ALIBRE SCRIPT ADVANCED API

MANUAL VERSION 1.0

DISCLAIMER

Information in this document is subject to change without notice and does not represent a commitment on the part of the manufacturer. The software described in this document is furnished under license agreement or nondisclosure agreement and may be used or copied in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the software purchaser's use, without prior written permission.

CONTENTS

CHAPTER 1: INTRODUCTION

Alibre Script provides a simplified set of functions and properties (together called an API) allowing quick and easy use of the underlying Alibre Design API using the Python language. In order to make it as easy to use as possible some sacrifices had to be made resulting in less flexibility than the Alibre Design API provides.

The Alibre Script Advanced API addresses this by providing advanced users with a way to access the full Alibre Design API from inside Alibre Script.

Familiarity with the Alibre Design API and how it is used is required. See the included AlibreAPIHelp.chm file for full details. Also familiarity with the C# programming language is also required. Neither of these topics are covered in this manual. Familiarity with Windows Forms (WinForms) is required for constructing user interfaces.

A WARNING

With great power comes great responsibility. Alibre Script contains error and sanity checking to try to stop users from doing things that could create problems. The Advanced API described in this manual has none of that and without careful use it could be possible to destabilize or crash Alibre Design, potentially causing data loss. Always thoroughly test code before sharing it with others.

CHAPTER 2: GETTING STARTED

This chapter will show some simple examples of how to use the Advanced API, including passing variables to and from the environment.

The premise is simple, by using a special class available in a script C# code can be compiled and executed. This C# code has full access to the Alibre Design API.

Here is a simple example.

```
Cs = CSharp()
Cs.CompileAndRun('int Foo = 1;')
```

An instance of the CSharp class is created then a piece of C# code is compiled and run. The C# code creates an integer called Foo and sets it to the value one.

This is not particularly useful because it doesn't do anything. Once the CompileAndRun function is finished the variable Foo is lost, along with the entire C# environment.

In order to get variables out of C# and into the script we simply need to look at what CompileAndRun returns. Adding:

```
print Cs.CompileAndRun('int Foo = 1;')
```

gives the following output in the console:

```
{'Foo': 1}
```

This is a Python dictionary. A dictionary is a list of values of different types, each one having a name. Now we can access Foo in the script:

```
OutVariables = Cs.CompileAndRun('int Foo = 1;')

print OutVariables['Foo']
```

The important aspect to remember is that variables created in C# are automatically added to the dictionary that is returned to the script.

Passing variables in works in exactly the same way, by using a dictionary:

```
InVariables = {'Bar': 41}

OutVariables = Cs.CompileAndRun('Variables["Bar"] = (int)Variables["Bar"] +
1;', InVariables)

print OutVariables
```

We created a dictionary with a value called Bar in the script. In the C# code we take that input value and increment it. When we look at what is returned we see:

```
{'Bar': 42}
```

There are a few important things to note.

- Variables passed in are accessed by using the 'Variables' C# dictionary
- Variables in Python do not have an explicit type (e.g. int) however they do in C# therefore variables passed in must always be cast to the correct C# type.
- Variables passed in are also passed out, even if they are not used in the C# code

C# code is compiled before it is run, and compilation takes time, however once complete execution is efficient. In the examples so far the C# code is compiled and run in a single step, however in some

situations this may not be ideal, for example if the same piece of C# code is run more than once. Fortunately compilation can be separated from running as this example shows:

```
InVariables = {'Bar': 41 }
MyCSharpCode = Cs.Compile('Variables["Bar"] = (int)Variables["Bar"] + 1;')
print Cs.Run(MyCSharpCode, InVariables)
print Cs.Run(MyCSharpCode, InVariables)
```

Note that a different dictionary of variables can be passed every time the compiled C# code is run.

CHAPTER 3: ACCESSING THE ALIBRE DESIGN API

OVERVIEW

Here is an example that returns a list of all the current parts and assemblies that are open along with the number of design axes in each one.

```
Cs = CSharp()
OutVariables = Cs.CompileAndRun('''
using AlibreX;
foreach (IADDesignSession Sess in AlibreRoot.Sessions) {
  Variables[Sess.Name] = Sess.DesignAxes.Count;
}
''')
print OutVariables
```

Note the use of triple quotes. This allows multi-line strings to be created in a script, making formatting of C# code more familiar.

First we have the using statement for the AlibreX assembly. This is required for any code that uses the Alibre Design API (AD API).

Next we get a list of all design sessions currently opened by using the AlibreRoot object. This is a global variable that is always available in C# code to access the IADRoot instance.

We then add to the Variables dictionary the number of design axes, using the part/assembly name as the dictionary key. This is an example of what is returned from running this script:

```
{'DN125 Flange PN16': 3, 'CSharpTest': 3}
```

THE CURRENT PART OR ASSEMBLY

When a script executes it is always inside a part or assembly. At the AD API level this is an IADSession object. Here is an example of how to get the name of the part or assembly that the script is executing in:

```
Cs = CSharp()
Variables = Cs.CompileAndRun('''
using AlibreX;
string CurrentSessionName = CurrentSession.Name;
''')
print Variables
```

and here is an example of the output:

```
{'CurrentSessionName': 'CSharpTest'}
```

The CurrentSession variable is always available and is the IADSession instance that the script is executing inside of.

CHAPTER 5: C# CLASSES

It is possible to create and use classes in C#. Here is an example that creates a class which wraps IADSession:

```
Cs = CSharp()
Variables = Cs.CompileAndRun('''
using AlibreX;

class MySession {
  public IADSession Session;
  public MySession(IADSession Session) {
    this.Session = Session;
  }
  public override string ToString() {
    return Session.Name;
  }
}

MySession Se = new MySession(CurrentSession);

Variables["Name"] = Se.ToString();
''')
print Variables['Name']
```

and here is an example output on the console:

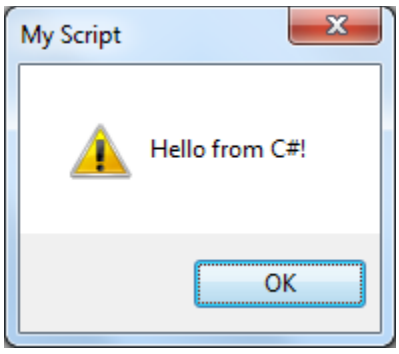
```
CSharpTest
```

CHAPTER 6: USER INTERFACES

Full access to WinForms is provided allowing more complex user interfaces to be constructed than is available in Alibre Script. Here is a simple example:

```
Cs = CSharp()

Cs.CompileAndRun('''

using System.Windows.Forms;

MessageBox.Show("Hello from C#!", "My Script", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);

''')
```

And the output:



The global variable ParentForm is available and is the Windows Form that you should use as the parent when creating new forms of your own. It can also be used to determine the correct user interface thread to use (e.g. ParentForm.InvokeRequired).

CHAPTER 7: UNITS

The Alibre Design API uses centimeters for distances and radians for angles, however scripts can use inches, centimeters or millimeters, along with degrees. Therefore when passing distances and angles to C# code a conversion from script units to Alibre Design API units must be made. Likewise when returning distances and angles from C# code a conversion from Alibre Design API units to script units must be made.

ANGLES

Scripts always use degrees and the Alibre Design API always uses radians, therefore the conversion is always the same. Here is an example:

```
Cs = CSharp()
print Cs.CompileAndRun('''
using System;
double Radians = Math.PI * (double)Variables["Angle"] / 180.0;
''', {'Angle': 180.0 })
```

Output:

```
{'Angle': 180.0, 'Radians': 3.1415926535897931}
```

DISTANCES

Distances are a bit more complicated because when writing C# code it is not known in advance what units the script will use. To solve this two helper functions are provided:

- Units.ToADUnits(value) – converts from script units to centimeters
- Units.FromADUnits(value) – converts from centimeters to script units

Here is a usage example:

```
Cs = CSharp()
print Cs.CompileAndRun('''
using AlibreScript.API;
double AD_Dist = Units.ToADUnits((double)Variables["Distance"]);
''', {'Distance': 12.3 })
```

Output:

```
{'Distance': 12.300000000000001, 'AD_Dist': 1.2300000000000002}
```

CHAPTER 8: OUTPUT

C# code can output to the console using a built in Utilities class. Examples:

```
Cs = CSharp()
Cs.CompileAndRun('''
Utilities.OutputLine("Hello {0}", 45);
Utilities.OutputLine("Foo");
Utilities.Output("A");
Utilities.Output(" {0:0.00}", 66.4);
''')
```

OutputLine adds a newline to the end of the text, whereas Output does not. Standard string.Format style formatting can be used directly.

Console output:

```
Hello 45
Foo
A 66.40
```

APPENDIX A: PREDEFINED GLOBAL VARIABLES

The following global variables are always defined an available in C# code:

Name	Description
Variables	A C# dictionary representing the python dictionary passed to CompileAndRun or Run
AlibreRoot	The Alibre IADRoot object
CurrentSession	The IADSession for the part or assembly that the script is executing in
CurrentScript	The path and file name of the currently executing script, if it has been saved to disk
ParentForm	A Form instance to use as a parent and UI thread when creating user interfaces
Utilities	Static class defining utility functions

APPENDIX B: HINTS AND TIPS

WRAPPING C# CODE

Small snippets of C# code can be written to perform specific tasks. This can then be placed inside a Python function, essentially wrapping it and hiding the fact that C# is used from the caller. Here is an example:

```
Cs = CSharp()


# increments a value using C# instead of python
def Increment(Value):
  VarIn = {'Value': Value }
  VarOut = Cs.CompileAndRun('''
    Variables["Value"] = (int)Variables["Value"] + 1;
    ''', VarIn)
  return VarOut['Value']


print Increment(16)
print Increment(41)
```

Here is the output to the console:

```
17
42
```

NETWORKING SUPPORT

Here is a simple example:

```
Cs = CSharp()
print Cs.CompileAndRun('''
using System.Net;
string HostName = Dns.GetHostName();
''')
```

The example output:

```
{'HostName': 'Cochise'}
```