Source Code Feature Extraction:

Developer’s Guide

Developers looking to extend the Source Code Feature Extraction pattern library will want to do three types of things: Add a new language to the library, create a new pattern for an existing language, and modify an existing pattern. We describe how to do each of these tasks below:

# Adding a New Language

1. Make an RPL file for the new language in the langlib directory.
2. Pick a common file extension for the language to be used in the pattern names as the language identifier.

e.x. C# -> cs (cs.line\_comments)

1. Add a directory in testfiles with the language name.
2. In regression.py in the “langs” dictionary, add a key listed as the same directory in step 3, and then set the associated value as the language identifier.

# Creating a Feature for a Pre-Existing Language

1. Pick a language.
2. Go to the RPL file associated with the language.
3. Make a block comment describing the pattern and the structure of the new pattern. Find at least 3 input test files on GitHub or create them by hand.
4. The block comment should be a nested structure of how the patterns are going to look in the output, along with an explanation of the various patterns.
5. Manually verify that the output of the pattern when run against the selected test files matches what is expected.
6. Place the input files in testfiles/<language>/input/<feature>/ and rename it to <pattern\_name><test-id>.<language-extension>.

e.g. classes0.java

1. Place the verified output files in testfiles/<language>/input/<feature>/ and rename it to <pattern\_name><test-id>.json,

e.g. classes0.json

1. In regression.py add the feature name to the “test” array.

# Modifying an Existing Pattern

1. Modify the pattern in the associated rpl file.
2. Run regression.py to ensure the changes didn’t break anything.

# Running “regression.py”

The regression script uses the file/directory structure of the “testfiles” directory for all of its pattern execution purposes. Below is the expected structure expected for input and output files:

Input File: “testfiles/<language>/input/<feature>/<pattern><numeric id>.<language file extension>”

Output File: “testfiles/<language>/output/<feature>/<pattern><numeric id>.json”

Where “pattern” dictates the pattern to be run WITHOUT the language identifier, language is the language associated with the pattern, feature is the associated with the pattern, language file extension is simple a source extension, and numeric id is some number associated with the test case.

The “language” specified is used by the regression script to match the correct language identifier with the test. This is handled by a dictionary maintained by the regression script, and any future languages (directories) must be added to this dictionary. The supplied “ feature” directory is also maintained in the python script to maintain a list of testable features.

An example:

Input File: “testfiles/java/input/comments/line\_comments0.java”

Output File: “testfiles/java/output/comments/line\_comments0.json”

Will cause the regression script to run the Rosie pattern “java.line\_comments” against the file “line\_comments0.java”, and will compare the output to the json found in “line\_comments0.json”.

In order to run the script use the following command at the root directory of the langlib repository

$ python regression.py <build number>

This will create a file “results<build number>.html” in the root directory, which contains a pretty diff of any differences found between input and output files.

# Useful Language Documentation

We provide resources which we found helpful when developing the patterns in the library.

## Java

* language specification: https://docs.oracle.com/javase/specs/jls/se8/html/index.html

## C

* Language specification: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf

## C++

* C++ standards, working drafts can be found for free: https://isocpp.org/std/the-standard

## C#

* Language specification: https://docs.microsoft.com/en-us/dotnet/articles/csharp/language-reference/language-specification

## Python

* Python2 language reference: https://docs.python.org/2/reference/
* Python3 langauge reference: https://docs.python.org/3/reference/

## JavaScript

* List of useful javascript resources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language\_Resources

## Ruby

* Ruby documentation: http://ruby-doc.org/core-2.4.1/

## R

* Language definition: https://cran.r-project.org/doc/manuals/r-release/R-lang.pdf

## Go

* Language specification: https://golang.org/ref/spec

## Bash

* Reference manual: https://tiswww.case.edu/php/chet/bash/bashref.html

## VB

* Language specification: https://docs.microsoft.com/en-us/dotnet/articles/visual-basic/reference/language-specification