Fasper Manual

Version 0.9

Donated by AMIQ srl

Contents

1 Introduction

1.1 About Fasper Donation

Here are few of the reasons that determined us to donate fasper to the open source community :

- ✔ strong believe in open source community principles
- ✔ to encourage antlr based tools development and verification
- ✔ to give a hand on antlr quality assurance

1.2 About Fasper

Fasper is a framework for testing parsers. It contains the framework, default implementations for few components and a graphical interface for easy debugging. The user can write it's own plug-ins to accommodate Fasper to it's needs.

1.3 How to use Fasper?

There are two ways of using ETools:

- ✔ as a component of your application – **Tool**
- ✔ as an application – **Fasper**

1.4 About this manual

This manual contains a detailed overview of the **Fasper Package** and all the information you might require for installing the **Fasper**, integrating it with your environment or using application.

1.4.1 Terminology

Next tables contain terms used in this manual.

Table 1-1 ANTLR related terms

Term	Explanation
Activation	An operation by which a component is made active. For example a parser is activated after it has parsed a source file.
AST	**A**bstract **S**yntax **T**ree is the result of the parsing process.
AST node	A node in the AST. It's main characteristics are type and text.
Golden model (aka gm, reference AST, expected AST)	A model that can be used as reference in a comparison process.
Parser Activation	A component that activates a parser : parses a source file.
Parser Under Test	Parser to be tested.

Term	Explanation
Resulted AST	The AST resulted from parsing the source file using the parser under test.
Source File	A text file that can be parsed.
Test Activation	A component that activates a test : reads the test file and extracts reference AST and other informations.
Test File	A file containing a reference AST and information about how a parser can be activated.
Token Mapping	A mapping between a string and a integer, representing token's type. This mapping can be extracted from *TokenTypes.txt.

1.4.2 Visual Conventions

This manual uses visual cues to help you locate and interpret information easily. These cues are explained in next table.

Table 1-2 Visual Conventions

Visual Cue	Explanation
arial font	Text body.
arial bold font	Highlights a particular element.
`courier font`	Indicates code.
`courier bold font`	Highlights a particular code element.
`courier font`	Highlights code comments.
<#.#>	Indicates a variable eg installer version.
`$>`	Unix prompt.
`%>`	Windows prompt.

1.5 Where to find more information?

You may find links to most important resources in **Resources** section.

2 Installation Notes

2.1 System Requirements

- ✔ **OS** : Windows, Linux, Solaris
- ✔ **Java JDK**: 1.4.2+
- ✔ At least **256Mb RAM** for fast execution

NOTE! Fasper was tested only under Linux!

2.2 Installation Steps

Fasper can be downloaded in two versions : **lite** and **full**. Lite version contains only the needed jars to run Fasper and documentation (only PDF and javadoc), while full version contains the whole project (including sources and docs in sxw format).

Installation is very easy :

- ✔ **Download fasper-<#.#>[-src].zip**
- ✔ **Unix**: open a console and type :

```
$> unzip fasper-<#.#>[-src].zip
```

Windows: double-click installation zip file in a file browser

- ✔ For a full **install**, you should have a directory structure like:
  - **external** : antlr-<#.#.#>.jar and junit-<#.#.#>.jar
  - **src** : sources directory
  - **build** : class files
  - **docs** : FasperManual.pdf, FasperManual.sxw, API documentation
  - **LICENSE.txt, README.txt, CHANGELOG.txt, TODO.txt** info files
  - **build.xml** and **build.properties** ANT build files
  - **fasper.jar** : the fasper library
- ✔ For a **lite install**, you should have a directory structure like:
  - **lib** : fasper.jar, antlr-<#.#.#>.jar and junit-<#.#.#>.jar
  - **docs** : FasperManual.pdf, API documentation
  - **LICENSE.txt, README.txt, CHANGELOG.txt, TODO.txt** info files

2.3 Setup Fasper Lite

Include **fasper.jar** and **antlr.jar** libraries in your CLASSPATH :

```
$> export CLASSPATH=${CLASSPATH}:/fasper/lib/fasper.jar:/fasper/lib/antlr-
<#.#.#>.jar
```

2.4 Setup Fasper with Sources

Next steps are necessary :

- ✔ include **fasper.jar** and **antlr.jar** libraries in your CLASSPATH :

```
$> export CLASSPATH=${CLASSPATH}:/fasper/fasper.jar:/fasper/external/antlr-<#.#.#>.jar
```

- ✔ make sure you have ANT in your path
- ✔ accommodate **build.properties** to your needs

2.5 Installation Troubleshooting

If you encounter problems during process please report them to stefan@amiq.ro.

3 Requirements

This is the list of requirements that Fasper tries to solve.

3.1 Checking

- ✔ check both node type and node text
- ✔ control comparison process (eg ignore AST nodes)
- ✔ run self-checking tests based on a external golden model (eg compare the parser AST with an AST saved in a file)
- ✔ run self-checking tests based on a internal golden model (eg compare two ASTs generated by two tree parsers)
- ✔ accurate comparison mismatch signaling

3.2 External tools integration

- ✔ integrate with any kind of parser (ANTLR related or not)
- ✔ user defined test activation
- ✔ user defined comparator
- ✔ user defined AST filter
- ✔ user defined test writer
- ✔ integrate with other tools (eg Junit)

3.3 User interface

- ✔ load/reload test action
- ✔ configuration action
- ✔ save self-check test action
- ✔ display parser/tree parser AST
- ✔ display target AST vs golden model AST
- ✔ display/edit source
- ✔ display test debug information (eg “console”)

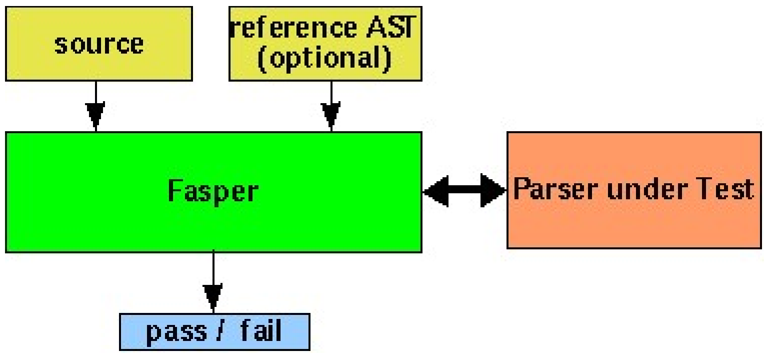
3.4 Other

- ✔ plug-in editor/generator
- ✔ text mode UI

4 Implementation Details

The basic idea behind Fasper is presented in next photo:

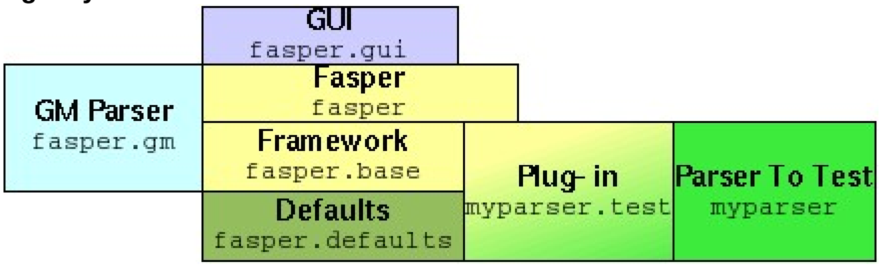
Picture 4-1 Basic Scheme



Fasper reads the test file and extracts the reference AST and any parameters needed to activate the parser under test. The parser under test is then activated and source code parsed. At the end reference AST and resulted AST are compared taken into consideration any comparison constraints (eg nodes that should be filtered). Fasper communicates with the parser under test through a java interface (a set of interfaces), which can be extended for any parser (even for C/C#/C++ ones by writing an adapter!!!).

From a java perspective we have next package layout:

Picture 4-2 Package layout



As you can see there are few packages with well specified functionality:

- ✔ **Framework** contains interfaces and classes that are common to plugins and fasper (eg. interfaces for AST comparators, AST filters, exception)
- ✔ **Defaults** has default implementations of some of the components defined in **Framework**(eg DefaultASTComparator which implements ASTComparator)
- ✔ **GM Parser** is a default implementation for test parser (see : Test File Format)
- ✔ **Fasper** contains a factory to instantiate plugins and an interface to accept commands
- ✔ **GUI** contains classes of a basic user interface
- ✔ **Plugin** contains the customized components (implement functionality defined in `fasper.base`)
- ✔ **Parser to test** is user parser

Note!

- Package fasper.defaults does not exist yet; for the moment it's contents is in fasper.
- In future releases fasper.gm will move to fasper.defaults
- Plugins definition and activation are not finalized yet (plugin concept has to be refined).
- User parsers are limited to ANTLR generated parser (uses AST derivations).

Most of the issues above will be fixed starting with 1.0.

4.1 Test File Format

At this moment the test is kept in a separate file for easy debugging (tried to break data from the model). The test file has a simple, readable format, with a look&feel very close to the trees displayed in a ASTFrame. Bellow is an example:

```
dutActivationClass etools.test.EToolsActivation
argTypeAndValue java.lang.String #"test.e"#
-->
[E_PROGRAM, #"top: test.e"#]
|--[E_CLUSTER, #"cluster: test.e"#]
|  |--[E_MODULE, #"module: /home/stefan/fasper/tests/test.e"#]
|  |  |--[PREPROC_DEFINE, #"#define"#]
|  |  |  |--[BACKTICK, #"backtick"#]
|  |  |  |--[ID, #"`MDA"#]
|  |  |  |--[PREPROC_DEFINE_REPLACEMENT, #"struct s {}"#]
|  |  |--[EXTEND_STRUCT, #"extend"#]
|  |  |  |--[ID, #"sys"#]
|  |  |  |--[STRUCT_MEMBER_BLOCK, #"struct_member_block"#]
|  |  |  |  |--[METHOD_DECLARATION, #"method_declaration"#]
|  |  |  |  |  |--[ENCAP_OPT, #"encap_opt"#]
|  |  |  |  |  |--[ID, #"run"#]
|  |  |  |  |  |--[PARAMETER_LIST_OPT, #"parameter_list_opt"#]
|  |  |  |  |  |--[RESULT_OPT, #"result_opt"#]
|  |  |  |  |  |--[EVENT_OPT, #"event_opt"#]
|  |  |  |  |  |--[K_also, #"also"#]
|  |  |  |  |  |--[ACTION_BLOCK, #"action_block"#]
|  |  |--[K_struct, #"struct"#]
===
```

Test starts with the tested parser activation and the name of the source file (value for the parameter of the constructor). Then tree data is put between --> and === markers; these markers must be alone on their lines.

The tree data consists of :

- tree construction informations : siblings & children positioning
- node data (token text & type)

Any node is written between two square brackets and consists of :

- ✔ the token type (eg EXTEND_STRUCT)

- ✔ a comma
- ✔ the text of the token, between #” ”# markers (instead using quotes “); for example

Note!

The format of the test file will be changed if users will decide so. XML format is not very readable and hard to debug.

Must discuss with ANTLR group members.

5 Plugin

5.1 Step 1 : create a new package for plugin

If your parser lives in the package called `myparser`, create a new package called, for the sake of the example, `myparser.test`. Assuming your parser is a java one, then it's enough to make a directory `test` under directory `myparser`.

5.2 Step 2 : create an Parser Activation

Now we have to make a class that :

- ✔ knows to make an instance of your parser
- ✔ is able to command your parser to load a file
- ✔ is recognized by fasper factory as an **Parser Activation** class
- ✔ can retrieve the parser generated AST
- ✔ can retrieve the tree parser generated AST; if there isn't such a thing, than it should return parser generated AST
- ✔ can retrieve the list of “accumulated” errors (if your parser is able tot handle multiple errors)
- ✔ can retrieve the token's text-type mapping

Bellow is an Parser Activation class example:

```
package myparser.test;
// you may customize list of imports to your needs
import fasper.base.*;
import antlr.*;
import antlr.collections.*;
import java.util.*;
import myparser.*;

// implement fasper.base.ParserActivation to make the class activable by
// fasper factory
public class MyParserActivation implements ParserActivation {
   // parser under test component
   private MyParser  myp;

   // tree parser under test (if any)
   private MyParserWalker mypWalker = null;

   // the name of the loaded file
   public String filename;
     // activates multiple error handling (if there is one)
   private boolean multiError = false;
```

```
// this is mandatory and must contain the full name of the class under
// test in our example can be one of myparser.MyParser or
// myparser.MyParserWalker; it will disappear starting with 1.0
public static final String CLASS_UNDER_TEST = "myparser.MyParserWalker";

// a basic constructor which initialize file name and multiple error mode
public EToolsActivation(String filename, boolean tmp) {
   this.multiError = tmp;
   this.filename = filename;
}

// activates the parser; is called by fasper
public void activate()
   throws TokenStreamException, RecognitionException, MyParserException {
   // create a new instance
   myp = new MyParser();
   // set multiple error mode
   myp.setMultipleErrors(this.multiError);
   // command parser to load a file
   myp.load(filename);
   // extracts the parser generated AST; in this case the generated AST is
   // a custom one, respectively MyAST
   MyAST pAST = (MyAST)myp.getAST();
   // if parser generated AST is not null and we have a tree parser,
   // we can try to extract an tree parser generated AST
   if (pAST != null) {
      // create new instance of the tree parser
      mypWalker = new MyParserWalker();
      // set node of the generated AST to MyAST
      mypWalker.setASTNodeClass("myparser.MyAST");
      // parse tree using top rule program
      mypWalker.program(pAST);
   };
}

// return the parser generated AST
public AST getParserAST() {
   if (myp != null)
      return myp.getAST();
   else
      return null;
}

// return the tree parser generated AST; if you don't have a tree parser,
    // then you return the parser generated one
    public AST getTreeParserAST() {
       if (mypWalker != null)
          return mypWalker.getAST();
       else
          return null;
    }

    // returns list of errors if there are any
    public ArrayList getAllErrors() {
       return myp.getAllErrors();
    }

    // returns the token's mapping type-text
    public HashMap getTokenMap() {
       return myp.getTokenMap();
    }
 }
```

5.3 Step 3 : create an AST Comparator

The AST comparator is the central piece of the fasper; here the ASTs are compared under user defined constraints. Basically it :

- compares reference and resulted ASTs at tree level (have the same structure)
- compares reference and resulted ASTs at node level (have the same type and text)
- computes the mismatch point and signals it
- applies comparison restrictions at tree level (ignore tree branches and/or nodes)
- applies comparison restrictions at node level (ignore text or filters text to eliminate noise)

The fasper package comes with a default implementation called `fasper.DefaultASTComparator`

NOTE!

Example will be available starting with 1.0

In 1.0 Zrcomparator will be part of the `fasper.defaults`.

5.4 Step 4 : create an AST Filter

Filters are components that embody user defined constraints. AST comparator applies them to tree/nodes during comparison process. There are two main categories:

- ✔ tree level filter : ignore branches or nodes of the tree
- ✔ node level filter : filter text to eliminate noise

First category is useful when you test a portion of a grammar and want to ignore the irrelevant pieces. The second one helps you ignore text noise like white spaces, machine dependent content etc.

Basically a filter must :

- ✔ identify the target node/branch
- ✔ apply user defined constraint
- ✔ be recognizable by fasper

Here it is an example:

```
package myparser.test;
// you may customize list of imports to your needs
import fasper.base.*;
import java.util.*;
import antlr.collections.*;
import java.io.*;

// implement fasper.base.ASTFilter to make the class activable by
// fasper factory and AST comparator
public class MyFilter implements ASTFilter {
   // token's mapping text-type
   private HashMap tokenMap = null;
   // contains mapping (token-type, filter-to-apply)
   private HashMap filters = new HashMap();
   // contains mapping (token-type, true)
   private HashSet ignored = new HashSet();

   // initialize filters and ignored fields
   {
       // implement a filter which keeps the last element of a file path
       StringFilter one = new StringFilter() {
            public String filter(String input){
               String[] _res = input.split(File.separator);
               if (_res.length > 1)
                  return _res[_res.length - 1];
               else
                  return input;
            }
       };
       // associate filter “one” to three different token types (nodes)
       filters.put(new String("ANOTHER_AST_NODE"), one);
       filters.put(new String("PLICTISEALA_1"), one);
       filters.put(new String("HAHAHAHA_3"), one);
       // implement a filter to eliminate path noise
       StringFilter two = new StringFilter() {
               public String filter(String input){
                  String[] _res = input.split(File.separator);
                  if (_res.length > 1)
```

```
                    return _res[_res.length - 1];
                 else
                    return input;
              }
           };
        filters.put(new String("SOME_MODULE"), two);
        filters.put(new String("E_NA_FILE"), two);
        // set the nodes to ignore
        ignores.put(new String(“IGNORE_ORBIT”), true);
        ignores.put(new String(“IGNORE_MENTOS”), true);
     }

  // called by AST comparator during comparison on both reference and
  // resulted nodes
  public String filterNode(AST _node) {
     if (tokenMap == null)
        return _node.getText();
     String _s1 = (String) tokenMap.get(new Integer(_node.getType()));
     String _s2 = _node.getText();
     if (filters.containsKey(_s1))
        return ((StringFilter) filters.get(_s1)).filter(_s2);
     return _s2;
  }

  // called by AST comparator during comparison on both reference and
  // resulted AST trees
  public boolean isNodeIgnored(AST _node) {
     if (tokenMap == null)
        return false;
     String _s1 = (String) tokenMap.get(new Integer(_node.getType()));
     if (ignored.contains(_s1))
        return true;
     return false;
  }

  // sets the token's mapping to use
  public void useTokenMap(HashMap _tokenMap){
     tokenMap = _tokenMap;
 }
}
```

Note!

Tree level filtering will be available from 1.0

5.5 Step 5 : create a Test Parser Activation

Note!

It's definition and functionality will be better defined in 1.0

Example will be available in 1.0

5.6 Step 6 : modify preferences to integrate your plugin

In order to integrate your plugin you have to modify **resources/init.init** file, inside the jar (#!@#$@#$@).

You have to change :

- **TEST_EXT** with the extension of your test files; defaults to “z”
- **DUT_EXT** with the extension of your source files; defaults to “e”
- **DUT_ACTIV_CLASS** to your default parser activation (eg `myparser.test.MyParserActivation`)
- **DUT_ACTIV_ARGS** to a list of parameters of the default parser activation (eg. DUT_ACTIV_ARGS=java.lang.String java.lang.boolean)
- **DUT_ACTIV_VALUES** to a list of values for the parameters defined in **DUT_ACTIV_ARGS** (eg. DUT_ACTIV_VALUES=src_file.c true)
- **FILTER** to your default AST filter (eg `myparser.test.MyFilter`)

Note!

For the time being plugin integration is not well defined, as the concept of plugin is not well defined. It will change in 1.0.

5.7 Step 7 : create a Test Writer

If you have your own test format you can use a custom test writer to save the golden model. All you have to do is to create a class which implements `fasper.base.TestWriter`, as it follows :

```
package myparser.test;

import fasper.base.*;
import java.io.*;
public class  MyTestWriter implements TestWriter {
   public MyTestWriter(){}
      public void writeASTToFile(AST _tree, String _fileName)
         throws Throwable
      {
         /**
            put your AST-to-String code here
         **/
         File _file = new File(_fileName);
         try {
            BufferedWriter w = new BufferedWriter(new FileWriter(_file));
            w.write(someASTString);
            w.flush();
            w.close();
         } catch (IOException _ioe) {
            throw new FasperException("UNABLE_TO_SAVE_TEST"
            , "Unable to save test to file :" + _file.getAbsolutePath()
            , _ioe);
         }
   }
}
```

NOTE!

Test Writer should save the whole test to a file (using string templates). This feature will be available starting 1.0.

5.8 Step 8 : Write the test

Now it's time to write the test. Bellow is a piece of code which activates the fasper :

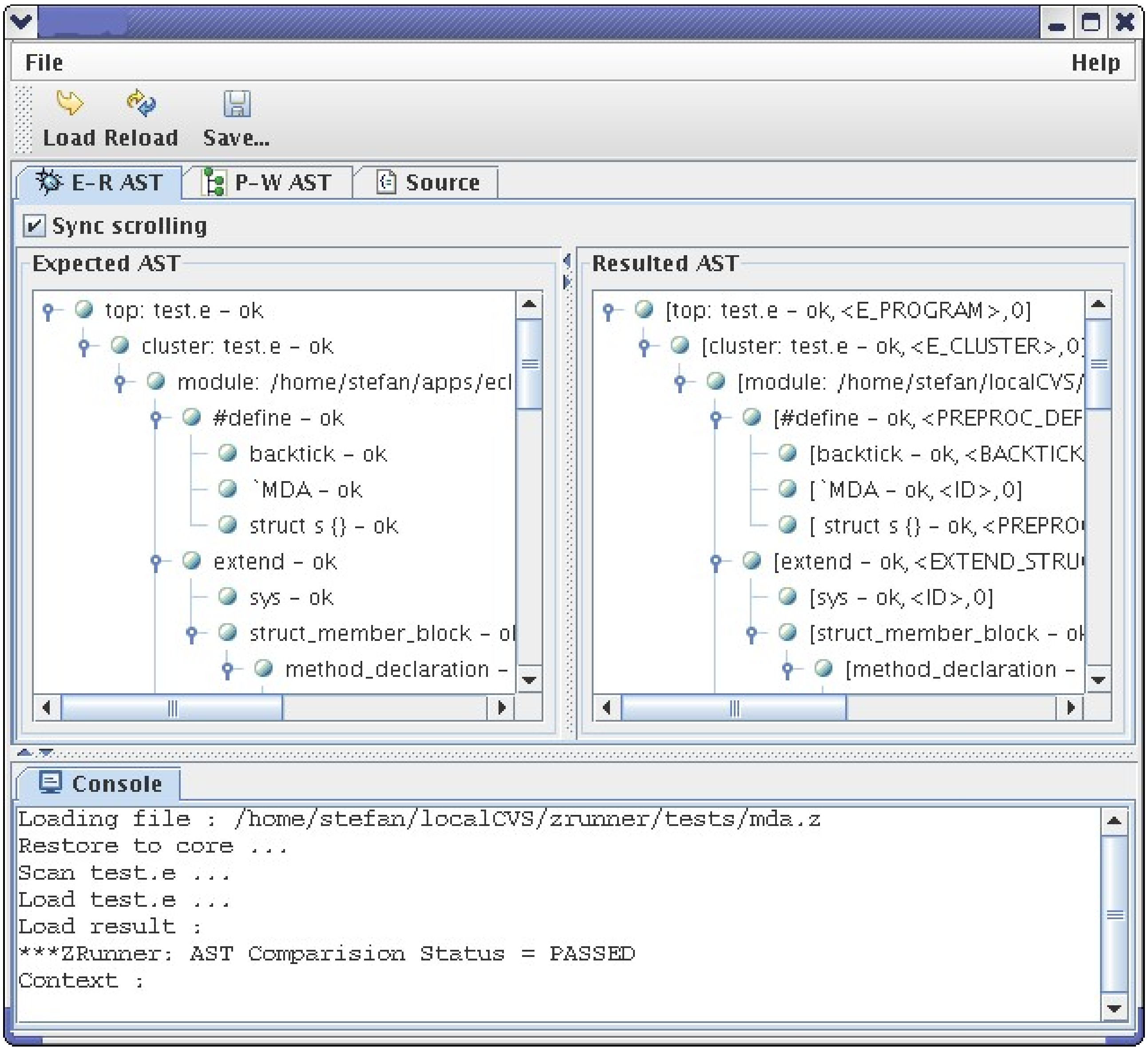
```
try {
   fasper.Tool tool = new fasper.Tool();
   tool.load(new File(_test));
} catch (ComparisonException _cex) {
   if (!_cex.getErrorCode().equals("PASSED"))
       p("Error: Tree compare : Failed!\n" + _cex.toString());
   else {
       p("***  SelfCheck Test Passed!");
       p("*** Tree compare : Passed!");
   }
} catch (FasperException _fe) {
   p("Error: Tree compare : Failed!\n" + _fe.toString());
} catch (Throwable _thr) {
   p("Error: Tree compare : Failed!\n" + _thr.toString());
   p("***Error: Exception : " + _thr.toString());
}
```

6 Fasper GUI

Fasper package contains a simple GUI for easy debugging of tests. It is able to load/reload tests, show the comparison mismatch, display the process.

Next picture represents a panoramic view of the Fasper GUI.

Picture 6-1



Expected AST:

```
top: test.e - ok
  cluster: test.e - ok
    module: /home/stefan/apps/ecl
      #define - ok
        backtick - ok
        `MDA - ok
        struct s {} - ok
      extend - ok
        sys - ok
        struct_member_block - o
          method_declaration -
```

Resulted AST:

```
[top: test.e - ok, <E_PROGRAM>,0]
  [cluster: test.e - ok, <E_CLUSTER>,0]
    [module: /home/stefan/localCVS/
      [#define - ok, <PREPROC_DEF
        [backtick - ok, <BACKTICK
        [`MDA - ok, <ID>,0]
        [ struct s {} - ok, <PREPRO
      [extend - ok, <EXTEND_STRU
        [sys - ok, <ID>,0]
        [struct_member_block - o
          [method_declaration -
```

Console:

```
Loading file : /home/stefan/localCVS/zrunner/tests/mda.z
Restore to core ...
Scan test.e ...
Load test.e ...
Load result :
***ZRunner: AST Comparision Status = PASSED
Context :
```

6.1 Menu & Toolbar

Menu & Toolbar contain basic commands like load, reload.

Picture 6-2 Toolbar

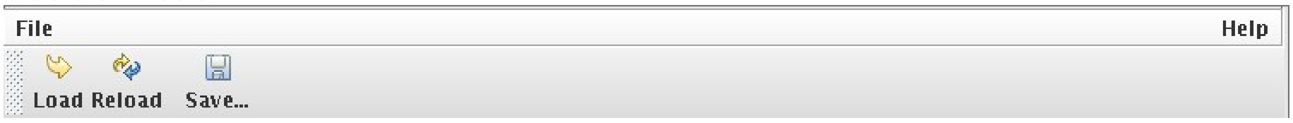


Table 6-1 Menu & Toolbar functions description

Function	Icon	Description
Load		Loads a source or test file. The difference is made by extension. (This function will split in two starting with 1.0 : load test & load source)
Reload		Reloads last loaded test/source file.
Save ...		Saves the test that is computed on the currently loaded file.
Help Contents	?	Fasper help.(Will be available starting with 1.0)
About	i	Some info panel.(Will be available starting with 1.0)

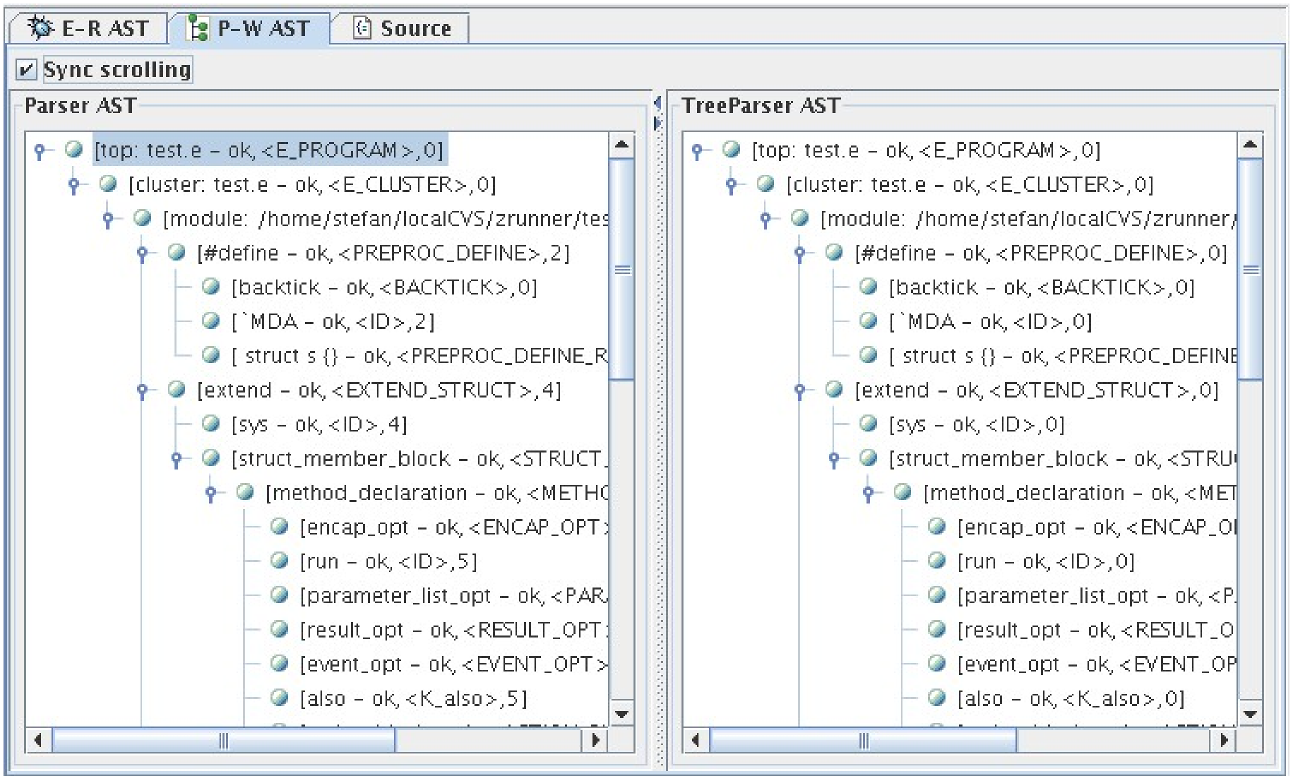
Note!

Help and About actions will be implemented starting with 1.0

6.2 Parser – TreeParser AST Tab

In this panel you can view the ASTs generated by the parser and tree parser.

Picture 6-3 Parser – TreeParser ASTs Tab



```
[top: test.e - ok, <E_PROGRAM>,0]
  [cluster: test.e - ok, <E_CLUSTER>,0]
    [module: /home/stefan/localCVS/zrunner/tes
      [#define - ok, <PREPROC_DEFINE>,2]
        [backtick - ok, <BACKTICK>,0]
        [`MDA - ok, <ID>,2]
        [ struct s {} - ok, <PREPROC_DEFINE_R
      [extend - ok, <EXTEND_STRUCT>,4]
        [sys - ok, <ID>,4]
        [struct_member_block - ok, <STRUCT_
          [method_declaration - ok, <METHO
            [encap_opt - ok, <ENCAP_OPT>
            [run - ok, <ID>,5]
            [parameter_list_opt - ok, <PAR
            [result_opt - ok, <RESULT_OPT>
            [event_opt - ok, <EVENT_OPT>
            [also - ok, <K_also>,5]
```

```
[top: test.e - ok, <E_PROGRAM>,0]
  [cluster: test.e - ok, <E_CLUSTER>,0]
    [module: /home/stefan/localCVS/zrunner/
      [#define - ok, <PREPROC_DEFINE>,0]
        [backtick - ok, <BACKTICK>,0]
        [`MDA - ok, <ID>,0]
        [ struct s {} - ok, <PREPROC_DEFINE
      [extend - ok, <EXTEND_STRUCT>,0]
        [sys - ok, <ID>,0]
        [struct_member_block - ok, <STRU
          [method_declaration - ok, <MET
            [encap_opt - ok, <ENCAP_OI
            [run - ok, <ID>,0]
            [parameter_list_opt - ok, <P
            [result_opt - ok, <RESULT_O
            [event_opt - ok, <EVENT_OP
            [also - ok, <K_also>,0]
```

Table 6-2 Parser-TreeParser Tab Functions

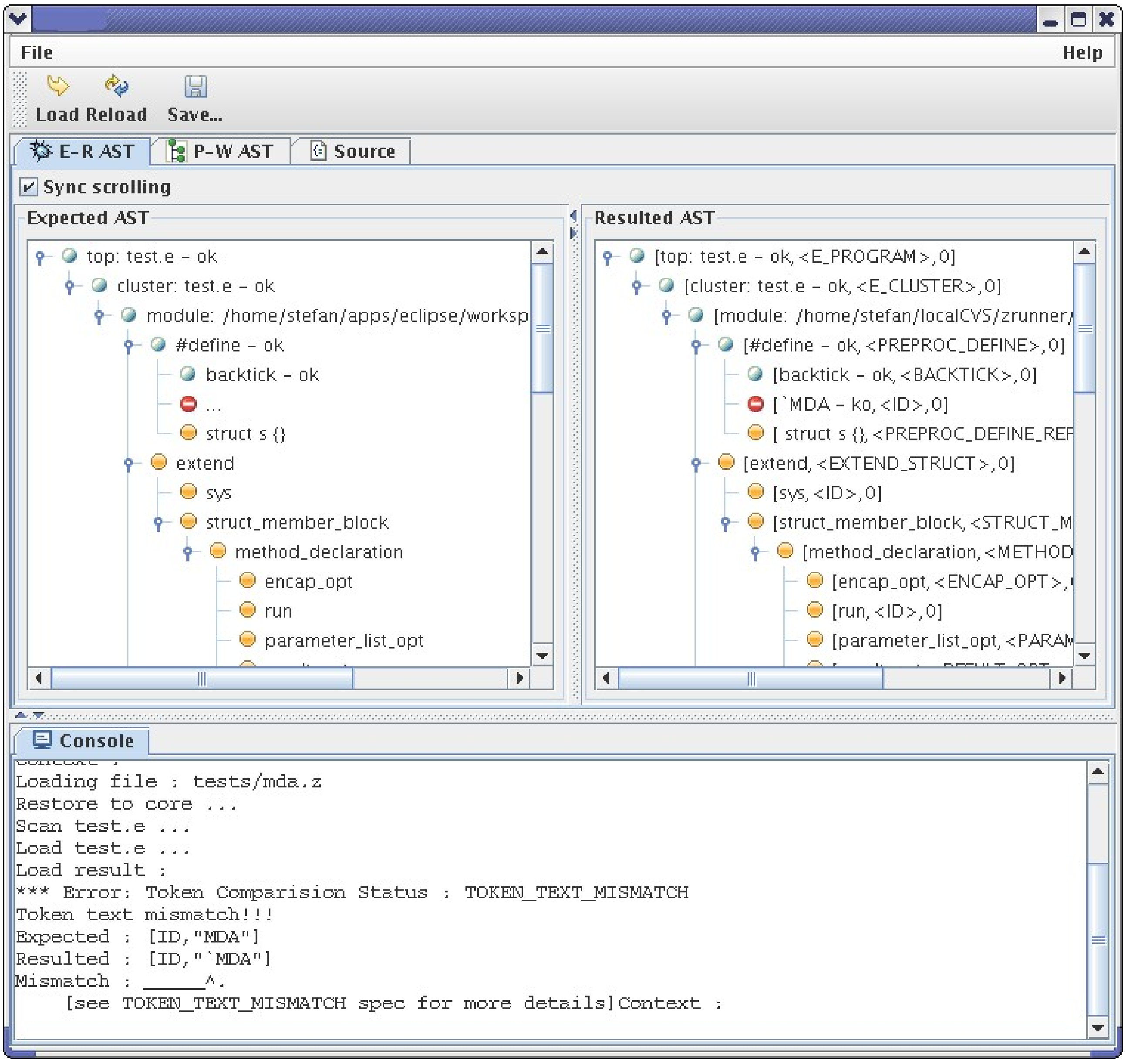
Function	Icon	Description
Synchronized	N.A.	Synchronizes left-right scrolling
Checked Node	●	The nodes have been compared and passed.
Not Checked Node	●	The nodes have not been compared.
Comparison Failed Node	●	The nodes have been compared and failed.

The Functions above are available for **Reference – Resulted AST Tab.**

6.3 Reference – Resulted AST Tab

This tab is very similar to the previous, only that reference and resulted ASTs are displayed and compared.

Picture 6-4 Reference – Resulted AST Tab



```
top: test.e - ok
  cluster: test.e - ok
    module: /home/stefan/apps/eclipse/worksp
      #define - ok
        backtick - ok
        ...
        struct s {}
      extend
        sys
        struct_member_block
          method_declaration
            encap_opt
            run
            parameter_list_opt
```

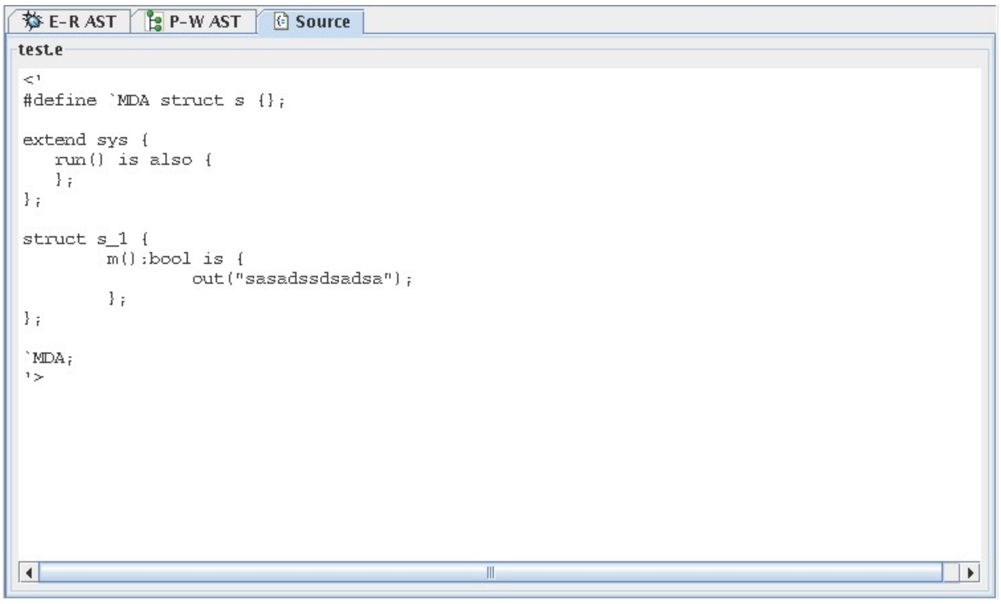
```
[top: test.e - ok, <E_PROGRAM>,0]
  [cluster: test.e - ok, <E_CLUSTER>,0]
    [module: /home/stefan/localCVS/zrunner/
      [#define - ok, <PREPROC_DEFINE>,0]
        [backtick - ok, <BACKTICK>,0]
        [`MDA - ko, <ID>,0]
        [ struct s {}, <PREPROC_DEFINE_REF
      [extend, <EXTEND_STRUCT>,0]
        [sys, <ID>,0]
        [struct_member_block, <STRUCT_M
          [method_declaration, <METHOD
            [encap_opt, <ENCAP_OPT>,
            [run, <ID>,0]
            [parameter_list_opt, <PARAM
```

```
Loading file : tests/mda.z
Restore to core ...
Scan test.e ...
Load test.e ...
Load result :
*** Error: Token Comparision Status : TOKEN_TEXT_MISMATCH
Token text mismatch!!!
Expected : [ID,"MDA"]
Resulted : [ID,"`MDA"]
Mismatch : ______^.
    [see TOKEN_TEXT_MISMATCH spec for more details]Context :
```

6.4 Source Tab

This tab shows the currently loaded file in read-only mode.

Picture 6-5 Source Tab

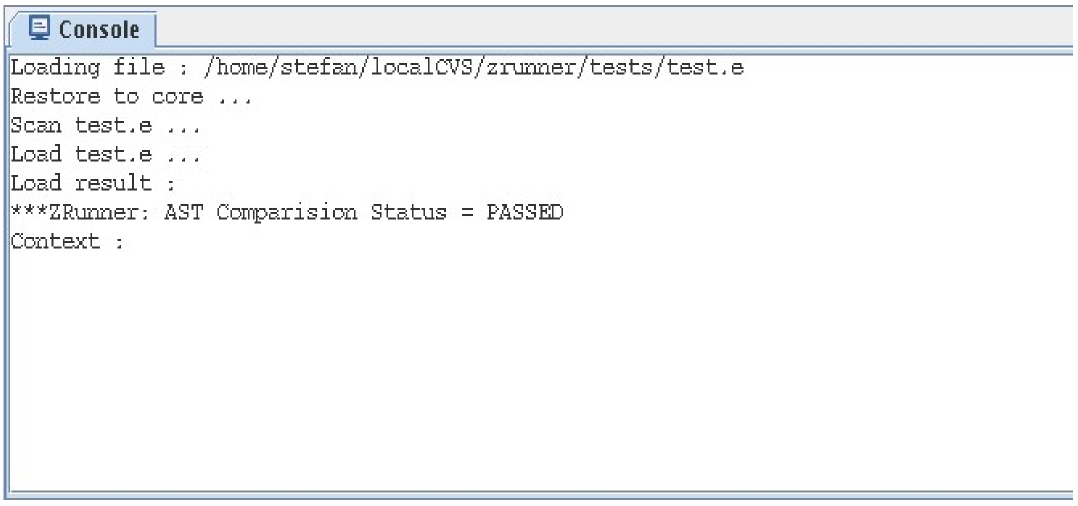


```
<'
#define `MDA struct s {};

extend sys {
   run() is also {
   };
};

struct s_1 {
        m():bool is {
                out("sasadssdsadsa");
        };
};

`MDA;
'>
```

6.5 Console

Console display all messages that go to System.out System.err streams.

Picture 6-6 Console Tab



```
Loading file : /home/stefan/localCVS/zrunner/tests/test.e
Restore to core ...
Scan test.e ...
Load test.e ...
Load result :
***ZRunner: AST Comparision Status = PASSED
Context :
```

Note!

The GUI is quite simple and some functionalities are not yet implemented. They will be available starting with 1.0.

7 Error Codes

Table 5-1 Error Codes Description

Error ID	Description
UNKNOWN_ERROR_CODE	The error code is not registered
FILE_IS_NULL	The file argument of `Tool.load()` is null.
UNABLE_TO_SAVE_TEST	Unable to save test.
NOT_USED	The comparator is not used.
PASSED	The test has passed.
EXPECTED_AST_IS_NULL	Expected AST is null. (To change in 1.0 to NULL_REFERENCE_AST)
RESULTED_AST_IS_NULL	Resulted AST is null. (To change in 1.0 to NULL_RESULTED_AST)
BOTH_ASTS_ARE_NULL	Both reference and resulted ASTs are null. (To change in 1.0 to COMPARE_NULL_ASTS)
EXPECTED_NODE_IS_NULL	Expected AST node is null. (To change in 1.0 to NULL_REFERENCE_AST_NODE)
RESULTED_NODE_IS_NULL	Resulted AST node is null. (To change in 1.0 to NULL_RESULTED_AST_NODE)
BOTH_NODES_ARE_NULL	Both reference and resulted AST nodes are null. (To change in 1.0 to COMPARE_NULL_AST_NODES)
TOKEN_TEXT_MISMATCH	Reference token and resulted token have a different text.
TOKEN_TYPE_MISMATCH	Reference token and resulted token have a different type.
DIFFERENT_AST_STRUCTURE	Resulted AST has a different structure than reference AST.
UNKNOWN_ACTIVATION_CLASS	Can not resolve the specified activation class (originates in a reflection error).
UNKNOWN_ACTIVATION_CONSTRUCTOR	Can not resolve the specified activation class constructor (originates in a reflection error).
ACTIVATION_INSTANCE_FAILED	Can not create an instance for the activation class.
UNKNOWN_ARGUMENT_CLASS	The class of the argument can not be resolved.
PARSER_ACTIVATION_FAILED	The activation of the parser failed.
WRONG_ACTIVATION_CLASS	Activation class does not implement one of the activation interfaces.
UNKNOWN_FILTER_CLASS	Unable to resolve filter class.
UNKNOWN_FILTER_CONSTRUCTOR	Unable to resolve filter class constructor.
FILTER_INSTANCE_FAILED	Unable to create a filter instance.

7 Error Codes

Error ID	Description
WRONG_FILTER_CLASS	Filter class does not implement the filter interface.
UNKNOWN_COMPARATOR_CLASS	Unable to resolve AST comparator class.
UNKNOWN_COMPARATOR_CONSTRUCTOR	Unable to resolve AST comparator class constructor.
COMPARATOR_INSTANCE_FAILED	Unable to create an AST comparator instance.
WRONG_COMPARATOR_CLASS	Comparator class does not implement AST comparator interface.

Note!

Exception codes will be stabilized in 1.0

8 antlrWorks Integration FAQ

8.1 Why should integrate fasper with antlrWorks?

Any development environment should offer :

- ✔ a fast way to verify the product under development (eg test generation)
- ✔ easy test/product debugging
- ✔ easy test management

8.2 Can we integrate fasper with the antlrWorks ?

Of course we can. All we have to do is to agree on a common interface that fasper developers should implement in order to accept commands from antlrWorks GUI. Another way would be to make a pluggable GUI component for fasper to integrate with antlrWorks.

8.3 What can antlrWroks & fasper offer by working together?

Here is a list of possible functionalities of a fasper/antlrWorks integration:

- ✔ automatic test generation
- ✔ test running & debugging
- ✔ automatic fasper plugin generation (java classes necessary to adapt parser under test to fasper framework)

8.4 What does test generation mean?

A test consists is made of two parts : test activation code and test data. Test activation code is the code that : creates new test object, loads test data, do some checking etc. Test data includes the source, parsed by the parser under test , and reference model, to which results of the source parsing are compared to. Test data can be provided by the test developer or by a test generator. The test generator can be similar to the one proposed by Terence Parr in thread : “generating random sentences”, in 05.02.2005.

8.5 What does plugin generation mean?

In order to make a test work with fasper framework, there should be some “glue” code (java/C++/C# classes), for example ParserActivation, ATSFilter etc. This code can be automatically generated using string templates, giving user the freedom of changing some of the parameters.

9 Known issues

There are a few known issues that will be solved in 1.0:

- ✗ the format of the test should be discussed
- ✗ difference between source files and test files: at this moment this difference is based on file's extension (quite primitive)
- ✗ the way the activation classes are specified: at this moment they are specified in a resource file : init.init or in z-test
- ✗ Defs singleton: it should be changed; if the two issues above are solved than most of this one is solved
- ✗ create plugin descriptor
- ✗ AST save should be done by a TestWriter, also pluggable
- ✗ create pluggable components only once (now every time load is called, new instances of plug-in components are made)
- ✗ string filters use
- ✗ a way to link to C++/Python/C# generated parsers
- ✗ move fasper.gm to fasper.defaults
- ✗ move DefaultASTComparator, DefaultTestWriter to fasper.default
- ✗ examples will be available starting with 1.0
- ✗ most of the issues & requirements will be fixed in accordance with the user feedback.

10 The People

Special thanks to :

- **Bogdan Mitu**
- **Cristian Amitroaie**
- **Stefan Birman**
- **Adrian Simionescu**
- **Terence Parr & ANTLR group members**
- **Users!**

We welcome anybody who can give a hand and/or give us feedback on quality, requirements etc

11 Resources

11.1 Java

- ✔ Home : http://java.sun.com
- ✔ Install notes : http://java.sun.com/j2se/1.5.0/jre/install.html
- ✔ Download : http://java.sun.com/j2se/1.5.0/download.jsp

11.2 Eclipse

- ✔ Home : http://eclipse.org

11.3 ANTLR

- ✔ Home : http://www.antlr.org
- ✔ Groups : http://groups.yahoo.com/group/antlr-interest/
- ✔ Groups : http://www.antlr.org:8080/mailman/listinfo/antlr-interest

11.4 Fasper

- ✔ Home : http://www.amiq.ro
- ✔ Install notes: http://www.amiq.ro/fasper_install.html
- ✔ Download: http://www.amiq.ro/fasper/download/fasper-1.0.zip

11.5 Contact

Stefan : birmanstefan@yahoo.com