# CDInterfaceModule Developer Guide

## Purpose of the document

This document will describe the following details

* How to configure your setup for development of the CDInterfaceModule
* How to get a copy of the source code
* How to test the code
* How to release a new version of the code

Please refer to the following documents for related information

CDInterfaceModule Support Guide for more details on how to install the application.

CDInterfaceModule User Guide for more details on how to use the application.

CDInterfaceModule Design for more details on how to the code works within the application.

## Configure Environment

To develop the CDInterfaceModule you need a number of tools.

* Microsoft Visual Studio Code (VSC)
* Microsoft Powershell (PS)
* Microsoft Powershell plugin for VSC
* Pester plugin for VSC
* Git

In addition, to run the example java invocation code

* Java JDK (for example java code)
* Recommended Java plugins for VSC

Other tools can be used, but this set will be assumed in this document.

### Powershell

Powershell is included with Windows 10 and Windows 2019. The version in use during development of CDInterfaceModule is

Shape, rectangle

Description automatically generated

In some cases you will need to run commands in an Administrator Powershell session.

To open Powershell with Adminstrator privileges right click on the executable and select ‘Run as Administrator’. You do need to be in the Administrator group to do this successfully.

### Miscrosft Visual Studio Code

This is a code IDE, that is the easiest way to actually write the code. VSC supports the Powershell language fully via a Plugin provided by Microsoft. VSC will suggest appropriated plugins to use when you open code in the editor.

For development Microsoft Powershell plugin v2021.10.0 was used.

VSC can be downloaded from https://code.visualstudio.com/ . The version used during development was v1.61.0.

You should install the latest versions of VSC and associated Plugins.

### Pester

CDInterfaceModule includes Unit tests in the file CDInterfaceModule.Test.ps1.

To run these tests you need to install the Pester test tool, which is itself a Powershell module available from Powershell Gallery.

Refer to the Pester documentation on installing the tool <https://pester.dev/docs/quick-start>

During development v5 of Pester was used.

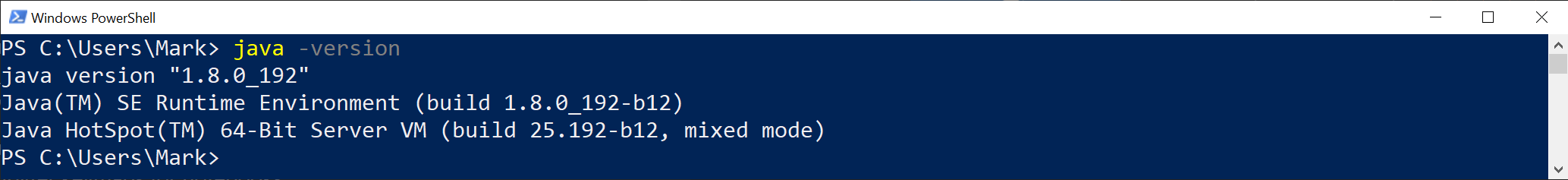
In addition to installing Pester itself, you should also install the Pester plugin from Pester for VSC via VSC’s plugins page. During development v2021.10.1 was used.

You should install the latest versions of Pester and the Pester plugin.

### Java JDK

In order to run the example java application and demonstrate the invocation of CDInterface, you need to have a JDK installed. The example application has minimal needs and should work in any recent java version.

During development Java v8 was used



### Git

Download and install the latest Git for Windows from <https://git-scm.com/download/win>

During development v2.33.1 was used.

You should install the latest available version.

## Getting the source

At the moment CDInterfaceModule is hosted in my personal github repository. It is expected that this will be migrated to the Corporate repository prior to a full production release.

To get a copy of the source locally, choose a local location in which to hold the code. For example I’ve used a mounted network drive mounte as Y:\ and created an ‘HPE Work’ directory in there. In a terminal window move to that location, and run

git clone <https://github.com/nosdod/CDInterface>

You should then have a CDInterface sub-directory.

The easiest way to bring up VSC in the right location, is to now

cd CDInterface

and invoke VSC with

code

You will now be on the ‘main’ branch of the code.

When working on a new version, you should be working on a branch from the ‘main’ version. Choose a branch name that reflects the nature of the change you are making. To checkout a version for development run,

git checkout -b ‘myNewVersion’

## Executing CDInterface in development

Once you have the environment configured, you should be able to execute CDInterface without installing it.

With the module, CDInterfaceModule.psm1 open in the editor, select the ‘Run and Debug’ icon on the left. The first time you run this on a project VSC will prompt you to create a launch.json run configuration file. Create the config using the ‘Powershell: Interactive Session’ option. Now when you select the ‘Run and Debug’ icon you will see a green triangle next to this launch configuration. Clicking the triangle runs the code.

Actually CDInterface hasn’t run yet …. In the PS Terminal window type

Import-Module ./CDInterfaceModule.psm1

Followed by

CDInterface -list

You should now see that CDInterface runs successfully.

### Autoloading

In order to make sure you are running the version of the module you are editing, you need to make sure the PS terminal session doesn’t have another version already loaded. This can happen for example on a system where you have installed CDInterfaceModule before. You also need to remove any installations, as these may get loaded in preference to the version you are editing.

The deploy.ps1 script has an option which will remove all the installed versions of the module, -cleanallonly. The -cleanonly option will just remove the current version.

To get rid of any in memory versions of the module use

Remove-module CDInterfaceModule

In the terminal PS session.

To confirm it is not loaded, and doesn’t autoload, run

CDInterface

You should see an error dialog

A screenshot of a computer

Description automatically generated with medium confidence

### Debugging CDInterface

With the above setup you can now set breakpoints in the source code, and these will be hit when you invoke CDInterface.

Once the code has hit a breakpoint you can step through code and inspect variables etc.

## Unit testing

A number of unit tests have been written to support CDInterface. These make use of Mocking to replace the actual interactions the code makes with physical drives via IMAPI.

The purpose of the unit tests is to exercise the code and confirm it behaves as required. Any changes to CDInterface will likely also require changes to the Unit Tests.

To run the tests click on the ‘Testing’ icon on the left. You should see ‘Pester’ listed in the panel next to the toolbar. Click on the white triangle to run the tests.

Pester runs in two main phases. First it will run over the tests in its Discover phase, after it’s done this it will display the hierarchy of tests. Then it will run each test in sequence.

Once you have let it discover and run the existing tests, you should then be able to, for example, re-run any individual tests using similar run icons that appear throughout the hierarchy. This is useful when you are developing or debugging a test.

To debug a test, use the icon to the right of the run icon. In this mode execution will break at any breakpoints set in either the test code or the application code.

## Integration testing

Whilst the unit tests will confirm CDnterface behaves as expected, it is also necessary to carry out a series of operations with an Installed CDInterfaceModule, to give full confidence that it works as intended. Integration testing should cover all the functionality used in production environments, and ideally ALL functionality.

Refer to the Integration Tests document for a full list of recommended tests.

## Releasing new versions

At the moment CDInterfaceModule is hosted in my personal github repository. It is expected that this will be migrated to the Corporate repository prior to a full production release.

The steps necessary to release a new version are similar for any GIT based repository.

When you are happy your changes are complete, associated tests succeed and the deploy script is updated, you are ready to release the new version.

Add all the changes

git add .

Commit the changes

git commit

Enter a short description of the changes, and exit the editor.

Push the changes back to Github.

git push origin HEAD

(Of course there is full Git integration in VSC so that can be used as well)

Once the changes are in Github, other people can potentially look at your new version and check it. In order for you to get full feedback you will need to produce a new versioned release, tagged with the version number.

To produce a new release, create a pull request in the Github browser UI to merge the branch to ‘main’. Tag it with the new version number. You can safely delete the branch at this point, any subsequent rework will need to become a new release of the module.

Release numbering follows the standard 3 part convention

<MajorNumber>.<MinorNumber>.<PatchNumber>

e.g. 1.0.1