Scripting

READY LABEL VER 1.+

# Introduction

Ready Label has a built-in scripting engine that balances being both flexible and powerful. This document describes the basics of how this component works and what developers need to know to leverage it for their own processes. This document will also include some basic examples to help developers get started. The appendix includes the two sample scripts included in the with the program installation as well as the object descriptions.

# The Basics

## **Languages**

Currently the scripting engine only supports C#.

## **Structure**

A script file should be in the standard .cs file format. The namespace and class name must be set to “BioSero.ReadyLabel.Scripting” and “Script” respectively or the engine will not be able to locate the correct object. Inside the classes the scripting engine will look for methods with predesignated names and params. See Appendix A for full script examples.

## **Execution**

The scripting engine can be thought of as having two distinct execution domains: generate and print.

### **Generate**

This execution domain begins when a batch of labels are being created based on the current template. This domain has three execution points: the start of a batch generate process, the start of an individual label generate process, and the end of the batch generate process. These three execution points are mapped to the following script methods:

#### **Start of Batch**

**Method:**

public void OnStart(int numberOfLabels)

**Params:**

numberOfLabels – the number of labels to be generated.

#### **Start of Label Generate**

**Method:**

public void Run(Dictionary<string,string> data, BioSero.ReadyLabel.LabelTemplate template, BioSero.ReadyLabel.Label label)

**Params:**

data – A key-value pair of data to be added to the label. Ready label will look for any fields with the associated header key (specificied as “{“ + key + “}” in the label template field) and replace those fields with the assigned value at generate time.

template – the current label template

label – object reference to the current label generate result

#### **End of Batch**

**Method:**

public void OnEnd()

**Params:**

N/A

### **Print**

This execution domain begins when a batch of generated labels are being printed. This domain has four execution points: the start of a batch print process, before the individual label prints, after the individual label print has completed, and the end of the batch print process. These four execution points are mapped to the following script methods:

#### **Start of Batch**

**Method:**

public void OnStart(int numberOfLabelsToPrint)

**Params:**

numberOfLabelsToPrint – Total number of labels to be printed

#### **Before Label Print**

**Method:**

public byte[] ManipulateRawData(BioSero.ReadyLabel.LabelPrinter printer, byte[] rawData, BioSero.ReadyLabel.Label label)

**Params:**

printer – the current label printer class

rawData – the data to be sent to the printer

label – the label object being printed

#### **After Label Print**

**Method:**

public void OnEachLabelPrintComplete(BioSero.ReadyLabel.Label label, Dictionary<string, string> data)

**Params:**

label – the label object that was printed

data – the data printed, organized in key-value pairs

#### **End of Batch**

**Method:**

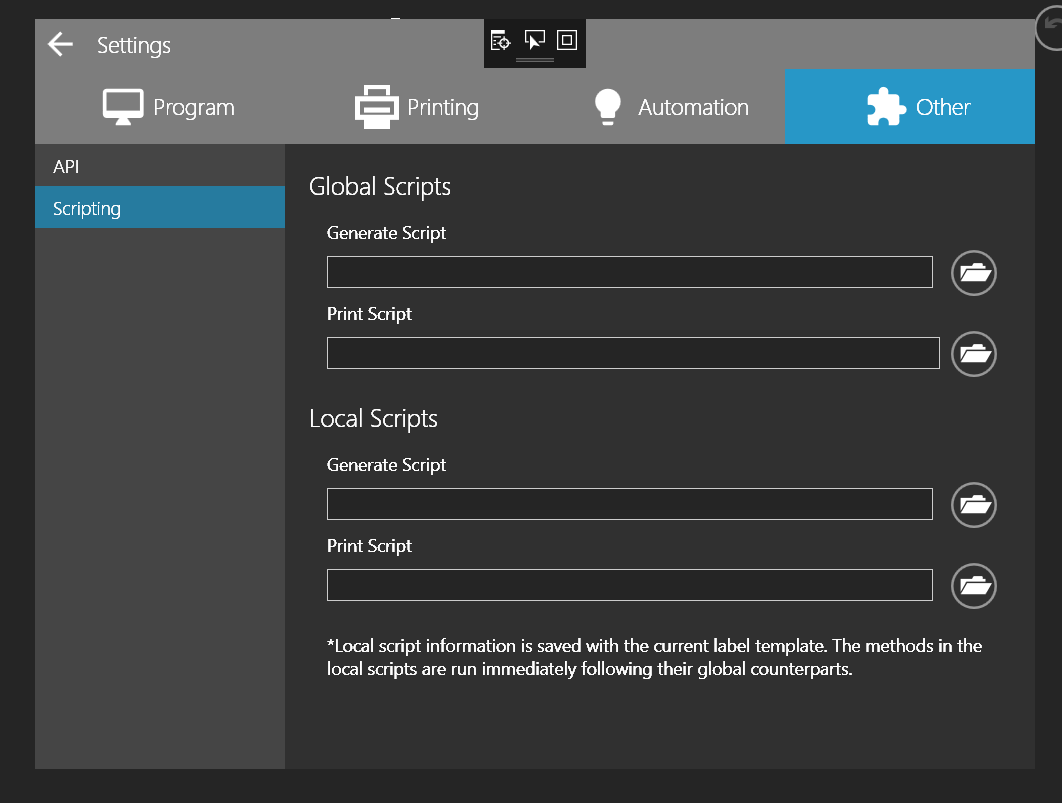
public void OnEnd()

**Params:**

N/A

## **Setup**

To add your script the program, go to Settings > Other > Scripts. Once there you can browse for the script file you want to set for each execution domain.



### **Global vs Local**

In Ready Label you can set your script to the Global or Local sections. The Global script section will execute every time a label is printed regardless of the label template being used. The Local script, on the other hand, is saved with the currently opened label template and will only be run when that label template is generated or printed. The execution order for these is always Global method > Local method. In other words, the global script method will always be called and executed before the local script method at each execution point.

\*Note: the local section only appears in the settings window when a label template has been opened.

# Appendix

## **A – Sample Scripts**

### **SampleGenerateScript.cs**

//REFERENCE C:\Program Files (x86)\MySQL\Connector NET 6.8.3\Assemblies\v4.0\MySql.Data.dll

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Forms;

//using MySql.Data.MySqlClient;

namespace BioSero.ReadyLabel.Scripting

{

public class Script

{

//MySqlConnection Connection;

public void OnStart(int numberOfLabels)

{

//This procedure is called a the start of a batch generate. One use for this would be to open a data connection. See Below.

//string server = "localhost";

//string database = "sequence";

//string uid = "testuser";

//string password = "pa55word";

//string connectionString = "SERVER=" + server + ";" + "DATABASE=" + database + ";" + "UID=" + uid + ";" + "PASSWORD=" + password + ";";

//Connection = new MySqlConnection(connectionString);

//Connection.Open();

}

public void Run(Dictionary<string,string> data, BioSero.ReadyLabel.LabelTemplate template, BioSero.ReadyLabel.Label label)

{

//This procedure is called everytime a label is printed. Use it to add data to a label.

if (!data.ContainsKey("Data"))

{

data.Add("Data", "Data Test"); //Data Test will be aplied to the field {Data}

}

else

{

data["Data"] = "Data Test";

}

//This can also pull in data from the source

//MySqlCommand command = new MySqlCommand("Select nextval('Sequence1')", Connection);

//string result = command.ExecuteScalar().ToString();

//if (!data.ContainsKey("Result"))

//{

// data.Add("Result", result); //the result of the sql evaluation will be aplied to the field {Result}

//}

//else

//{

// data["Result"] = result;

//}

}

public void OnEnd()

{

//This procedure is called a the end of a batch generate. Here you can terminate any processes or connections.

//Connection.Close();

}

}

}

### **SamplePrintScript.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Threading;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

//internal references

using BioSero.ReadyLabel.Scripting;

using BioSero.ReadyLabel;

using BioSero.Utilities;

using MahApps.Metro.Controls;

using BioSero.Utilities.Metro.Controls;

namespace BioSero.ReadyLabel.Scripting

{

public class Script

{

string PrintLogFileName;

//EmptyMetroWindow window;

public void OnStart(int numberOfLabelsToPrint)

{

// This procedure is called before any printing begins. It can be used to open the connection to a database or create a log file for logging all printed data.

//Set log path

PrintLogFileName = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + "\\Ready Label\\Print Log " + DateTime.NowdToString("yyyy\_MM\_dd\_HH\_mm\_ss") + ".txt";

//User Input -> display a message or get user input here

//Thread thread = new Thread(() =>

// {

// window = new EmptyMetroWindow();

// window.Width = 400;

// window.Height = 200;

// window.WindowStartupLocation = WindowStartupLocation.CenterScreen;

// window.Title = "Example Window";

// TextBlock block = new TextBlock();

// block.Text = "This is an example.";

// block.Margin = new Thickness(5);

// block.FontSize = 16;

// Button okButton = new Button();

// okButton.Content = "Okay";

// okButton.Margin = new Thickness(5,20,5,5);

// okButton.Width = 80;

// okButton.Click += okButton\_Click;

// StackPanel stackpanel = new StackPanel();

// stackpanel.Orientation = Orientation.Vertical;

// stackpanel.HorizontalAlignment = System.Windows.HorizontalAlignment.Center;

// stackpanel.VerticalAlignment = System.Windows.VerticalAlignment.Center;

// stackpanel.Children.Add(block);

// stackpanel.Children.Add(okButton);

// window.LayoutRoot.Children.Add(stackpanel);

// window.ShowDialog();

// });

//thread.SetApartmentState(ApartmentState.STA); //Set the thread to STA

//thread.Start();

//thread.Join();

}

//void okButton\_Click(object sender, RoutedEventArgs e)

//{

// window.Close();

//}

public byte[] ManipulateRawData(BioSero.ReadyLabel.LabelPrinter printer, byte[] rawData, BioSero.ReadyLabel.Label label)

{

// This procedure can be used to either manipulate the raw printer data before it is sent to the printer (for instance, special printer control characters could be added),

// or used to display data for debugging purposes.

//File.WriteAllBytes(@"c:\output.txt");

//System.Diagnostics.Process.Start("notepad.exe", @"c:\output.txt")

return rawData;

//Alternatively, data could be converted to a string then displayed in a message box.

//string text = System.Text.Encoding.Default.GetString(rawData);

//MessageBox.Show(text);

//return System.Text.Encoding.Default.GetBytes(text);

}

public void OnEachLabelPrintComplete(BioSero.ReadyLabel.Label label, Dictionary<string, string> data)

{

// this procedure is called after each label is printed

if (!File.Exists(PrintLogFileName)) // write header if first time

{

List<string> headers = new List<string>(data.Keys);

headers.Insert(0, "Element Count");

headers.Insert(1, "Print Error");

headers.Insert(2, "Error Text");

File.WriteAllText(PrintLogFileName, string.Join(",", headers) + Environment.NewLine);

}

string values = label.PrintElements.Count() + "," + label.LabelHasError.ToString() + "," + label.LabelErrorText + ",";

values += string.Join(",", data.Values);

File.AppendAllText(PrintLogFileName, values + Environment.NewLine);

}

public void OnEnd()

{

// this procedure is called after all labels have been printed.

}

}

}

## **B – Object Models**

### **Label**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Errors | IEnumerable<LabelException> | List of errors associated with the label instance (during print or generate steps) |
| GenerationInfo | GeneratedLabelInfo | Information associated with the label creation. See type description below. |
| LabelErrorText | string | Returns concatenated string of label Errors list |
| LabelHasError | bool | Returns true if Errors.Count > 0 |
| PhysicalProperties | LabelPhysicalProperties | The labels physical properties (height, width, etc.). See type description below |
| PrintElements | IEnumerable<PrintElement> | The print elements contained by the label instance |
| Rotation | double | Label's rotation (usually restricted to 90 degree angles) |

### **LabelPhysicalProperties**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| IsFlipped | bool | Determines whether the label should print flipped 180 degree |
| BackingWidth | double | The backing paper width for the label in mm |
| CornerRadius | double | The cornerradius of the label in degrees |
| Height | double | The height of the label in mm |
| LabelSpacing | double | The space between labels on a roll in mm |
| Width | double | The width of the label in mm |
| XOffset | double | The xoffset in mm used to determine the x 0 position of printable surface |
| YOffset | double | The yoffset in mm used to determine the y 0 position of printable surface |

### **GeneratedLabelInfo**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| GenerateTime | DateTime | The date and time when the label was created |
| PrintTime | DateTime | The date and time when the label was printed |
| TemplatePath | string | The file path to the label template used to generate the label |