Growing Your Pester Skills
Advanced Function Testing

Robert C. Cain, MVP
OWNER, ARCANE TRAINING AND CONSULTING

@arcanecode www.arcanecode.com

Introduction to Pester

Implementing Your First Pester Tests

Basic Function Testing with Pester

Requirements – Get-PodcastMedia

Accept data from Get-PodcastData as parameter named RSSData

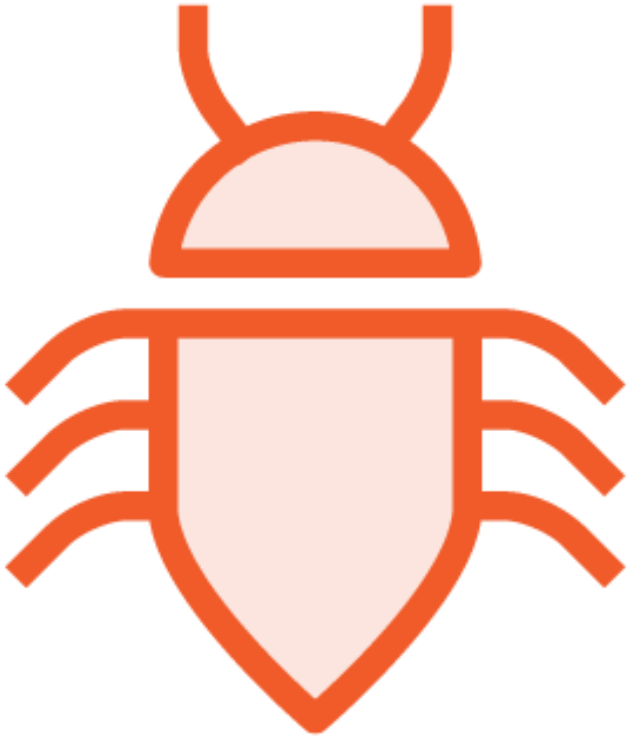
Use the AudioURL with Invoke-WebRequest to download the audio file

Store audio file in folder specified in parameter named OutputPathFolder, which will have a default value

Use the filename from the AudioURL as the file name in OutputPathFolder

Returns a list of the file names downloaded

Unit Tests – Get-PodcastMedia

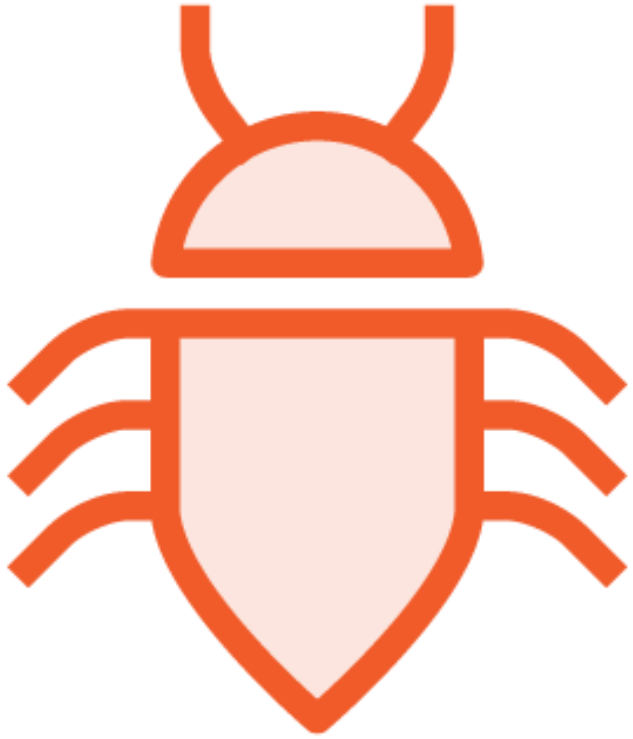


Use mocked, known data.

Validate each file from the RSS feed exists in the unit testing location indicated in the OutputPathFolder parameter

Validate each file name exists in the array of file names returned from the function

Acceptance Tests – Get-PodcastMedia



Use live data from Get-PodcastData.

Validate each file from the RSS feed exists in both the default value and a passed in value for the OutputPathFolder parameter

Validate each file name exists in the array of file names returned from the function

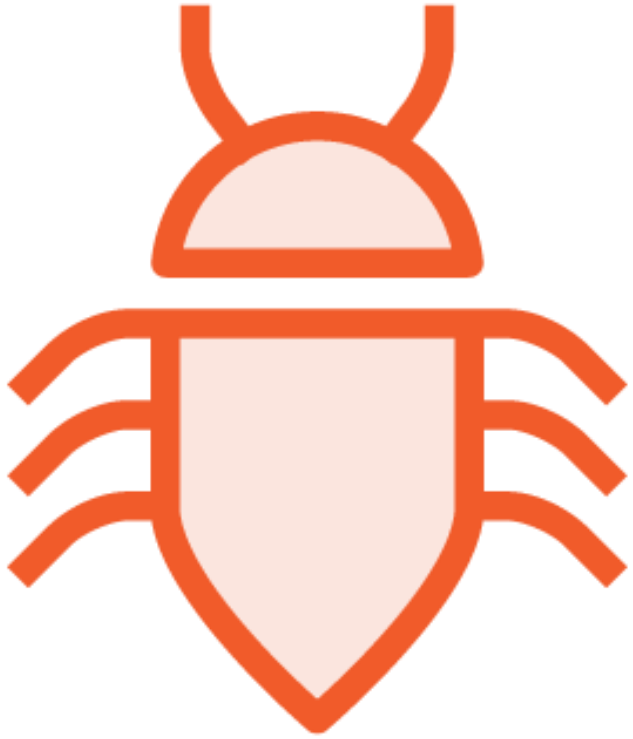
Requirements – ConvertTo-PodcastXML

Accept data from Get-PodcastData as pipeline enabled parameter named PodcastData

Returns an array of strings with XML data

```
<Show>
        <Title>…</Title>
        <Link>…</Link>
        <Hosts>…</Hosts>
        <PublicationDate>…</PublicationDate>
        <ImageURL>…</ImageURL>
        <ImageFileName>…</ImageFileName>
        <AudioURL>…</AudioURL>
        <AudioFileName>…</AudioFileName>
        <AudioFileLength>…</AudioFileLength>
    </Show>
```

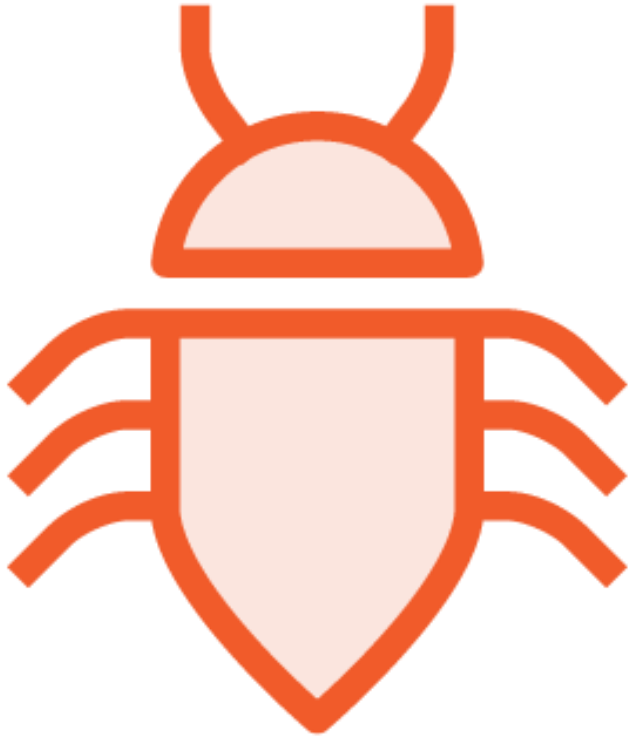
Unit Tests – ConvertTo-PodcastXML



Use mocked, known data.

Validate each value in the PodcastData exists as a value, within the correct tags, in the XML data

Acceptance Tests – ConvertTo-PodcastXML



Use live data from Get-PodcastData.

Validate each value in the PodcastData exists as a value, within the correct tags, in the XML data

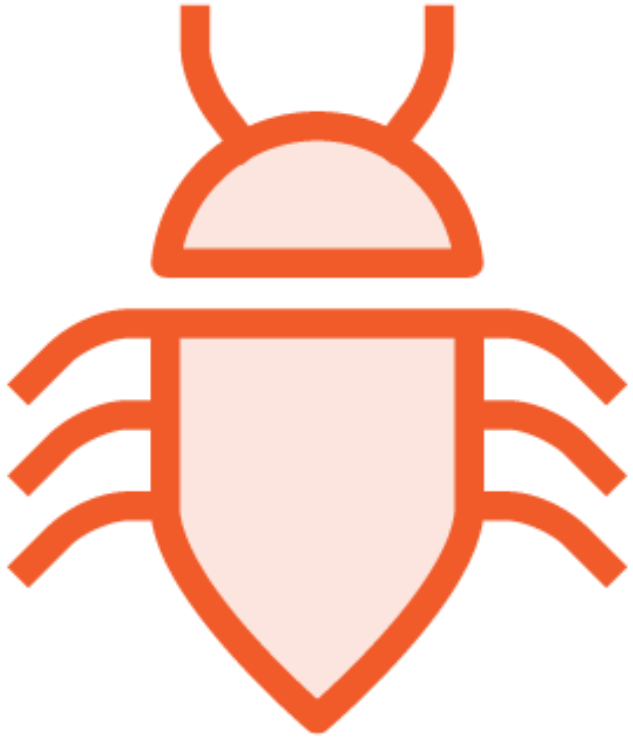
Requirements – Write-PodcastXML

Accept data from ConvertTo-XML in a parameter named XMLData. Pipelined.

XMLFilePath parameter for the location of the output file. Has a default value.

Create an XML file.

Unit Tests – Write-PodcastXML

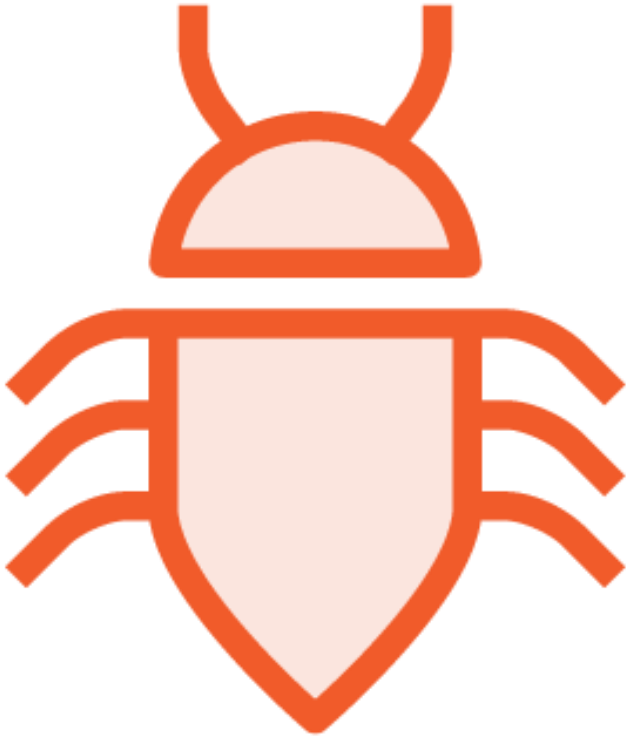


Use mocked, known data for XMLData.

Validate file was created XMLData was passed as a parameter

Validate file was created when XMLData was pipelined

Acceptance Tests – Write-PodcastXML



Use live data from Get-PodcastData and ConvertTo-PodcastXML

Ensure files are created using default and passed in values for XMLFilePath

Read in XML file and validate values against the data from Get-PodcastData

Requirements – Get-NoAgenda

Three Optional Parameters:

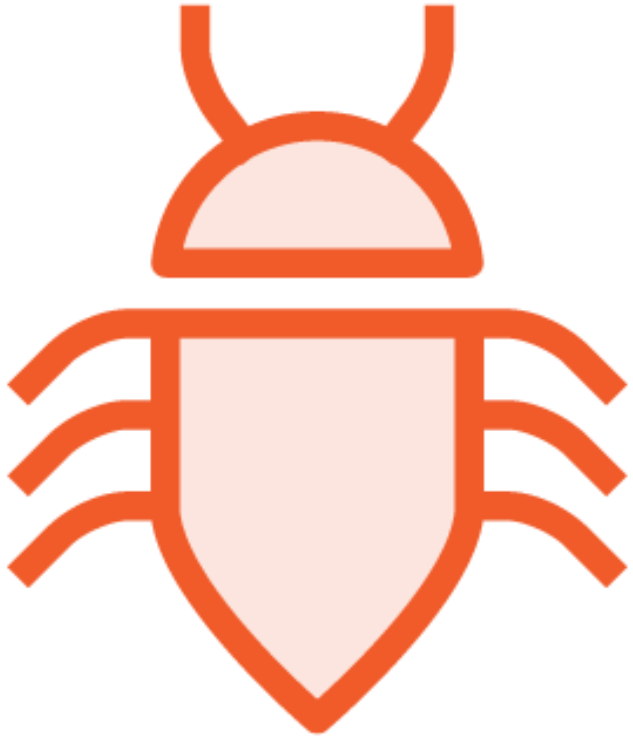
- OutputPathFolder
- XmlFileName
- HtmlFileName

Requirements – Get-NoAgenda

Call all functions

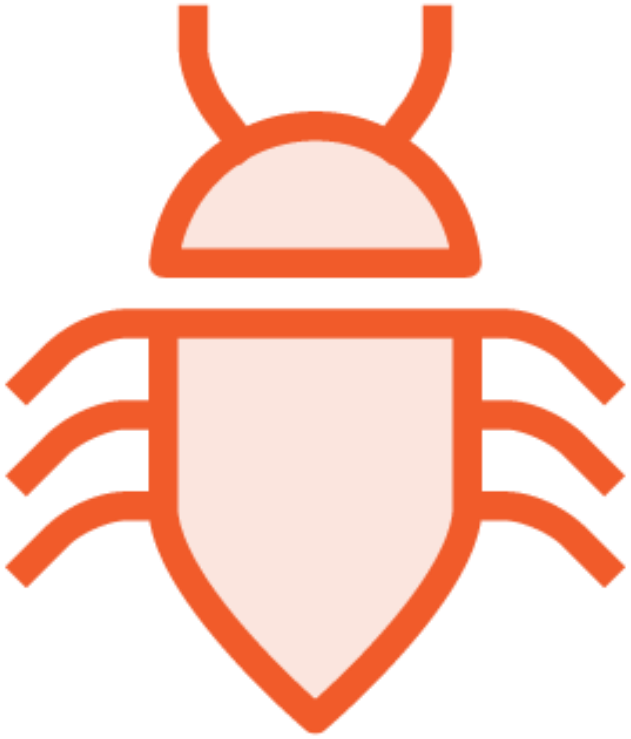
- Get-PodcastData
- Get-PodcastImage
- Get-PodcastMedia
- ConvertTo-PodcastHtml
- Write-PodcastHtml
- ConvertTo-PodcastXml
- Write-PodcastXml

Unit Tests – Get-NoAgenda



Verify Get-NoAgenda calls each of the functions listed in the requirements

Acceptance Tests – Get-NoAgenda



Verify the XML and HTML files were correctly created using the default values for the parameters

Verify the XML and HTML files were created when values are passed in for the parameters

Summary

- Dynamic Tests
- Create Mocked Functions
- Assert-MockCalled
- Integration Test