Why Scripting On Your Okuma Control

Automating tasks and accessing API functions on your Okuma machine tool *is possible without* downloading Microsoft Visual Studio or spending weeks learning Windows programming!

IronPython scripting is a lightweight alternative which will allow you to use the Okuma Open API to access machine data and wait for events, write Excel files, send emails and texts and a lot more. Using IronPython and a few simple tools, you can utilize the power of your P-series control to eliminate repetitive tasks, reduce entry/calculation errors and capture data exactly the way you want it.

> Links online frequently change.
> At time of writing, all links in this tutorial were verified to work. If you run across a link that doesn't work, please let us know so we can fix it!

This tutorial will cover the following topics. Each concept builds on the previous so it is strongly suggested you work through each item before moving to the next.

Downloading IronPython and the Okuma Open API

> Nothing you're going to download needs to be installed but you *will* want to keep it all together. I keep a full scripting setup on a USB stick in a folder called IronPythonScripting so I can easily move it to machine tools. 2GB should be more than sufficient.

Getting IronPython

IronPython is a scripting and programming language which can use .NET libraries including the Okuma Open API and the .NET Framework. It's the core of what we'll be working with.


You'll want to visit the IronPython homepage at ironpython.net to download the official version of IronPython.

Click the Download IronPython link. (At time of writing, August 2015, the newest version is 2.75 and the download link looks like the screenshot to the right.)

On the download page there are two download options. You can get the windows installer or a zip file that does not need to be installed. Since we're building a portable setup to easily transfer between machines, we want the zip file.
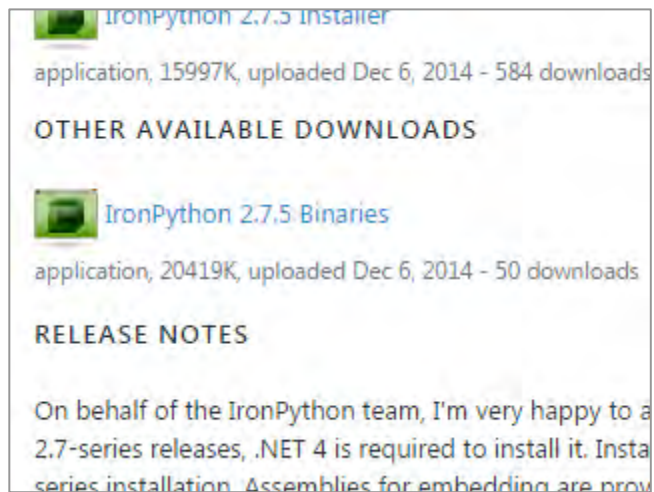
Click the "IronPython 2.7.5 Binaries" download link and save it in a location you can find it (usually defaults to the downloads folder.)
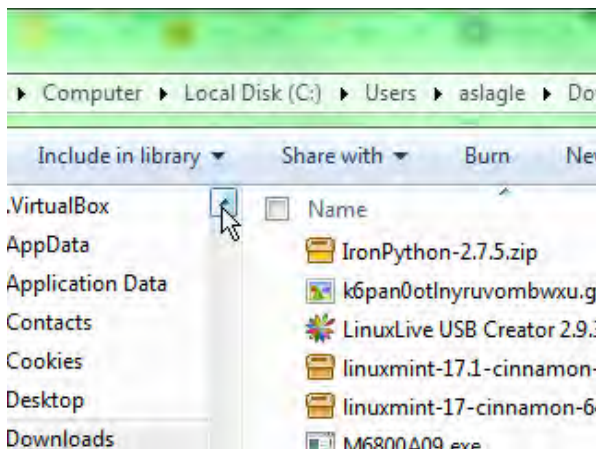
Double click the file (called something like "IronPython-2.7.5.zip") and Windows should automatically open it.
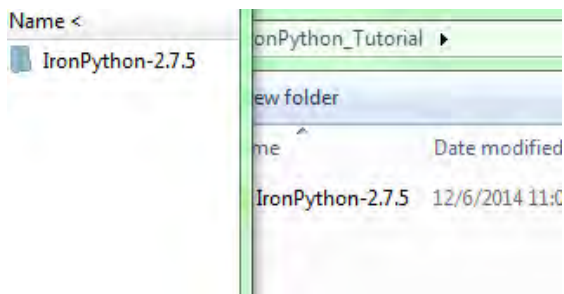
There should be a folder called IronPython-2.7.5 in the zip file. If you haven't already created a folder (on your USB drive maybe?) to hold your IronPython projects, create one. Then drag the IronPython-2.7.5 folder out of the zip file and into the working directory you created.

Getting the Okuma Open API

The Product Engineering Group at Okuma maintains a repository on GitHub which contains help files, libraries and examples for the Okuma Open API.

https://github.com/OkumaAmerica/Open-API-SDK

> The Okuma Open API help files with .chm extensions are help files and files with .dll extension are library files. The help files list all the functions and variables available in the Okuma Open API. The library files contain the functions and variables used by other programs to interact with data on the OSP-P control.

Copy the API help files and API libraries from the latest SDK release to a new folder in the same directory you extracted "IronPython-2.7.5" folder.
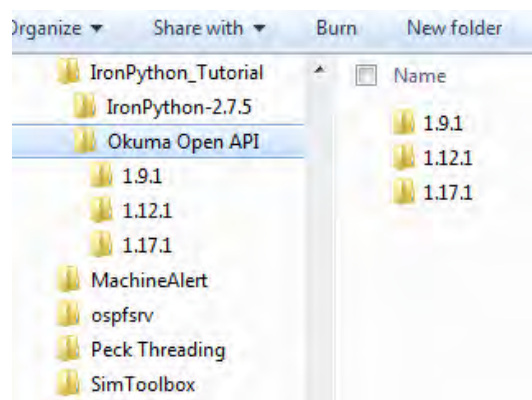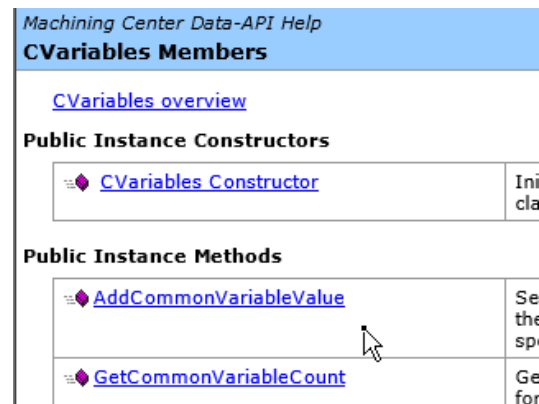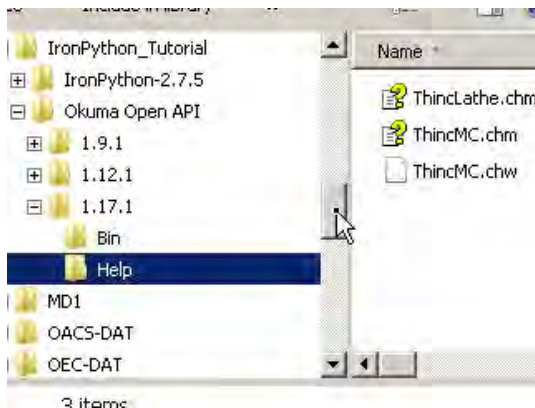
Create a new folder called "Okuma Open API" which contains folders for the recommended development versions of the API (currently 1.9.1, 1.12.1 and 1.17.1 - more on that later.)

The Interactive Interpreter

There are three ways you'll use IronPython.

1. From a live interactive IronPython command interpreter

    Executing python commands from an interactive prompt allows you to test and tweak functions and learn about their behavior with immediate feedback. This is like executing G-Code in MDI mode.

2. As an script you run using the IronPython interpreter

    A text file of commands can be run with the same result as if you typed them at the interactive prompt. This works much like executing a part program

3. As a compiled program

    This produces an executable .exe file you can share with other people the same as writing an app in C#.NET or Visual Basic.NET (the programming languages most machine tool app developers use.)

 In this step, we'll use the interactive command interpreter.
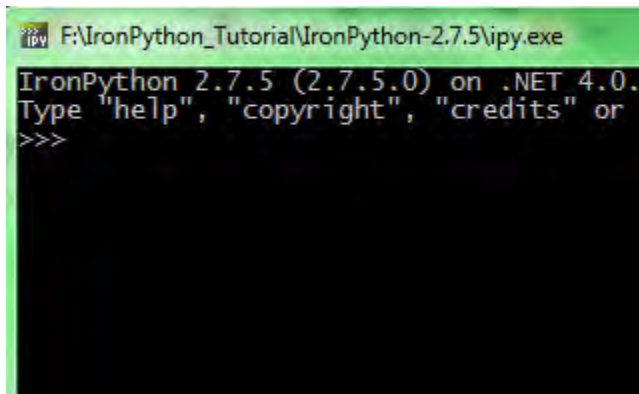

Launch the IronPython interpreter

In the location where you extracted IronPython-2.7.5.zip (or newer version maybe) there should be a file called either "ipy.exe" or just "ipy" if you have Windows set to hide file extensions.

Run this to launch the interpreter and a black box with some information and a command prompt should open.
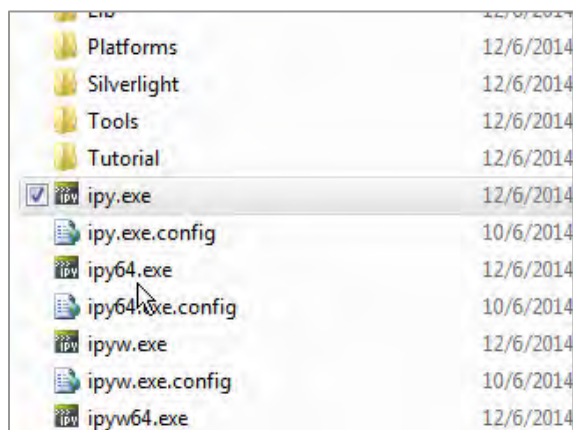
Basic usage

Entering commands in the interactive interpreter is like entering G-Code in MDI mode - it's executed immediately and you see the result. You enter commands at the command prompt which looks like ">>>" in IronPython.
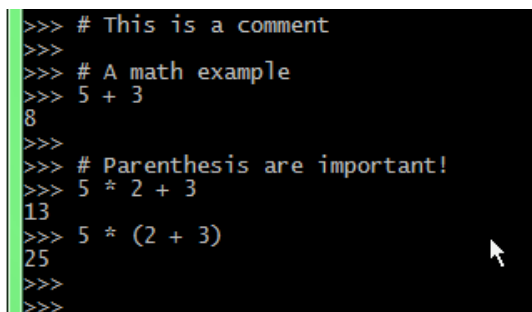

Comments help document your code so you can remember what you did or intended to do later. One way to add comments in Python is to begin with a "#" such as "# This is a comment"

> Comments are the most important thing you'll write in ANY coding language!

Just like G-Code, you can execute simple math in line such as "2 + 2"

Addition, Subtraction, Multiplication, and Division are all built in as "+ - * /" respectively.

Complex math should always use parenthesis to ensure proper order of evaluation.

> Parentheses ensure proper order of operation!
>
> "5 * 2 + 3" is not the same as "5 * (2 + 3)"

Libraries

For more functionality, you can import libraries of functions and variables using the import statement. One such library you'll commonly use is called "math" because (you guessed it) it contains math functions.

> Python is case sensitive meaning "math" is not the same as "Math"!

Try importing Math by typing "import math"

Once "math" is imported, you can access its contents by typing "math._____" and filling in the blank with what you want.

For example, try the variable "math.pi"

```
>>> # Importing a library
>>> import math
>>>
>>> # Using a variable in a library
>>> math.pi
3.141592653589793
>>>
>>> # Using a function in a library
>>> math.pow(5,2)
25.0
>>>
>>> # Area of a 4 inch diameter circle
>>> math.pi * math.pow(2, 2)
12.566370614359173
>>>
```

Exponents aren't standard in Python so they're included in the math library as the power function "pow()".

Try 5 squared "math.pow(5,2)"

And to apply it, find the area of a 4 inch diameter circle (pi r squared):

"math.pi * math.pow(2, 2)"

Your own variables

You'll definitely need to store some variables to make calculations easier.

You define a variable by writing "variable_name = value"

e.g. `inchesinfoot = 12`

You can see the value of a variable by typing it's name. Let's calculate rpm's needed for 100 feet per min surface speed on a 4 inch workpiece.

`cutspeed = inchesinfoot * 100`

`diameter = 4`

`rpm = cutspeed / (math.pi * diameter)`

Type rpm to see the value it calculated.

If this were a script, the number alone would be meaningless so you might want to print it in a sentence. Try this:

`print "RPM needed = " + str(rpm)`

`str` is a built in function that tries to turn whatever value you give it into a nicely formatted string.

```
>>> # Making your own variables
>>> inchesinfoot = 12
>>> cutspeed = inchesinfoot * 100
>>> cutspeed
1200
>>> diameter = 4
>>> rpm = cutspeed/(math.pi*diameter)
>>> rpm
95.4929658551372
>>>
>>> # Using variables to print text
>>> print "RPM needed = " + str(rpm)
RPM needed = 95.4929658551
>>>
>>>
>>>
>>>
```

Date and Time

Another common item needed in scripts is the Date and Time. Especially if you're going to write a log of events to a file, you need to record when the event took place. We'll import the `datetime` library to do this.

The `datetime` library has a class in it also called `datetime`. Classes are basically another grouping of functions and variables. A library could have several classes inside it. We'll use the `datetime.datetime.now()` function to get the current date and time. When you type that in and press enter, you'll see a cryptic output that has a series of numbers in it. We can try the `str()` function to clean it up.

Try: `str(datetime.datetime.now())`

That output looks better but for a log file it would be nice if the seconds didn't have so many decimal places. We'll make a variable to grab the current time and then we'll format it using a function of datetime:

`mytime = datetime.datetime.now()`

`mytime.strftime('%b, %d %Y %H:%M:%S')`

That looks like a mess to remember. I promise I have to google it every time. (I searched "how to format datetime in ironpython", clicked the first result and did some searching on the page until I found the `strftime` function.

strftime behavior

```
> # Getting the date and time
> import datetime
> datetime.datetime.now()
tetime.datetime(2015, 8, 19, 11, 1, 55,
> str(datetime.datetime.now())
015-08-19 11:02:08.185000'
>
> # Formatting time stamp
> mytime = datetime.datetime.now()
> mytime.strftime('%b, %d %Y  %H:%M:%S')
ug, 19 2015  11:02:29'
>
>
>
>
>
>
```

Standalone Script - Log Time To A File

Operator Dialog Box - Append Comments to a CSV

pop up a message box that says "Add comments" and store it to a CSV:

A	B	C	D
DEBUG	8/25/2015 9:02	Mundane message that c	
INFO	8/25/2015 9:02	Something that you migh	
WARNING	8/25/2015 9:02	Potential problem	
ERROR	8/25/2015 9:02	Something went wrong	
CRITICAL	8/25/2015 9:03	The world is ending! righ	

The API - Poll a Common Variable

Get Library Version

Reading and setting a common variable to the current time
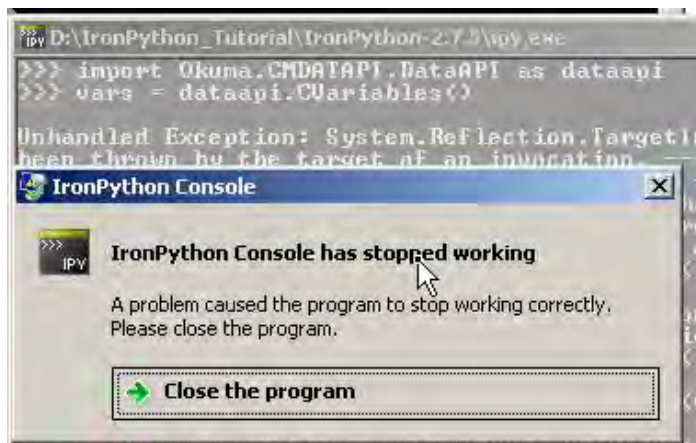
Send a Text or Email

send a text message when Part Count reaches 50:

Putting It All Together

A script to log cycle times and comments and automatically send a report and notification when parts are completed.

There's obviously *a trove* of detail that I'm leaving out and we can discuss more in the forum but I hope this set some hamsters running with ways scripting will make your life in the shop easier.

IronPython samples from Microsoft's Codeplex site: http://ironpython.codeplex.com/wikipage?title=Samples