Reusable Software
A Java Package

Trevor Nash

DR. JOSEF GROSCH

COCOLAB - DATENVERARBEITUNG

GERMANY

Cocktail

Toolbox for Compiler Construction

Reusable Software - A Java Package

Trevor Nash

January 11, 2000

Document No. 36

Dr. Josef Grosch
CoCoLab - Datenverarbeitung
Breslauer Str. 64c
76139 Karlsruhe
Germany

Phone: +49-721-91537544
Fax: +49-721-91537543
Email: grosch@cocolab.com

Abstract

A brief description of a useful package of reusable classes written in Java is given. The package is oriented towards compiler construction.

1. Overview

The most interesting classes are:

Class	Task
DynArray	dynamic and flexible arrays
Idents	identifier table - unambiguous encoding of strings
Sets	sets of scalar values (without run time checks)
Position	handling of source positions
Errors	error handler for parsers and compilers

Full details may be found in *doc.html/index.html.*

2. DynArray: dynamic and flexible arrays

Classes are provided for all the basic Java types providing dynamic and flexible arrays. The size of a dynamic array is determined at run time, and may be altered during its lifetime. The classes are designed for efficiency.

3. Idents: identifier table - unambiguous encoding of strings

The classes IdentTable and Ident are provided for the encoding of strings used as identifiers. Use of these classes provides an efficient way of comparing identifiers and of mapping them to associated information such as a symbol table.

4. Position: handling of source positions

A simple representation of the position of tokens in a source file consisting of fields for line and column. This class can be extended or copied and tailored to the user's needs, if necessary.

5. Errors: error handler for parsers and compilers

This module is needed by parsers generated with the parser generators *lark* or *ell*. It can also be used to report error messages found during scanning or semantic analysis.

This module can be regarded as a prototype for reporting compiler error messages. It can be copied and modified or even replaced in order to meet the requirements of the user's application. Three flags control the style of the error messages:

brief	summarize syntax errors in one error message instead of several messages
first	report only the first error message on a line instead of all messages
truncate	truncate additional information for messages (such as the set of expected symbols) to around 25 characters

Example: The following Pascal program contains two syntax errors:

```
program test (output);
begin
   if (a = b] write (a;
end.
```

If all three flags are set false then the following messages are reported:

```
3, 13: Error       syntax error
3, 13: Information token found    : ]
3, 13: Information expected tokens: ) = + - <> <= >= < > IN OR * / DIV MOD AND
3, 15: Information restart point
3, 15: Repair      token inserted : )
3, 15: Repair      token inserted : THEN
3, 23: Error       syntax error
3, 23: Information token found    : ;
3, 23: Information expected tokens: , ) = + - : <> <= >= < > IN OR * / DIV MOD AND
3, 23: Repair      token inserted : )
```

If brief is true then this is compressed into two lines:

```
3, 13: Error       found/expected : ]/) = + - <> <= >= < > IN OR * / DIV MOD AND
3, 23: Error       found/expected : ;/, ) = + - : <> <= >= < > IN OR * / DIV MOD AND
```

If brief and first are true then this results in just one line:

```
3, 13: Error       found/expected : ]/) = + - <> <= >= < > IN OR * / DIV MOD AND
```

If brief, first and truncate are all true (the default) then this one line becomes even shorter:

```
3, 13: Error       found/expected : ]/) = + - <> <= >= < > IN OR * / ...
```

In all of the abbreviated styles the information about restart points or inserted tokens is suppressed and the messages reporting the found token and the set of expected tokens are combined into one message.

6. General: miscellaneous functions

```
int  log2 (int x)
                        /* Returns the logarithm to the base 2 of ’x’.  */
int  Exp2 (unsigned long x);
                        /* Returns 2 to the power of ’x’.              */
```

Contents