*TestReadFileEcho*` to run just one specific test. Of course you should substitute the file name and test name parameters as makes sense. And of course you can find out more about `cdd` and `execd` on the `T-shell` page of `OSGWiki`.

Fortunately, running things through `T-shell` in this fashion is exactly the same way that the testlab orchestrates the tests. If you still don't get good data this way, you can use the `Connect via Console` mechanism way above to try to run things under `WinDBG` or the `Visual Studio Remote Debugger` manually on the machine to get them to repro or under the debugger more completely.