

Scripting

This page is still under construction, some information may be incomplete.

Introduction

The emDrive Configurator scripting feature lets you create custom C# (.cs) scripts to communicate with CAN Open devices. Scripts can be run in a dedicated Script Window within the application. You can perform actions such as reading and writing device parameters (SDO operations), sending direct commands, managing device states via NMT commands, and even logging or displaying messages. This guide explains the basics—from script file creation to using the scripting API for simple operations.

User Interface Overview

When you open the Script Window (via **Device > Script**), you will see several controls:

- **Browse Button:** Opens a File Open window so you can select your C# script file. Once a file is loaded, its path is displayed next to the button.
- **Run Script Button:** Starts script execution.
- **Stop Button:** Terminates a running script.
- **Infinite Loop Checkbox:** Indicates when your script contains long-running or infinite loops. For scripts that use a `CancellationToken` (introduced in version 2.9.0.0 and later), this option is set automatically.
- **Shortcut Assignment:** You can add scripts to predefined shortcuts by selecting one from the drop-down menu and clicking the Set button.
- **Script Editor:** A simple editor is available from within the window to quickly modify and save your script.
- **Command Overview & Log Sections:** These areas display feedback messages when API calls (such as SDO reads/writes or logging commands) are executed.

Long-running scripts cannot be used with Shortcuts.

ScriptWindow.png
Image not found or image unknown

Creating a Script File

All scripts must be written in C#. You can use any text editor or an IDE like Visual Studio Code to create a `.cs` file. Two common templates are provided:

Basic Template

For quick scripts without long-running loops, use a template similar to this:

```
using System;
using emDrive_Configurator; // or "using eDrive_Configurator;" for versions 2.5.4.0 and lower

public class Script
{
    public void Main()
    {
        // Your code here (e.g., SDO read/write, logging, etc.)
        Procedure.Log("Hello from the basic script!");
    }
}
```

Version 2.5.4.0 and lower use `using eDrive_Configurator;` instead of `using emDrive_Configurator;` .

Long-Running Template (Infinite or Looping Scripts)

For scripts that must run continuously (e.g., monitoring sensor values), change the Main method to accept a Cancellation token. This lets the tool cancel your loop when you click Stop.

This feature is available only from version 2.9.0.0 and onwards. Long-running scripts should not be used in versions below 2.9.0.0.

```
using System;
using System.Threading;
using emDrive_Configurator; // or "using eDrive_Configurator;" for versions 2.5.4.0 and lower
```

```

public class Script
{
    public void Main(Cancellation_token cancellationToken)
    {
        while (!cancellationToken.IsCancellationRequested)
        {
            // Periodically execute code (e.g., monitor sensor value)
            Procedure.Log("Script running...");
            Procedure.Delay(1000); // Pause for one second
        }
        Procedure.Log("Script canceled.");
    }
}

```

Ensure you include `using System.Threading;` if you are using long-running loops with a cancellation token. (Version 2.9.0.0 and later support this feature.)

Scripting API Methods

The emDrive Configurator exposes several API methods that you can use in your scripts. Below is a detailed summary of the methods available for interacting with your CAN Open device.

Every method needs to be called using the `Procedure` class as seen in the examples above.

SDO Data Transfers

SDO_Read Methods

These methods read data (Service Data Objects) from the device. They come in several overloads:

```
bool SDO_Read(int nodeId, uint index, ushort subindex, out object value, out string errorMessage);
```

Usage: Reads a parameter from a specified node and object index/subindex. The method converts the device's hex response into a readable value and returns any error messages if the read fails.

```
dynamic SDO_Read(int NodeId, uint Index, ushort Subindex);
```

Usage: Returns the read value as a dynamic type after automatically processing the hex data.

```
bool SDO_Read(int NodeId, uint Index, ushort Subindex, Action<CO_Object, dynamic> Parser);
```

Usage: Reads data and passes the value to a provided callback (Parser) for further processing.

SDO_Write Methods

These methods write a value to a CAN Open object.

```
bool SDO_Write(int nodeId, uint index, ushort subindex, dynamic value, out string errMessage);
```

Usage: Writes the specified value to a node's object. It logs the operation and returns a message indicating success or error.

Direct Command Execution

```
string Direct_Command(string text);
```

Usage: Sends a direct command (as a simple string) to the device and returns the response.

This is recommended only for advanced users who are familiar with the CAN Open protocol on the EMSISO CAN Card

Timing Control

```
void Delay(int Duration);
```

Usage: A simple wrapper around `Thread.Sleep` that pauses script execution. The duration is in milliseconds.

Network Management (NMT) Commands

These methods allow you to change the state of CAN Open nodes:

```
bool NMT_Operational();  
bool NMT_Operational(int nodeId);
```

Usage: Switch the device (or a specific node) to Operational Mode.

```
bool NMT_Preoperational();  
bool NMT_Preoperational(int nodeId);
```

Usage: Set the device (or node) into Preoperational Mode.

```
bool NMT_Stopped();  
bool NMT_Stopped(int nodeId);
```

Usage: Stops the device (or node).

```
bool NMT_Reset();  
bool NMT_Reset(int nodeId);  
bool NMT_Reset_Communication();  
bool NMT_Reset_Communication(int nodeId);
```

Usage: Resets the device or its communication link.

User Interaction and Logging

```
void ShowMessageBox(string text);
```

Usage: Display a standard message box to the user.

```
bool ShowDialog(string text);
```

Usage: Open a dialog window that returns a Boolean value (for example, after the user confirms an action).

```
dynamic InputDialog(string caption);
```

Usage: Shows a prompt for user input and returns the entered text.

Script Examples

Below are two example scripts that illustrate how to use the API methods to interact with CAN Open devices.

Example 1: Basic SDO Read/Write

This example shows how to read a value from a CAN Open object and then write a new temporary value.

```
using System;
using emDrive_Configurator;

public class Script
{
    public void Main()
    {
        // Define node and object identifiers (using temporary indexes)
        int nodeId = 1;
        uint index = 0x2000;
        ushort subindex = 0x01;

        object readValue;
        string errMessage;

        // Attempt to read the current value from the device
        if (Procedure.SDO_Read(nodeId, index, subindex, out readValue, out errMessage))
        {
            Procedure.Log("SDO Read successful. Value: " + readValue);
        }
        else
        {
            Procedure.Log("SDO Read error: " + errMessage);
        }

        // Set a temporary value and write it to the device
        dynamic tempValue = 123; // Change 123 to any temporary test value as needed
        if (Procedure.SDO_Write(nodeId, index, subindex, tempValue, out errMessage))
        {
            Procedure.Log("SDO Write successful. New Value: " + tempValue);
        }
        else
        {
            Procedure.Log("SDO Write error: " + errMessage);
        }
    }
}
```

Example 2: Test with Long-Running Loop

This script continuously reads from the device and exits the loop if a defined break condition is met. It demonstrates the use of a CancellationToken so that the script can be stopped via the UI.

```
using System;
using System.Threading;
using emDrive_Configurator;

public class Script
{
    // Use this template for scripts with long-running loops.
    public void Main(Cancellation_token cancellation_token)
    {
        int nodeId = 1;
        uint index = 0x2000;
        ushort subindex = 0x01;
        int condition = 20;

        while (!cancellation_token.IsCancellationRequested)
        {
            dynamic? value;
            string errorMessage;

            if (Procedure.SDO_Read(nodeId, index, subindex, out value, out errorMessage))
            {
                Procedure.Log("Current SDO Value: " + value);
            }
            else
            {
                Procedure.Log("Read error: " + errorMessage);
            }

            // Check if a break condition is met (example: value equals a specific test value)
            if (value is int && value == condition)
            {
                Procedure.Log("Break condition met. Exiting loop.");
                break;
            }
        }
    }
}
```

```
// Wait for 1 second before next iteration
Procedure.Delay(1000);
}

Procedure.Log("Script execution completed.");
}
}
```

Advanced Scripting

Script-Watch Interoperability

This functionality is only available for emDrive Configurator 2.13.0.0 and later.

These API calls allow the user to add *virtual script objects* to **Watch** and write to them, as well as request the last value read for a specific object already in Watch. This can be achieved by using the `WatchScriptObject` class or the method calls described below.

WatchScriptObject class

The `WatchScriptObject` class uses the API calls described below but is a more user-friendly approach to this functionality. Unless specified the `index` and `subindex` properties will be set automatically. `index` will always be `0xFFFF` and `subindex` will increment from `0` onwards. If the user wishes to reset the subindex counter they must call the `ResetSubIndexCounter` method.

```
public class WatchScriptObject
{
    // Properties
    public string Name { get; }
    public int NodeId { get; } // Default: 0
    public uint Index { get; } // Default: 0xFFFF
    public ushort SubIndex { get; }
    public ushort DataType { get; }

    // Constructors
    public WatchScriptObject(string name, ushort datatype);
    public WatchScriptObject(string name, ushort subindex, ushort datatype);
```



```

public WatchScriptObject(string name, uint index, ushort subindex, ushort datatype);

// Methods
public bool TryAddToWatch();
public bool TryAddValueToWatch(dynamic value);
}

```

The `TryAddToWatch` method will attempt to add the virtual object to **Watch**. See **TryAddScriptEntryToWatch** method for possible errors.

The `TryAddValueToWatch` method will attempt to add the value to the already created virtual object in Watch. See **TryAddScriptEntryValue** method for possible errors.

TryAddScriptEntryToWatch method

Attempts to create a new script virtual object in **Watch** with the specified `name`, `nodeId`, `index`, `subindex`, and `datatype`. Returns `true` if successful or `false` if failed.

```

public static bool TryAddScriptEntryToWatch(string name, int nodeId, uint index, ushort subindex, ushort
datatype)

```

Fail reasons:

- Object with specified `nodeId`, `index` and `subindex` already exists in **Watch**.

TryAddScriptEntryValue method

Attempts to write `value` to the script virtual object in **Watch**. Returns `true` if successful or `false` if failed.

```

public static bool TryAddScriptEntryValue(int nodeId, uint index, ushort subindex, dynamic value)

```

Fail reasons:

- Object with specified `nodeId`, `index` and `subindex` does not exist in **Watch**.
- Failed to convert value to the declared datatype.

TryGetLastWatchValue method

Attempts to get the last read value of the object specified with `nodeId`, `index`, `subindex` and saves it to `value`. Returns `true` if successful or `false` if failed.

```

public static bool TryGetWatchValue(int nodeId, uint index, ushort subindex, out double value)

```

Fail reasons:

- Object with specified `nodeId`, `index` and `subindex` does not exist in **Watch**.
- Failed to convert object value to type `double`.

Example file

Example file for Script - Watch Interoperability (also available as download)

```
/*
  Author:    Amadej Arnus
  Date:      2024-04-23
*/

using System;
using System.Windows.Forms;
using emDrive_Configurator;
using System.Threading;
using Timer = System.Threading.Timer;

using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Globalization;
using System.IO;
using System.IO.Ports;
using System.Linq;
using System.Management;
using System.Text;
using System.Text.RegularExpressions;

public class Script
{
    // Prepare error string
    string err = string.Empty;

    // Create object that we will attempt to read from watch
    CO_Object HwMonitorUdc = new CO_Object(0x3071, 0x00);
```

```
public void Main(CancellationToken cancellationToken)
{
    // This will run the example for the WatchScriptObject
    ObjectBasedVersion(cancellationToken);

    // This will run the example for the API calls
    //ObjectBasedVersion(cancellationToken);
}

private void ObjectBasedVersion(CancellationToken cancellationToken)
{
    // Create new objects for watch-script interoperability
    Procedure.WatchScriptObject intObject = new Procedure.WatchScriptObject("ScriptObject_Int32",
0x0004);
    Procedure.WatchScriptObject floatObject = new Procedure.WatchScriptObject("ScriptObject_Real32",
0x0008);

    // Add int object to watch
    if (intObject.TryToAddToWatch())
    {
        LogResult("ScriptObject_Int32 added to watch");
    }
    else
    {
        LogResult("ScriptObject_Int32 not added to watch");
    }

    // Add float object to watch
    if (floatObject.TryToAddToWatch())
    {
        LogResult("ScriptObject_Real32 added to watch");
    }
    else
    {
        LogResult("ScriptObject_Real32 not added to watch");
    }

    int i = 0;
```

```

// Do the loop until cancellationToken is requested
while (!cancellationToken.IsCancellationRequested)
{
    double hwMonitorValue = 0;

    // Read last value of HwMonitorUdc object
    if (HwMonitorUdc.TryGetWatchValue(1, out hwMonitorValue))
    {
        i++;

        // Set new value to int and float objects
        if (!intObject.TryToAddValueToWatch(i))
        {
            LogResult("Failed to write value to ScriptObject_Int32");
        }

        if (!floatObject.TryToAddValueToWatch(hwMonitorValue * 1.5))
        {
            LogResult("Failed to write value to ScriptObject_Real32");
        }
    }

    // If not successful, log the error
    else
    {
        LogResult("Failed to get HwMonitorUdc value");
    }

    Procedure.Delay(200);
}

private void ApiBasedVersion(Cancellation_token cancellationToken)
{
    // Add int object to watch
    if (TryAddScriptEntryToWatch("ScriptObject_Int32", 1, 0xFFFF, 0x01, datatype: 0x0004))
    {
        LogResult("ScriptObject_Int32 added to watch");
    }
}

```

```

}
else
{
    LogResult("ScriptObject_Int32 not added to watch");
}

// Add float object to watch
if (TryAddScriptEntryToWatch("ScriptObject_Real32", 1, 0xFFFF, 0x02, datatype: 0x0008))
{
    LogResult("ScriptObject_Real32 added to watch");
}
else
{
    LogResult("ScriptObject_Real32 not added to watch");
}

int i = 0;

// Do the loop until cancellationToken is requested
while (!cancellationToken.IsCancellationRequested)
{
    double hwMinutorValue = 0;

    // Read last value of HwMonitorUdc object
    if (TryGetWatchValue(1, 0x3071, 0x00, out hwMinutorValue))
    {
        i++;

        // Set new value to int and float objects
        if (!TryAddScriptEntryValue(1, 0xFFFF, 0x01, i))
        {
            LogResult("Failed to write value to ScriptObject_Int32");
        }

        if (!TryAddScriptEntryValue(1, 0xFFFF, 0x02, hwMinutorValue * 1.5))
        {
            LogResult("Failed to write value to ScriptObject_Real32");
        }
    }
}

```

```

    }
    // If not successful, log the error
    else
    {
        LogResult("Failed to get HwMonitorUdc value");
    }

    Procedure.Delay(200);
}

// The following methods are just wrappers to shorten the method calls
bool TryAddScriptEntryToWatch(string name, int nodeId, uint index, ushort subindex, ushort datatype)
{
    return Procedure.TryAddScriptEntryToWatch(name, nodeId, index, subindex, datatype);
}
bool TryAddScriptEntryValue(int nodeId, uint index, ushort subindex, dynamic value)
{
    return Procedure.TryAddScriptEntryValue(nodeId, index, subindex, value);
}
bool TryGetWatchValue(int nodeId, uint index, ushort subindex, out double value)
{
    return Procedure.TryGetWatchValue(nodeId, index, subindex, out value);
}

void LogResult(string LogString)
{
    LogString = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss") + ": " + LogString;
    Procedure.Log(LogString);
}

// CO_Object for readability
class CO_Object
{
    public uint index;
    public ushort subindex;
    public dynamic value;
}

```

```

public CO_Object(uint index, ushort subindex)
{
    this.index = index;
    this.subindex = subindex;
}

public bool TryGetWatchValue(int nodeId, out double value)
{
    return Procedure.TryGetWatchValue(nodeId, index, subindex, out value);
}
}

```

Script Features

Script features are extensions to the scripting functionality.

In order to use script features, the user must add

```
using emDrive_Configurator.Scripts.Features;
```

at the start of the script file.

Modbus TCP

This functionality is only available for emDrive Configurator 2.13.1.0 and later.

[Modbus_TCP.png](#) Image not found. Type unknown

ModbusTCP currently supports connection with Modbus Servers (Client Mode) through TCP. In order to connect to the device, you need to provide the IP Address, Port and Modbus Server ID (Bus Address).

In the following Example, the IP is 127.0.0.1, the Port is 502 and the Modbus Server ID is 1:

```
ModbusTCP ModbusDevice1 = new ModbusTCP("127.0.0.1", 502, 1);
```

Alternatively, the device endianness can also be specified in the connection constructor:

```
ModbusTCP ModbusLocalhost = new ModbusTCP("127.0.0.1", 502, 1, ModbusEndianness.LittleEndian);
```

Currently Read and Write Holding registers operation is supported using generic functions:

```
short i16_var = 0;

// Read Holding Register at Address 100
if(ModbusLocalhost.ReadHoldingRegister<short>(100, out i16_var) == true)
{
    // Read Successful
}
else
{
    // Read Failed
}

// Write Holding Register at Address 100
ModbusLocalhost.WriteHoldingRegister(100, i16_var);
```

Supported data types:

- *short*
- *ushort*
- *int*
- *uint*
- *float*

Useful Learning Resources

If you're new to C# or need additional reading to get familiar with the language and its concepts, check out these established sources:

- **Microsoft C# Documentation:**

Comprehensive reference and tutorials on the C# language.

<https://docs.microsoft.com/en-us/dotnet/csharp/>

- **C# Language Reference:**

Detailed descriptions of language syntax, keywords, and code examples.

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/>

- **.NET Tutorials:**

Learn the fundamentals of C#, .NET Framework, and best practices.

<https://docs.microsoft.com/en-us/dotnet/core/tutorials/>

Additional Tips and Conclusion

Version Considerations

For scripts with long-running loops, ensure you are using version **2.9.0.0** or later, which supports using a **CancellationToken**.

Earlier versions (e.g., **2.5.4.0** and lower) require a different namespace (`using eDrive_Configurator;`).

Script Debugging

Utilize the Log output function (`Procedure.Log`) to print messages and debug your script during execution.

Integration with Watch

Use the `WatchScriptObject` class to monitor variable values during script execution. This can help you verify that the SDO read/write operations are working as expected.

By following this guide, even users with minimal technical expertise will be able to write, run, and troubleshoot simple C# scripts to interact with CAN Open devices via the emDrive Configurator. Experiment with the examples provided and consult the useful links if you need further insight into C# programming.

Happy scripting!

Revision #18

Created 7 May 2024 13:01:07 by Amadej Arnuš

Updated 10 April 2025 13:34:47 by Amadej Arnuš