1. Introduction – Me and Mike then anyone else who might be there.
2. Title of the presentation
3. Intro to LPS – We have over 8,000 employees and dozens of applications that provide a wide range of financial services. Headquarters are in Jacksonville, which is where our team works as well. With such a large application domain, we have many different quality assurance groups and each group often handles multiple applications.
4. Overview – So as the title of this presentation suggest I will be discussing writing Rational Functional Tester scripts using a programming language called jython. However first we need a little background information on Jython so that my demos and examples can be understood easier for those who are not very familiar with jython. If at any time during my presentation you have a question feel free to ask. First I will be explaining what Jython is, some of you may know, some may not, so I will briefly explain what Jython is. I will then be explaining why Jython is useful and how it can benefit normal development. I am then going to introduce jython and show some of the basic syntax features of Jython that you will want to know before looking at the Demos or other code snippets. After discussing Jython I will be discussing what we call the “JythonLauncher” which, as its name suggests, is used to launch execute the Jython scripts. Finally we will be diving into the actual development RFT script with Jython, including dealing with the PyDev debugger.
5. What is Jython? – So, what is Jython? Jython is a Java implementation of python. What that means is that it can utilize java code and be executed from java code. The normal python that most of you may be used to is a C implementation of python and has many extensions that are written in C/C++, Jython however can utilize Java for its extensions. It does this by compiling the python source into java bytecode, which then runs directly on a JVM. Also like Java it is objected oriented, so that aspect will be familiar to everyone here. I have included a link to the jython homepage on the slide, but a half a second on google can get you there just as easy.
6. Benefits – So why would you use Jython? For one it would give you access to all of the Java libraries and a majority of the python standard library and any extension libraries written in python. However you do not have access to the C-based libraries, since jython is ultimately run as java byte code it has the same limitations as Java. However there are some of the C-based libraries that have been ported, but for the remainder you can always use the java equivalents. Another benefit of Jython is faster development. Jython, like python, is considered an easy language to pick up and use, a majority of the features that make python a fast development language are still present in Jython. Another reason is that many feel it is easier to read and maintain because of its simple syntax. It is also easy to build modular programs with jython, its flexibility lends well to module-based development where the code is well separated and you can pull out and put in another class with relative ease. This all allows for Jython code to be very dynamic, and that is very important when attempting to automate applications that are constantly evolving.
7. > normal RFT – Which brings us to the benefit it gives to RFT scripting. Since jython is easy to learn that allows new developers to jump into RFT scripting quicker, create scripts quicker, and reduce the overall development time for a test bed. On top of that as I mentioned it has an extensive API that has a wide range of classes that can help speed up development of scripts. RFT can get complex, by allowing users to use whichever language they feel comfortable in it can increase development speed.
8. Loss of code – I know some people might be wondering how would this fit into their current project, or how could they utilize what they currently have already. With jython there is no loss of any code written in Java, jython can use almost anything in written java as an extension. This means that if you have team members that are comfortable writing in java there is no downfall to letting them continue. Both Java and Jython have benefits to them, the choice of language depends on the task and programmer.

JYTHON INTRO

1. So since I am not sure how familiar all of you are with python syntax I will briefly go over some of the basics syntax and a couple of the ones I found to be used commonly. To start jython has dynamic typing. So you don’t have to explicit declare type when defining variables. However keep in mind all object do have a type. Whatever is assigned to the variable decides the type, and as expected different types have their own unique methods you can use. The other thing to note is that jython is whitespace delimited. So spaces, tabs, and newlines determine blocks in code. As you can see in the code example the only thing that jython uses to determine the method definition is that all elements in the definition are at least one tab to the right.
2. Imports – So here are three examples of importing different types of classes. The first example is importing normal python standard library libraries. The second is for importing some of the standard java libraries. As you can see there is no difference really in the syntax between the two, just the path is different. And finally I included an example of importing some of the RFT libraries used in actual scripting.
3. Arrays – array syntax in jython is simple as well, though the one key point to notice is that arrays can contain any types of objects. However java arrays are usually of very specific types, in order for jython to choose the proper methods when calling java code that require arrays it needs to know the array types. The second code snippet is an example of how that is done, the array function in the jarray class allows you to convert a normal array of jython objects into a type specific array that can be passed to java code. Note however that all the items in the array must be convertible to the type given, otherwise it will throw an exception.
4. Class specific methods – while jython is dynamically typed it does not magically figure out what class contains the method you want to use. Since variables do have types behind the scenes you still need to keep in mind what types you are dealing with in a method and convert the object accordingly. Constructing java objects in jython is not that different from the java way, you just do not have the new keyword or the type declaration on the left.
5. Extending classes – extending java classes in jython is identical to extending other jython classes in jython. You define the jython class and in parentheses next to it are the classes it extends. One of the nice things about jython is that technically you can extend multiple java or jython classes from the same class.
6. Super – when interfacing and extending java objects sometimes you need to call the extended object’s constructor similar to normal java constructors. The syntax for this is what you see here. You provider the super method with the Object you are currently in and the current instance of that object. Then you provide the constructor’s parameters to the \_\_init\_\_ method which is jython’s style for constructors. The \_\_init\_\_ method is created at runtime for the java objects. They use this syntax because as I said it is the standard syntax for constructing python objects. Also the arguments to the \_\_init\_\_ call must be resolvable to java objects if the extended class is a java object.
7. Protected – Jython does not provide an easy way to access protected methods and attributes from extended classes. However there is a syntax for accessing these methods as you can see on the slide. The super double underscore prefix is appended to all the protected methods at runtime. However, protected attributes are still not accessible through this syntax. By setting the respectJavaAccessability flag to false in the jython registry file allows Jython to resolve the methods normally. This however bypasses the classic security features of using protected methods and attributes since you can access these outside the class. Some many feel it is worth it to lose those features since the security aspects are not necessarily needed for RFT scripting, but for the purposes of this presentation, I did not assume that this would be set to false.
8. Jython from java – using jython from java only requires that you add the jython.jar file that comes with your jython install to the classpath through the java Buid Path tab in eclipse. Once that is added you will get access to the PythonInterpreter class. This class and several supporting classes are what we use to execute Jython from java. There are two main ways to execute jython from java: through the exec command that takes a string argument with the jython code to execute, or it can execute a file directly. Both examples can be seen in the second code snippet on this slide.
9. Jythonlauncher – The JythonLauncher is our implementation of an rft script that can launch Jython scripts written utilizing the RFT API. It uses the PythonInterpreter class mentioned in the previous slide to execute the target scripts. I will show and example here in a moment. The reason we need a launcher like this is because RFT actually runs several class loaders when you start the testMain that allow the Jython scripts to use the RFT API. Another benefits are that it allows us to execute several scripts in sequence without launching new scripts. Also as we will discuss in more detail later you need to append PyDev’s debugger to the python sys.path before you can use it during script debugging and the JythonLauncher provides a convenient way to do so. The alternative to using the JythonLauncher or something like it is to reverse engineer the process RFT goes through on execution of testMain. We have attempted this before and ended up giving up and going with a simpler solution, but it is still theoretically possible to do.
10. Pydev – when developing in any programming language it helps to have an IDE to work in. Conveniently enough eclipse has a very nice plugin for developing python and jython called PyDev. PyDev can be installed along side RFT easily keeping all the script development isolated to a single tool. Since it is part of eclipse it is also highly customizable, allowing you to have a UI that fits your development style. PyDev is also well maintained and has a strong user and developer base so it does get updated with the different versions of Jython and Python. Also since it is a pretty popular addin to eclipse there is a lot of information online on how to configure the projects. Of course there is not really any information on integrating RFT and RFT libraries in with PyDev, but hopefully we cover most of it in this presentation.
11. Setup- There are a couple steps that you need to do to get PyDev working with Jython and RFT. The first is to configure Pydev’s configuration for the Jython interpreter. This involves going to the eclipse preferences for pydev and pointing the Jython interpreter to your local install of jython. Then you go to the project properties and configure it to us the jython interpreter you added. Next you add the project to the source folder list so that pydev recognized the python files under that project as valid files. I will show you these location here in a moment during the demo.
12. Setup cont. - The last step in setting up a PyDev project is to add any jars you need to the PYTHONPATH. This allows pydev to resolve the imports and give auto completion features to jython files. This slide gives a display of all the jars you would need for an RFT 8.2.1 install but to find out the jars you need for your installation they are located under the RFT Libraries and RFT TestObjects Classpath containers. I will show you where they are located this demo.

----------------------------DEMO------------------------

Basic Jython Demo

---------------------------END DEMO----------------------------------

Code deign:

1. So before I get into showing you the actual code behind the RFT scripts I want to explain our code design strategy. In normal Object oriented fashion we wrap up all of your code in methods and classes. This increases the consistency since if everything routes through a single function then you can insure everything acts the same. It encourages re-use of code so you are not re-writing the same thing repeatedly. And it increases clarity since you can condense a series of actions into a single call that is easy for someone to understand, such as login.
2. The other code design we have is that we separate application specific code and commonly used code. By building a framework around commonly used actions it allows us to easily add new applications and scripts while having the same underlying code. So we split our code into two sections, the Common project, which contains all the actions and methods that we found to be common across applications, and the Application projects which hold application specific code that would have little use when dealing with another application.
3. So, now I am going to get into how methods can interface with the RFT API. The first and probably most used function is a findObjects function. I have an implementation of these functions on this slide, some of the code has been simplified for demonstration purposes. These implementations include only two parameters, the object that you want to start looking from and a variable list of properties. It then calls the basic TestObject.find method which takes an Anchor object generated by a method such as SubitemFactory.atDescendant which takes the array of properties that you pass it. I included an example of the method written in java and Jython so you can see the parallels in the two languages.
4. In general we have found that there are 6 major actions that span across nearly all applications. They are type, select, toggle, drag, hover, and click. Using these actions you can perform basic GUI automation. However there are several other methods that are required for verification points or screen validations. I will not go over the code for all these action but if you are interested, they are included in the demo code that we will provide at the end of this presentation. Now that we have gone over the basics for how our framework is laid out I will get back into the Jython code.
5. Finding by regular expression is an important feature in RFT, often fields can change between version of applications or you may want to find multiple objects at once. It is easy to tap into this feature in Jython as well as java. The first code example shows this, it uses the findObjects method mentioned before but one of the items in the property array is an RFT RegularExpression object. The second example on this slide is closer to the style of coding that you will see in the demos. RFT uses object maps to define objects, and this example shows the syntax you would use to implement something similar in Jython. So it simply looks up the object definition in the global table and uses that as parameters into the find objects call. If you are curious in the map syntax you see under that findKnownItems function, that is a built in python method that can imitate iteration over a list of objects. As for the if \_\_name\_\_ == main thing, that is similar to java’s main method. It basically states that if the python file that that code exists in is the file that is called directly then the code under it will execute. All of the scripting code that you see written in jython will be under that if statement.

----------------------------DEMO------------------------

Now I will get directly into some of the Jython scripts.

RFT Jython Demo

---------------------------END DEMO----------------------------------

DEBUGGING

1. An incredibly important aspect of development is access to a good debugging tool. However since Jython is run from java and the Java debugger is different from the Python debugger you have to do a little more than just launch a script in debug mode. Pydec has what is called a “Remote Debugger” which can debug python code run from another process. In order to use this feature you need to start the debug server in pydev and add some additional code to the startup process for Jython. The complete code for this is included in the JythonLauncher class in the demo code, but I provided an example snippet in the slide. As you can see this code is executed before you even call a jython script, inside the wrapper RFT script.
2. Here are the steps to enable debugging. The first step is to set breakpoints in the script you are attempting to debug. The pydev remote debugging does allow for breakpoints to be set inside eclipse just like the java code. Second you start the pydev debug server.
3. Then you start the Jython launcher in debug mode. This tells the jythonlauncher that you want to debug instead of just run. Finally you launch the jython script using the jython launcher.
4. As you can see here it has all the same features as the java debugger that many of you might be familiar with.

----------------------------DEMO------------------------

Debugging Demo

---------------------------END DEMO----------------------------------

LOGGING

1. Log files are an important part of RFT scripts. In normal RFT scripts only a single log file is generated per script. However since the JythonLauncher is essentially a single RFT script only one log file is generated if you log messages through normal means. Also since the normal logging methods provided by RFT are static you cannot use them to create multiple logs. There are two solutions to this one is to artificially start and stop the logs. But since the code that that involves uses unsupported RFT methods that can cause problems, however the JythonLauncher we currently have does have the ability to do this. The other solution is to create a custom logging framework that allows you to customize how the logs are written. Both are viable solutions.

FUTURE

1. While working on this project there were several things we saw that would have been interesting projects that seemed worth mentioning. The first is replacing the JythonLauncher with an eclipse plugin. Rather than having to go to use a script the JythonLauncher could be built into eclipse. Another is to complete remove the JythonLauncher and reverse engineer the startup process for an RFT script with a testMain.
2. Interestingly enough there is another python distribution called IronPython which is a .NET implementation of pyton. The idea is that you could do the same thing we did with the Java RFT API and duplicate it for the VB.NET API. Which if done right could potentially allow a single RFT script to run as either Java or VB.
3. And finally you could include other automation tools such as Sikuli, which is a GUI automation too that uses Jython as well, along side RFT for even more options and power to control user interfaces.