Code Generator

Mike Hooper

This document gives an overview of the code generator. In addition is describes the step by step process needed to make a change to the structure of the files generated

Team Elderberry

Ron Astin, Chris Glasser, Jordan Hewitt, Mike Hooper, Josef Mihalits, and Clark Wachsmuth

2/13/2013

Overview

This document is divided into two main sections. First is understanding, this section is dedicated to the research steps the user should take before attempting to modify the code generator. Full understanding of these research steps is key to the success of being able to modify the generator. The second section is extending the generator. This section is dedicated to a step by step tutorial-by-example on extending the code generator.

Understanding

This section encourages the user to take the necessary steps needed to understand how to modify the code generator.

Miml

Firstly the user should read the miml documentation. It is located <here>. It is good to have a complete understanding of the miml language and how it works. Miml is very closely related to YAML. Documentation for YAML can be found here. http://www.yaml.org/spec/1.2/spec.html#Introduction

Code Generator

Secondly the code generator. Read the comments in codeGen.py

Please pay particular attention to the following

a) The parser class and its role in controlling everything. (states/transitions)
b) outputGenerator class and what it's used for
c) ErrorLogger class and how to use it, especially during validation
d) ParseHandler class, cg.conf and the patterns used in the defined handler functions in parseHandler
e) Debugging/Display features of Parser.transition and OutputGenerator display.

Extending the Generator

In this section we will extend the miml language and show the user the step by step process to do so.

Adding Author

Here we will add an author tag expecting a name and an email address. It will add a comment saying "x.h was written by 'author'" We will be working in the miml_examples folder from the root structure of the repository.

Logger.miml

In the Logger.miml file, we need to add the tag for the author at the top of the file, with the other hashkeys.

```
%YAML 1.2
---
include: logger.h
object: logger.o
init: loggerInit();
final: loggerFinal();
author:['Author_Name','Author_Name@gmail.com']
```

If you run the code generator now, it will fail. It contains illegal component "author". Since it doesn't have a handler, it will error out.

cg.conf

Here we will add the function call to the handler for our new author tag.

At the bottom of the cg.conf file, add the handler

```
      # Paramater validator
      validate_receivers: {path: '/modules/*/receivers/*', type: 'list'}

      # Miml Make files to track
      make_miml: {path: '/make_miml', type: 'list'}

      #author handler
      handle_author: {path: '/modules/*/author', type: 'list'}
```

if you run codegen now, it will fail. This time it will crash, since you are calling a function that has not been defined.

codeGen.py

At the bottom of codegen.py is where you define the author handler function.

```
def handle_author(self, data):
      p = self.parser
      e = p.errors
      o = p.output
      If p.stat == ParserStates.Validate:
         If len(data) != 2:
           e.new_error("Illegal author component in" + '/'.join(p.path))
      Elif p.state == ParserStates.parse:
         #Pass Note below
         o.append("code", 8, "// " +
         p.master['modules'][p.ptath[2]]['include'] + "was written by" +
         data[0])
         o.append("code", 8, "//  send inquiries to " + data[1] + "\n")
      Return false

parser = Parser('./cg.conf')
parser.parse()
```

In the if/elif portion of the code is where things are different. The rest of it is quite uniform. The If statement is purely checking to see if the data passed through contains two elements. Remember, our author tag is looking for a name and an email address only. So if the user adds a third argument, it will pass an error. In the elif statement the o.append arguments have an integer argument of '8'. This argument makes sense if you go to the parse function definition in codeGen.py and uncomment self.output.display() This will print to terminal a structure of the code, so you can append the data in the right part of the structures hierarchy. While writing the code, since the user won't know which level to put the append in, replace the elif section with 'pass' after uncommenting self.output.display() and run the program.

Conclusion

In conclusion, if there is no author listed in the miml file, nothing will happen, it will carry on as normal. However, if there is an author defined in any miml file, it will be handled and added to the files as they are generated.