How to Test Design Automation Apps Locally Using accoreconsole in AutoCAD

Introduction

In this article, we will explore how to locally test design automation apps using `accoreconsole`, a powerful command-line tool that comes with AutoCAD. By using this tool, you can run scripts, automate tasks, and test custom apps without the need for the full AutoCAD GUI. We'll also look at a practical example involving `.crx`, `.dll`, and `.lsp` files to demonstrate how to use `accoreconsole` in real-world scenarios.

What is accoreconsole?

`accoreconsole.exe` is a lightweight headless version of AutoCAD, designed to execute commands, scripts, and custom applications from the command line. It is commonly used as testbed for automating tasks in design automation workflows.

Switches and Command-line Arguments for accoreconsole

Here is a breakdown of the most commonly used switches and what they do:

- **/i** (Input Drawing File (.dwg |.dxf)
  This switch specifies the drawing file that AutoCAD will open. It must be followed by the path to the file.
  Example: `/i "C:\Path\to\your\file.dwg"`
- **/s** (Script File)
  This switch runs a script file (.scr) after the drawing file is opened.
  Example: `/s "C:\Path\to\your\script.scr"`
- **/al** (Autoload)
  This switch allows you to load a `.bundle` in to AcCoreContext. Example: `/al "C:\Path\to\your\app"`, where `app` is folder containing the plugin bundle.
- **/isolate**
  - This switch allows you isolate the coreconsole context without altering the AutoCAD environments.

    We copy the all the files under the current lang's local and roaming folders to the specified input `userDataFolder` folder and copy the registry keys under the user root node to the `userid` key

    Example: `/isolate user1 D:\Temp\ToDeleteLater`

- **/l**

  This switch allows you choose language if a language pack is installed on target machine.

  `/l en-US`

  --------------------These may not be required for Design Automation workflows-----------

- **/p** (Profile)
  This switch loads a specific user profile before running any commands.
  Example: `/p "YourProfile"`

- **/loadmodule**
  This switch loads a module (crx, dbx or .net) before executing any commands.
  Example: `/loadmodule C:\Path\to\your\plugin.dll`

- **/product** (Product Selection)
  This switch allows you to specify the AutoCAD product variant you are working with.
  Example: `/product "AutoCAD"`

- **/readonly**

  This switch load the drawing in read only mode.

  Example: `accoreconsole.exe /i test.dwg /readonly`

Example Project: Running Custom Apps with accoreconsole

The provided C# code defines a plugin for AutoCAD that extracts layer names from the active document and saves them to a text file called `layers.txt`. The plugin is written in a class named `ExtractorCommands` within the `LayerExtractor` namespace.

1. **Namespace and Class**:

   - The `LayerExtractor` namespace encapsulates the plugin functionality.
   - The class `ExtractorCommands` contains methods for AutoCAD command execution.

2. **AutoCAD Command**:

   - The `[CommandMethod("EXTRACTDATA")]` attribute defines a custom AutoCAD command

```
[CommandMethod("EXTRACTDATA")]
public void ExtractData()
{
    var doc = Application.DocumentManager.MdiActiveDocument;
    if (doc is null) return;
    var ed = doc.Editor;
    try
    {
        //extract layer names and save them to layers.txt
        var db = doc.Database;
        using (var writer = File.CreateText("layers.txt"))
        {
            dynamic layers = db.LayerTableId;
            foreach (dynamic layer in layers)
                writer.WriteLine(layer.Name);
        }
    }
    catch (System.Exception e)
    {
        ed.WriteMessage("Error: {0}", e);
    }
}
```

Bundle Structure for Autoloading.

For **AcCoreConsole plugin autoloading** to work correctly, the bundle must follow a specific folder structure and adhere to certain guidelines. The bundle directory, referred to by the path in the `PackageContents.xml`, must be organized as follows:

```
D:\BUNDLE
└───LayerExtractor.bundle
    │   PackageContents.xml
    │
    └───Contents
            LayerExtractor.deps.json
            LayerExtractor.dll
```

- The **LayerExtractor.bundle** folder:
  - This is the top-level folder for the plugin. It must have the `.bundle` suffix to indicate that it is a bundle package.
- The **PackageContents.xml** file:
  - This file contains all metadata about the plugin and tells AutoCAD how to load and handle the plugin. The file must be placed in the root of the `.bundle` folder.

- The **Contents** folder:
  - This folder contains the actual plugin files, including:
    - **LayerExtractor.dll**: The compiled DLL with the plugin code.
    - **LayerExtractor.deps.json**: Other supporting files such as dependencies and debugging information, required from **AutoCAD 2025 onwards**.

Autoloading Process:

When the bundle is placed in the appropriate folder in AutoCAD's support path, AutoCAD automatically detects it and uses the instructions in `PackageContents.xml` to load the plugin when the specified conditions are met (e.g., on command invocation).

Important Requirements:

1. **The bundle directory should contain only one folder**: the `.bundle` folder that includes `PackageContents.xml` and the `Contents` folder. No additional files or folders should exist at the same level as the `.bundle` folder.

2. **PackageContents.xml**: This is a mandatory file that defines how the plugin is loaded. It must match the structure specified for AutoCAD, ensuring correct loading behavior. For more information on the contents and structure of the `PackageContents.xml`, refer to the Autodesk documentation.

By following this structure and configuration, your AutoCAD plugin will be correctly autoloaded when needed, making it available without manual intervention after installation.

PackageContents.xml

The `PackageContents.xml` file defines the metadata for the AutoCAD plugin, specifying how and when the plugin should be loaded.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<ApplicationPackage SchemaVersion="1.0" AppVersion="1.0.0"
 ProductCode="{30410EF1-E42C-4FC4-BC80-7C3803B8976A}"
 Name="Layer Extractor Plugin"
 Description="Exporting layers to a txt file"
 Author="Madhukar Moogala">
    <CompanyDetails Name="Autodesk, Inc"
                    Url="http://tutorials.autodesk.io"
                    Email="aps.help@autodesk.com"/>
    <Components>
        <RuntimeRequirements OS="Win64" SeriesMin="R25.0"
                                        SeriesMax="R25.0"/>
        <ComponentEntry AppName="LayerExtractor"
                        ModuleName="./Contents/LayerExtractor.dll"
                        AppDescription="Exporting layers to a txt file"
                        LoadOnCommandInvocation="True"
                        LoadOnAutoCADStartup="False">
            <Commands GroupName="EXTRACTDATA">
                <Command Global="EXTRACTDATA" Local="EXTRACTDATA"/>
            </Commands>
        </ComponentEntry>
    </Components>
</ApplicationPackage>
```

- **ApplicationPackage Element**:
    - Defines general plugin metadata, including schema version, app version, and unique product code.
    - Describes the plugin as "Layer Extractor Plugin," with the author listed as "Madhukar Moogala."
- **CompanyDetails**:
    - Information about the company (Autodesk), including website and support email.
- **Components**:
    - Specifies **runtime requirements**, such as Windows 64-bit (`OS="Win64"`) and support for AutoCAD R25.0 and later only (`SeriesMin="R25.0"` and `SeriesMax="R25.0"`).
- **ComponentEntry**:
    - Defines the component entry for the plugin.
    - `AppName` : Name of the application (`LayerExtractor`).
    - `ModuleName` : Points to the DLL file that contains the code (`./Contents/LayerExtractor.dll`).

    - `AppDescription`: Describes the purpose of the plugin as "Exporting layers to a txt file."
    - `LoadOnCommandInvocation="True"` ensures the plugin is only loaded when the `EXTRACTDATA` command is invoked.
    - `LoadOnAutoCADStartup="False"` prevents the plugin from loading automatically when AutoCAD starts.

- **Commands**:

    - The `EXTRACTDATA` command is registered globally (`Global="EXTRACTDATA"`) and locally (`Local="EXTRACTDATA"`), matching the command defined in the C# code.

Sync Between Code and XML:

- The command name `EXTRACTDATA` defined in the C# `[CommandMethod]` attribute is referenced in the `PackageContents.xml` to ensure the plugin loads correctly when the command is invoked.
- The `ModuleName` points to the DLL that contains the `ExtractorCommands` class, ensuring that the plugin's functionality is loaded from the correct file.

Written by Madhukar Moogala (APS Developer Advocate)