**Packages within source code**

**Engines**

Provides an interface for running the generated programs on each of the supported engines.

**Files:**

* **\*Engine –** an interface for a specific engine (for example: DynjsEngine)
* **RunEngineResult –** Defines a result of a specific run on a specific engine.
* **EnginesCompareModel -** Provides an interface that enables us to compare output and results of programs, after running them on each of the engines.
* **EnginesUtil –** Provides the compare function, for comparing results from different engines.

**Generator**

This package provides the generator, a tool for generating random JavaScript programs.

**Files:**

* **ApiOptions –** enum for different API functions available in the generated JavaScript program (these functions are injected to the beginning of each program).
* **Context –** Defines a program context while generating program. Contains the symbol table, current loop depth, etc.
* **ExecFlow –** enum for the execution flow logs options.
* **Generator –** Generates a JavaScript program. The generation is recursive. Each JST (JavaScript AST) node has its own 'create' function in the generator. Each such function generates the specific node according to the statistical model defined by the configuration file.
* **GenLogic** – provides the functions for generating expression and statement nodes.

**Sub Packeges**:

* **Config:** an interface provided to the generator, for reading the statistical configurations.
* **Params:** createParams is an abstract class and each of the other \*Params classes extends it. Each such class defines a parameter provided for a specific creation function in the generator. For example, IdentifierParams is provided as a parameter for the CreateIdentifier functions within the generator.
* **SymTable:** the SymTable object is being created through the process of the program generation. It contains both functions and variables declared within the enclosing scope.

**JST**

This package defines the Abstract Syntax Tree of the partially JavaScript language this tool generated programs of.

The AST hierarchy



Other classes:

* **Helper.Factory –** provides singleton nodes for the program creation (literal nodes, 'continue' node, etc.)
* **Visitors.JsToJs –** converts an AST to an actual javascript program.
* **Visitors.JsToTree –** given an AST, return a String representation of the tree.

**Main**

* **JSFuzzer –** contains the main functions of the program.
* **JSFuzzerArgs –** provides the functions for arguments parsing.

**Utils**

* **FilesIO –** provides functions for writing/reading data to/from files.
* **OutLog – manages the program logs.**
* **StdRandom - provides methods for** generating random number from various distributions.
* **StringCounter –** manages the counted variables generated in the program (v1, v2, …)
* **ThreeVal –** a three value enum (true, false, undefined) used by the generator.
* **TimerRunner** – capable of measure execution time for a given task. Used when running JavaScript file over specific engine to get its runtime.

**Folders within Resources**

**Config –** this file defines the statistical model, which the generator will use to create programs.

**Snippest:**

* **BrowserLauncher –** enabling us to run the generated javascript programs within a browser (by wrapping the code with html).
* **rawUtilsFunctions –** this file will be injected to the file, as is, before writing the generated program to it. It enables us to create and object (named as '$'), provides some api functions, for disabling some known bugs (or even differences) on different engines. For example, some engines support the 'print' function, while others support the 'console.log' function (both functions prints to the output stream).Using a pre-written print function (written in this file), we replace each print expression in the generated program with the pre-written print function. This function gives the same result on different engines.