Contents

Introduction

Workshop 2 serves to introduce concepts of the EgmClientsLite library while exploring configuring an EGM from cold boot to playing a basic game. By the end of this workshop, you'll have created a test that uses EgmClientsLite to bypass cold boot configuration, enable each paytable on the machine, and play a game for each paytable. Throughout the workshop, we will create small tests that work towards this goal, and then bring them all together at the end. Please refer to the following resources from Workshop 2 and follow the 6 Parts of the workshop as documented below.

Workshop 2 Resources:

Workshop 2 PowerPoint

Workshop 2 Recording

EgmClientsLite Documentation

Part 1: Creating Your First Test to Enable a Game

This series of steps serves to introduce the concept of bypassing the cold boot and enabling a game.

1. Start by getting an EGM/AoW to a cold boot or cleared state. It should have the following tilt:
2. Open up your repository in Visual Studio or VS Code.
3. Create a new file/class within the Tests folder of your repository.

   - For VS Code: In the Explorer  view of your repository, right-click on the Tests folder, then select New File.
   - For Visual Studio Professional: In the Solution Explorer, right-click on the Tests folder, then select Add -> New Item.

VS Code	Visual Studio Professional 2022

4. Name this new file ValidateAllConfigsTests.cs

5. If your new file has any code in it upon creation, go ahead and delete it to start from scratch. Add a using statement to include BugSweeper.Tests. Then add the [TestFixture] attribute before defining the class. This allows our Test Explorer to look for tests in this class. Let this class inherit from EgmTestBase. Your class should look like this:

```
using BugSweeper.Tests;

[TestFixture]
internal class ValidateAllConfigsTests : EgmTestBase
{

}
```

6. Now we want to create a SetUp() method inside this class and mark it with the [SetUp] attribute. The [SetUp] attribute makes this method run before every test in the class.

```
using BugSweeper.Tests;

[TestFixture]
internal class ValidateAllConfigsTests : EgmTestBase
{
    [SetUp]
    public void SetUp()
    {

    }
}
```

7. For this scenario, we want our SetUp() to bypass the cold boot configuration pages. We can look at the available functionalities in EgmClientsLite here: EgmClientsLite Documentation.

    a. Open the wiki to explore EgmClientsLite functionalities.

    b. When looking at what's available, everything will chain from AscentEgm, AscentEgmApi, and Proxy (CommonEgm, EgmApi). Search for *Cold*Boot. You'll need to use wildcards/asterisks to search.

    c. Take a look at the ColdBoot class that shows up as a result. Within this class, there is a BypassColdBootConfig() method. That's what we'll want to use in our SetUp().

    d. Go back into your ValidateAllConfigsTests class. Let's call this method from inside our SetUp() method.

```
using BugSweeper.Tests;

[TestFixture]
```

```
internal class ValidateAllConfigsTests : EgmTestBase
{
    [SetUp]
    public void SetUp()
    {
        Egm.Configure.ColdBoot.BypassColdBootConfig();
    }
}
```

8. Our SetUp is complete! Now we can create our first test. Create a method called EnableFirstPaytableFromColdBoot() and mark it with the [Test] attribute above the method. This test serves to showcase how quickly we can enable a single game from a cold boot state.

```
using BugSweeper.Tests;

[TestFixture]
internal class ValidateAllConfigsTests : EgmTestBase
{
    [SetUp]
    public void SetUp()
    {
        Egm.Configure.ColdBoot.BypassColdBootConfig();
    }

    [Test]
    public void EnableFirstPaytableFromColdBoot()
    {

    }
}
```

9. We'll need to define which theme we want to target here. We can just use the first theme for now. Let's create a variable called firstTheme and set it equal to the first theme on our EGM.

```
using BugSweeper.Tests;

[TestFixture]
internal class ValidateAllConfigsTests : EgmTestBase
{
    [SetUp]
    public void SetUp()
    {
        Egm.Configure.ColdBoot.BypassColdBootConfig();
    }

    [Test]
```

```
    public void EnableFirstPaytableFromColdBoot()
    {
        var firstTheme = Egm.Themes.First();
    }

}
```

10. We'll also need to define which paytable we want to target. Again, we can just use the first paytable in the theme. Create another variable called firstPaytable and set it equal to the first game variation in our theme (game variation is synonymous with paytable here).

```
using BugSweeper.Tests;

[TestFixture]
internal class ValidateAllConfigsTests : EgmTestBase
{
    [SetUp]
    public void SetUp()
    {
        Egm.Configure.ColdBoot.BypassColdBootConfig();
    }

    [Test]
    public void EnableFirstPaytableFromColdBoot()
    {
        var firstTheme = Egm.Themes.First();
        var firstPaytable = firstTheme.GameVariations.First();
    }
}
```

11. One last thing we need for enabling this paytable is a denomination. We'll just use the first supported denom for this game. We can enable multiple denoms (e.g., [1, 10, 50, 100]), but for this scenario, we'll only do the one.

```
using BugSweeper.Tests;

[TestFixture]
internal class ValidateAllConfigsTests : EgmTestBase
{
    [SetUp]
    public void SetUp()
    {
        Egm.Configure.ColdBoot.BypassColdBootConfig();
    }

    [Test]
    public void EnableFirstPaytableFromColdBoot()
```

```
    {
        var firstTheme = Egm.Themes.First();
        var firstPaytable = firstTheme.GameVariations.First();
        firstPaytable.EnabledDenominations = [firstPaytable.Denominations.First()];
    }
}
```

15. Your first test is complete! Run this test. You should see your EGM/AoW enable a game without having to manually configure the machine. For help with running tests, see the How to run tests in vs code wiki page. —|— ## Part 2: Validating a Game This series of steps serve to introduce concepts of reusing code (GameUtilities.PlayOneGame).

16. You will need a few more files for this next step. You can find those here: `\\engfile6\transfer\PublicDrop\BugSweeper\Series2Files`. Copy those files (EgmMenuExtensions, GameUtilities) into the Tests folder of your repository.

17. We'll want to add these to our using statements in our ValidateAllConfigsTests class.

```
using BugSweeper.Tests;
using Utilities;
using global::Tests.EgmExtensions;
```

3.Now we're ready to create our next test. Create a method called ValidateCurrentGame() and mark it with the [Test] attribute.

4. One way we can validate a game is by playing it. We provided you with a PlayOneGame() method in that GameUtilities file. We can call it from inside our test. PlayOneGame() will need the Egm parameter passed into it. We can hover our cursor over it to see what parameter(s) it requires.

```
[Test]
public void ValidateCurrentGame()
{
    var beforeScPath =
        "C://Screenshots//" +
        Egm.ActiveGame.Name + "_" +
        Egm.ActiveGame.Code + "_before.png";
    var afterScPath =
        "C://Screenshots//" +
        Egm.ActiveGame.Name + "_" +
        Egm.ActiveGame.Code + "_after.png";

    GameUtilities.TakeScreenshot(Egm, beforeScPath);
    GameUtilities.PlayOneGame(Egm);
    GameUtilities.TakeScreenshot(Egm, afterScPath);
```

```
        Assert.That(File.Exists(beforeScPath), Is.True);
        Assert.That(File.Exists(afterScPath), Is.True);
    }
```

* If you are working with a Poker Game, you may need to make a small adjustment to PlayOneGame(). Right now it presses the "Spin" button once, but you need it to press it twice for Deal/Draw. In the GameUtilities class, you can uncomment these two lines. Then save this file.

5. Run the ValidateCurrentGame() test. No need to add credits – PlayOneGame() will do that for us. You should see one game play on your EGM.

—|— ## Part 3: Validating A Game With Screenshots 1. Another way we can validate a game is by capturing screenshots. Let's add that functionality to your ValidateCurrentGame() test. We'll want to take a screenshot before and after we play a game. First, we need to define where we want to save these screenshots. Create a new varable called beforeScPath and set it equal to the path to save this screenshot. In this example, we have it save to C:/Screenshots. This folder will be created when the test is run – no need to create it! Create another variable called afterScPath for the second screenshot. Make sure to modify the path so the screenshots have different names:

```
    [Test]
    public void ValidateCurrentGame()
    {
        var beforeScPath =
            "C://Screenshots//" +
            Egm.ActiveGame.Name + "_" +
            Egm.ActiveGame.Code + "_before.png";
        var afterScPath =
            "C://Screenshots//" +
            Egm.ActiveGame.Name + "_" +
            Egm.ActiveGame.Code + "_after.png";

        GameUtilities.TakeScreenshot(Egm, beforeScPath);
        GameUtilities.PlayOneGame(Egm);
        GameUtilities.TakeScreenshot(Egm, afterScPath);

        Assert.That(File.Exists(beforeScPath), Is.True);
        Assert.That(File.Exists(afterScPath), Is.True);
    }
```

2. Now we have to add commands to actually take the screenshots. There's a TakeScreenshot() method in the GameUtilities class you downloaded. You can use that here. Make sure to fill in the parameters it requires: Egm and the path of the screenshot that you just defined.

```
    [Test]
```

```
public void ValidateCurrentGame()
{
    var beforeScPath =
        "C://Screenshots//" +
        Egm.ActiveGame.Name + "_" +
        Egm.ActiveGame.Code + "_before.png";
    var afterScPath =
        "C://Screenshots//" +
        Egm.ActiveGame.Name + "_" +
        Egm.ActiveGame.Code + "_after.png";

    GameUtilities.TakeScreenshot(Egm, beforeScPath);
    GameUtilities.PlayOneGame(Egm);
    GameUtilities.TakeScreenshot(Egm, afterScPath);

    Assert.That(File.Exists(beforeScPath), Is.True);
    Assert.That(File.Exists(afterScPath), Is.True);
}
```

3. Before we run this test, we should also add an Assert to check that the files saved properly. We want to check (or "assert") that the file exists in the path we defined. We'll do this for both files:

```
[Test]
public void ValidateCurrentGame()
{
    var beforeScPath =
        "C://Screenshots//" +
        Egm.ActiveGame.Name + "_" +
        Egm.ActiveGame.Code + "_before.png";
    var afterScPath =
        "C://Screenshots//" +
        Egm.ActiveGame.Name + "_" +
        Egm.ActiveGame.Code + "_after.png";

    GameUtilities.TakeScreenshot(Egm, beforeScPath);
    GameUtilities.PlayOneGame(Egm);
    GameUtilities.TakeScreenshot(Egm, afterScPath);

    Assert.That(File.Exists(beforeScPath), Is.True);
    Assert.That(File.Exists(afterScPath), Is.True);
}
```

4. Run your updated ValidateCurrentGame() test. It should take a screenshot, play a game, then take another screenshot. Open your C:/Screenshots folder to see the screenshots.

—|— ## Part 4: Viewing Test Output This series of steps serve to showcase how

to iterate through multiple themes and games to do something.

1. Create another test. Add the [Test] attribute and name this one EnableEachGameOneByOne(). This test will do exactly that: Go through each theme and enable each paytable, one by one.

```
[Test]
public void EnableEachGameOneByOne()
{

}
```

2. Create a loop that will go through each theme on the EGM.

```
[Test]
public void EnableEachGameOneByOne()
{
    foreach(var theme in Egm.Themes)
    {

    }
}
```

3. Inside the loop you just created, create another loop that will go through each paytable for the theme.

```
[Test]
public void EnableEachGameOneByOne()
{
    foreach(var theme in Egm.Themes)
    {
        foreach(var paytable in theme.GameVariations)
        {

        }
    }
}
```

4. Inside the loops, we'll add a command that prints information out to our test explorer. Use Console.WriteLine() to print out information. For this example, we'll print out the theme name, the paytable code, and the available denoms for that paytable. By putting it inside these loops, it will print this out for every paytable for each theme.

```
[Test]
public void EnableEachGameOneByOne()
{
    foreach(var theme in Egm.Themes)
    {
        foreach(var paytable in theme.GameVariations)
```

```
            {
                Console.WriteLine(
                    $"Theme: {theme.Name}, Paytable: {paytable.Code}, " +
                    $"Available Denom: {string.Join(", ", paytable.Denominations)}"
                );
            }
        }
    }
```

5. Run this test and view the output. You can view this in VS Code by opening the "Test Results" section in place of the Terminal or in Visual Studio Professional by clicking on the test in the Test Explorer.

VS Code	Visual Studio Professional 2022

—|— ## Part 5: Enable Each Game One By One 1. Now we'll add to the EnableEachGameOneByOne() test to complete the desired functionality. We set up the two loops to iterate though each theme and each paytable per theme. Now, we need to enable a paytable inside these loops. We'll do this right below our Console.WriteLine() from the previous step. First we need to enter the Operator menu. We can use the EnterOperatorMenu() method that exists in EgmClientsLite:

```
    [Test]
    public void EnableEachGameOneByOne()
    {
        foreach(var theme in Egm.Themes)
        {
            foreach(var paytable in theme.GameVariations)
            {
                Console.WriteLine(
                    $"Theme: {theme.Name}, Paytable: {paytable.Code}, " +
                    $"Available Denom: {string.Join(", ", paytable.Denominations)}"
                );

                Egm.EnterOperatorMenu();
            }
        }
    }
```

2. Once we've entered the operator menu, we want to enable the game. We'll need to define the denom we want to activate and set it as an enabled denomination, just as we did in Step 11 of Part 1. Again, we can just use the first supported denom for the paytable.

```
[Test]
public void EnableEachGameOneByOne()
{
    foreach(var theme in Egm.Themes)
    {
        foreach(var paytable in theme.GameVariations)
        {
            Console.WriteLine(
                $"Theme: {theme.Name}, Paytable: {paytable.Code}, " +
                $"Available Denom: {string.Join(", ", paytable.Denominations)}"
            );

            Egm.EnterOperatorMenu();
            var denomToActivate = paytable.Denominations.First();
            paytable.EnabledDenominations = [denomToActivate];
        }
    }
}
```

3. Now we can exit the Operator menu. Just like entering the menu, there is an ExitOperatorMenu() method in EgmClientsLite.

```
[Test]
public void EnableEachGameOneByOne()
{
    foreach(var theme in Egm.Themes)
    {
        foreach(var paytable in theme.GameVariations)
        {
            Console.WriteLine(
                $"Theme: {theme.Name}, Paytable: {paytable.Code}, " +
                $"Available Denom: {string.Join(", ", paytable.Denominations)}"
            );

            Egm.EnterOperatorMenu();
            var denomToActivate = paytable.Denominations.First();
            paytable.EnabledDenominations = [denomToActivate];
            Egm.ExitOperatorMenu();
        }
    }
}
```

4. Once we're out of the Operator menu, we'll want to activate the paytable and denom that we just enabled to open the game on our EGM. We'll use the Activate method for the paytable and pass in the denom we defined:

```
[Test]
public void EnableEachGameOneByOne()
```

```
{
    foreach(var theme in Egm.Themes)
    {
        foreach(var paytable in theme.GameVariations)
        {
            Console.WriteLine(
                $"Theme: {theme.Name}, Paytable: {paytable.Code}, " +
                $"Available Denom: {string.Join(", ", paytable.Denominations)}"
            );

            Egm.EnterOperatorMenu();
            var denomToActivate = paytable.Denominations.First();
            paytable.EnabledDenominations = [denomToActivate];
            Egm.ExitOperatorMenu();
            paytable.Activate(denomToActivate);
        }
    }
}
```

5. Run this test. You should see it quickly enter and exit the Operator menu then load up the newly enabled paytable. It will do this repeatedly until it covers all paytables for all themes on your EGM. If you have a lot of themes on your machine, you may wish to stop the test early.

—|— ## Part 6: Putting It All Together 1. Let's create one more test to combine what we've done. Add the [Test] attribute, then name this new test ValidateAll-Games().

```
[Test]
public void ValidateAllGames()
{

}
```

2. Copy the contents of EnableGameOneByOne into this new test.

```
[Test]
public void ValidateAllGames()
{
    foreach(var theme in Egm.Themes)
    {
        foreach(var paytable in theme.GameVariations)
        {
            Console.WriteLine(
                $"Theme: {theme.Name}, Paytable: {paytable.Code}, " +
                $"Available Denom: {string.Join(", ", paytable.Denominations)}"
            );
```

```
            Egm.EnterOperatorMenu();
            var denomToActivate = paytable.Denominations.First();
            paytable.EnabledDenominations = [denomToActivate];
            Egm.ExitOperatorMenu();
            paytable.Activate(denomToActivate);
        }
    }
}
```

3. After the game is activate, we'll have the test play one game for that paytable. We'll add a Thread.Sleep() to tell the test to wait a few seconds before starting the game. Then, call the PlayOneGame() method from GameUtilities, like we did in Step 4 of Part 2.

```
[Test]
public void ValidateAllGames()
{
    foreach(var theme in Egm.Themes)
    {
        foreach(var paytable in theme.GameVariations)
        {
            Console.WriteLine(
                $"Theme: {theme.Name}, Paytable: {paytable.Code}, " +
                $"Available Denom: {string.Join(", ", paytable.Denominations)}"
            );

            Egm.EnterOperatorMenu();
            var denomToActivate = paytable.Denominations.First();
            paytable.EnabledDenominations = [denomToActivate];
            Egm.ExitOperatorMenu();
            paytable.Activate(denomToActivate);

            Thread.Sleep(15000);
            GameUtilities.PlayOneGame(Egm);
        }
    }
}
```

4. Run this test. It should go through all themes and paytables, enabling each paytable one by one and playing a game. Congratulations! You've completed all of the steps for Workshop 2.