STRUMENTA

RPG Parser

"

Why buy a commercial parser? In some cases there are open-source parsers available, so why consider buying a commercial parser?

Depending on your situation, you may need a parser that has been thoroughly tested, which is documented and having someone to call in case you encounter any problem. We create and distribute a commercial off-the-shelf RPG parser. A parser without any technical complexity and easy to integrate.

We create and distribute a commercial off-the-shelf RPG parser. You may want to buy a Commercial Parser to have a parser ready for the integration, without any technical complexity and easy to integrate. A ready to go RPG parser.

3 REASON TO BUY

PARSER BENCH & WARRANTY

With "Parser Bench"* you can try and discover all the features of the parser and then decide to buy it. Once bought it we offer a 1-year warranty extendable at your own convenience.

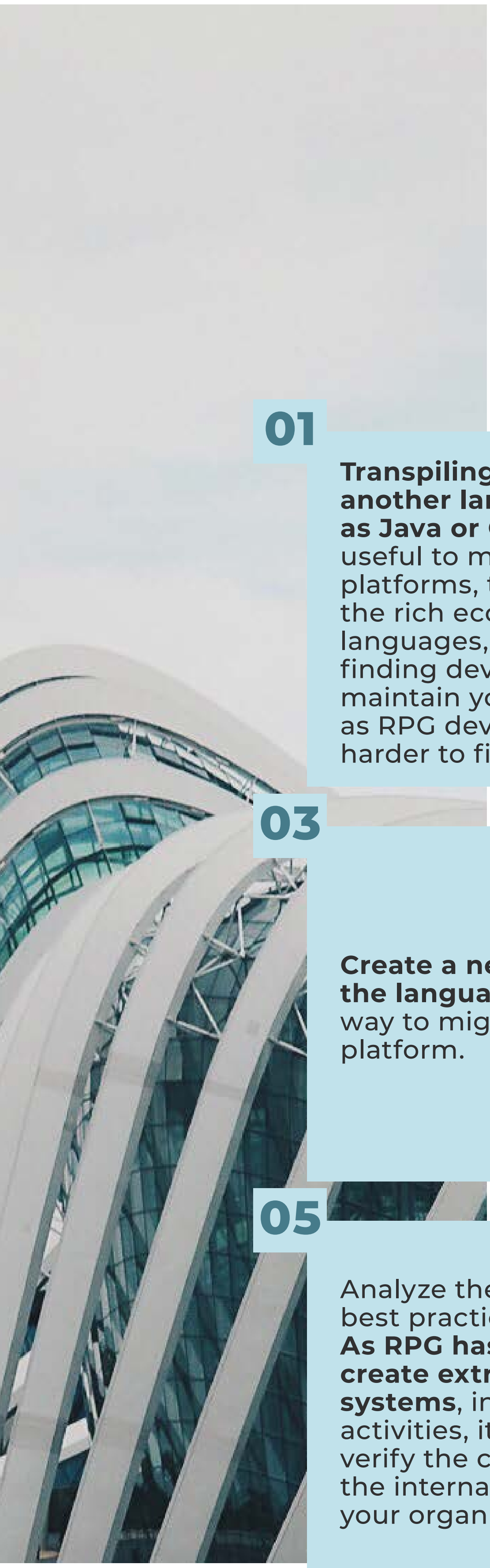
PERPETUAL LICENCE

We provide a perpetual, royalty-free license, permitting commercial usage and distribution, once bought you can use for as long as you want.

EASY INTEGRATION

All technical aspects are covered by us. We provide support, documentation and best practices.

*Parser Bench is an online tool to get demo for our parsers, you can find more information at page 8 of this brochure.



RPG is a language that has been used extensively for decades. Nowadays billions of lines composing extremely valuable software are defined in RPG.

There are several operations you may want to perform on your RPG code, wich our RPG parser might help you with.

LET’S DISCOVER THEM:

01

Transpiling that code to another language such as Java or C#. This may be useful to migrate to modern platforms, to benefit from the rich ecosystems of new languages, or to simplify finding developers to maintain your applications, as RPG developers become harder to find.

02

Write an interpreter for that language. This is an alternative to execute your RPG code on another platform. With this approach, we have also obtained the possibility of **combining Java and RPG code in a common system.**

03

Create a new compiler for the language, again another way to migrate to another platform.

04

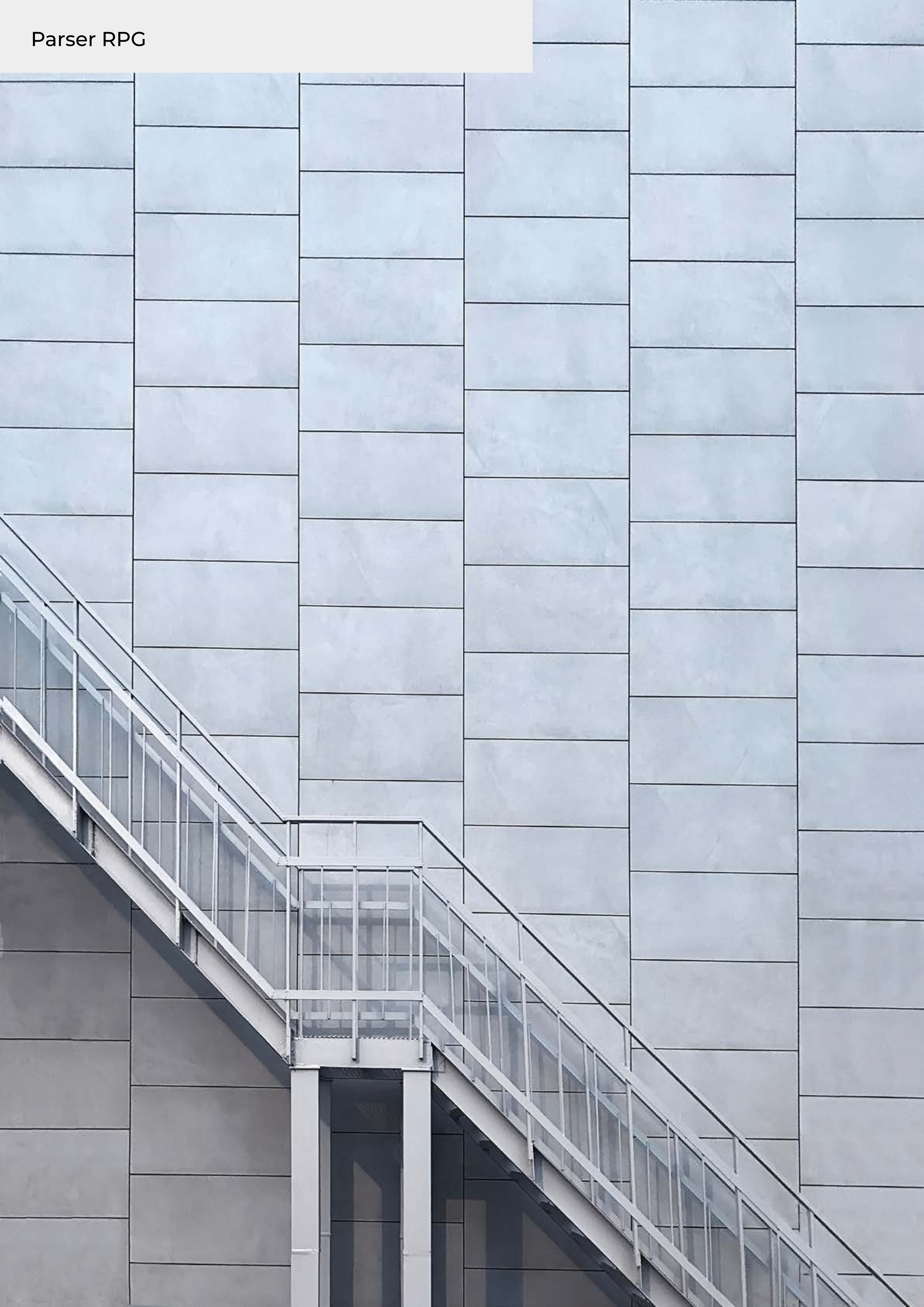
Build an editor with error checking and auto-completion. **This may be useful to provide better tools for your RPG developers.**

05

Analyze the code to ensure best practices are respected. **As RPG has been used to create extremely valuable systems**, in charge of core activities, it makes sense to verify the code respects all the internal rules defined in your organization.

06

Automatically refactor code. When dealing with a large codebase, there are occasions in which automatic transformations of code is the only way to perform changes at scale, **without the risk of incurring human errors.**



OVERVIEW OF THE RPG PARSER

The RPG Parser provided by Strumenta is a parser for the RPG language, i.e., it is a software component which process RPG code and return the information contained in it organized in an Abstract Syntax Tree (AST).

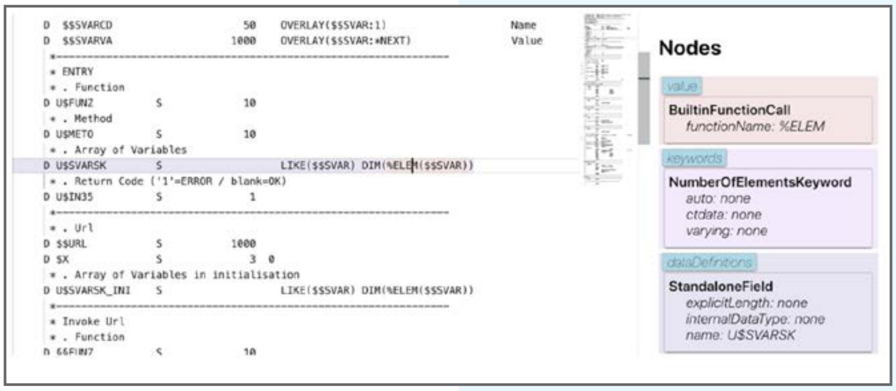
The parser can be used either as a library or a command-line tool.

- When it is used as a library it can be integrated into applications running on the Java Virtual Machine. For example, it can be invoked from Java, Scala, Kotlin, JRuby, and Groovy code. The library provides functionalities to navigate, modify, and transform the AST. Bindings for other languages could be built, if requested by clients.

- When the parser is used as a command-line tool, it can provide the AST as an output serialized in either JSON or XML format.

TECHNICAL SUPPORT

The parser supports RPGLE, both in the fixed-column and free format. It can also process SQLRPGLE files. It can process EXEC SQL statements: in that case, the SQL code is recognized but it is not parsed directly by the RPG Parser. **Strumenta can provide a separate SQL Parser** to be integrated for this goal. Below, we report some examples of files processed by the RPG Parser. The screenshots are obtained from a web-application named Parser Bench. **Parser Bench** has been developed by Strumenta to demonstrate the RPG Parser. It displays the AST as produced by the RPG Parser. Parser Bench itself is not part of the RPG Parser package. More information on Parser Bench are provided in a separate section.



```
D  $$SVARCD                  50    OVERLAY($$SVAR:1)                   Name
D  $$SVARVA                1000    OVERLAY($$SVAR:*NEXT)               Value
 *---------------------------------------------------------------
 * ENTRY
 * . Function
D U$FUNZ          S          10
 * . Method
D U$METO          S          10
 * . Array of Variables
D U$SVARSK        S                LIKE($$SVAR) DIM(%ELEM($$SVAR))
 * . Return Code ('1'=ERROR / blank=OK)
D U$IN35          S           1
 *---------------------------------------------------------------
 * . Url
D $$URL           S        1000
D $X              S           3 0
 * . Array of Variables in initialisation
D U$SVARSK_INI    S                LIKE($$SVAR) DIM(%ELEM($$SVAR))
 *---------------------------------------------------------------
 * Invoke Url
 * . Function
D $$FUNZ          S          10
```

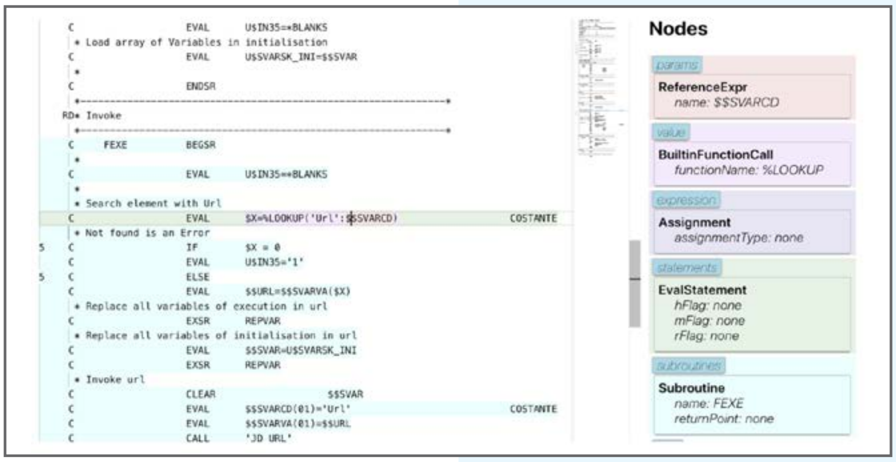
Nodes

value
BuiltinFunctionCall
functionName: %ELEM

keywords
NumberOfElementsKeyword
auto: none
ctdata: none
varying: none

dataDefinitions
StandaloneField
explicitLength: none
internalDataType: none
name: U$SVARSK

In the following example we can see a data definition being recognized and more in detail we can see an expression within a keyword usage being identified.



```
    C                   EVAL      U$IN35=*BLANKS
     * Load array of Variables in initialisation
    C                   EVAL      U$SVARSK_INI=$$SVAR
     *
    C                   ENDSR
     *-----------------------------------------------------------*
   RD* Invoke
     *-----------------------------------------------------------*
    C     FEXE          BEGSR
     *
    C                   EVAL      U$IN35=*BLANKS
     *
     * Search element with Url
    C                   EVAL      $X=%LOOKUP('Url':$$SVARCD)                     COSTANTE
     * Not found is an Error
5   C                   IF        $X = 0
    C                   EVAL      U$IN35='1'
5   C                   ELSE
    C                   EVAL      $$URL=$$SVARVA($X)
     * Replace all variables of execution in url
    C                   EXSR      REPVAR
     * Replace all variables of initialisation in url
    C                   EVAL      $$SVAR=U$SVARSK_INI
    C                   EXSR      REPVAR
     * Invoke url
    C                   CLEAR                   $$SVAR
    C                   EVAL      $$SVARCD(01)='Url'                             COSTANTE
    C                   EVAL      $$SVARVA(01)=$$URL
    C                   CALL      'JD URL'
```

Nodes

params
ReferenceExpr
name: $$SVARCD

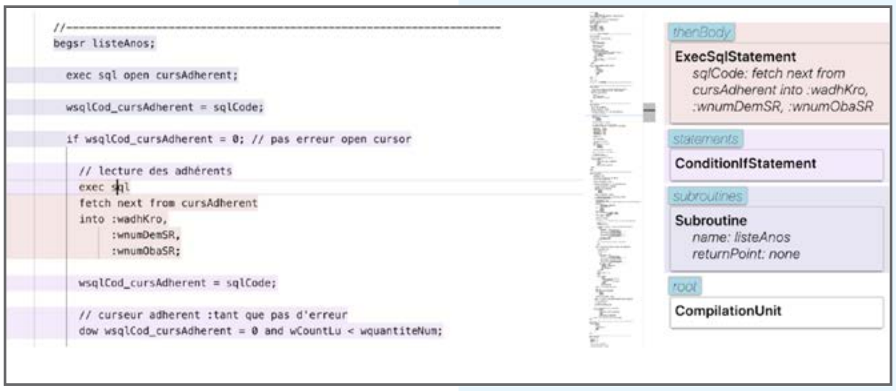
value
BuiltinFunctionCall
functionName: %LOOKUP

expression
Assignment
assignmentType: none

statements
EvalStatement
hFlag: none
mFlag: none
rFlag: none

subroutines
Subroutine
name: FEXE
returnPoint: none

In the following example we can see a reference expression, within the expression, within the invocation of Builtin Function, within an EVAL statement.



```
//---------------------------------------------------------------------
begsr listeAnos;

  exec sql open cursAdherent;

  wsqlCod_cursAdherent = sqlCode;

  if wsqlCod_cursAdherent = 0; // pas erreur open cursor

    // lecture des adhérents
    exec sql
    fetch next from cursAdherent
    into :wadhKro,
         :wnumDemSR,
         :wnumObaSR;

    wsqlCod_cursAdherent = sqlCode;

    // curseur adherent :tant que pas d'erreur
    dow wsqlCod_cursAdherent = 0 and wCountLu < wquantiteNum;
```

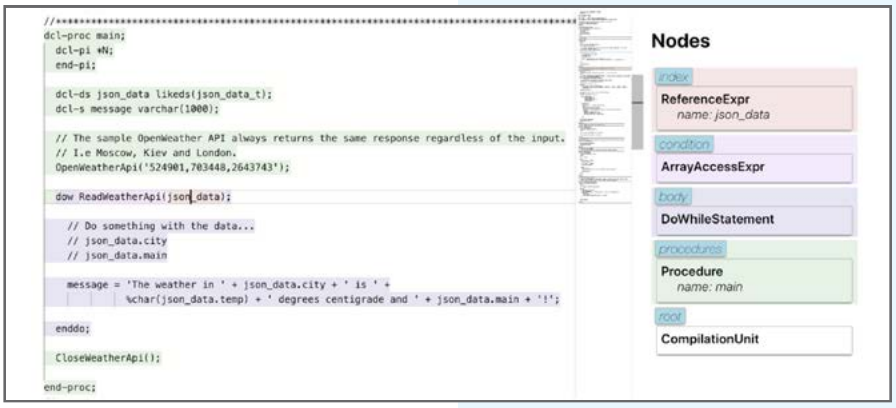
thenBody
ExecSqlStatement
sqlCode: fetch next from cursAdherent into :wadhKro, :wnumDemSR, :wnumObaSR

statements
ConditionIfStatement

subroutines
Subroutine
name: listeAnos
returnPoint: none

root
CompilationUnit

In the following example, we can an EXEC SQL statement being recognized. The SQL code is identified but it is not processed to build a corresponding AST for the SQL statement.
I.e., the RPG Parser does not include an SQL Parser. It could be combined with an SQL Parser that is provided by Strumenta.



```
//*************************************************************************************
dcl-proc main;
  dcl-pi *N;
  end-pi;

  dcl-ds json_data likeds(json_data_t);
  dcl-s message varchar(1000);

  // The sample OpenWeather API always returns the same response regardless of the input.
  // I.e Moscow, Kiev and London.
  OpenWeatherApi('524901,703448,2643743');

  dow ReadWeatherApi(json_data);

    // Do something with the data...
    // json_data.city
    // json_data.main

    message = 'The weather in ' + json_data.city + ' is ' +
              %char(json_data.temp) + ' degrees centigrade and ' + json_data.main + '!';

  enddo;

  CloseWeatherApi();

end-proc;
```

Nodes

index
ReferenceExpr
name: json_data

condition
ArrayAccessExpr

body
DoWhileStatement

procedures
Procedure
name: main

root
CompilationUnit

In the following example, we can see the RPG Parser processing RPG code in the free format.

PARSER BENCH

The RPG Parser's main goal is to analyze source code, producing an AST as a result, and a list of errors identified in the code. It is difficult for clients to analyze the AST produced when the ASTs are in memory or even when they are serialized as JSON or XML. For this reason, **we created a web application to visualize the ASTs produced by the parser** on a set of examples. This should permit to get an idea of the capabilities of the RPG Parser before deciding to adopt it.

The application is called Parser Bench and can be found here: PARSER BENCH STRUMENTA. Any user can register or log in with Google Authentication to see the public examples provided for the RPG parser and for other parsers we developed.

Examples provided for single clients can be loaded into Parser Bench and be made accessible only to users associated with those clients. In case a private installation of Parser Bench is requested, it could be provided.

DESIGN OF THE RPG PARSER

The design of the RPG Parser is based on the approach developed at Strumenta, based on the experience collected working on multiple parsers over the years.

The architecture is described has been presented in a webinar which is available here:

Strumenta can provide support to perform all the activities related to the RPG parser. Support can include training, architectural design, implementation, or coaching.

The goal of a parser is to provide a convenient way to work with the information contained in the code.

This requires two things:

1) Recognize the information contained in the code and organize them appropriately in the AST

2) Provide a convenient API to work with the AST. To satisfy this second requirement, the RPG Parser is built upon the Kolasu open-source library.

To satisfy this second requirement, the RPG Parser is built upon the Kolasu open-source parsers.

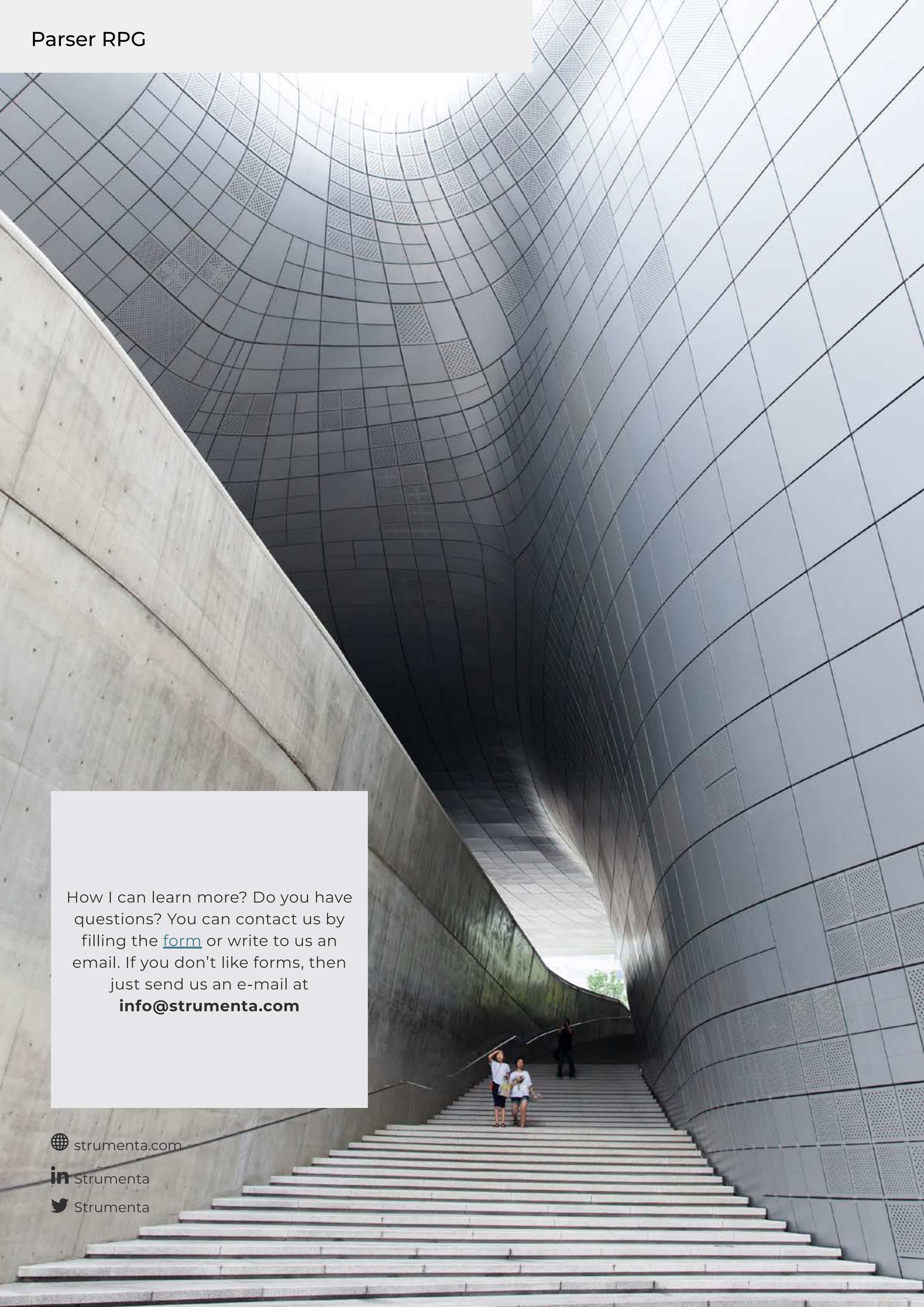
To get an overview of the benefits of the Kolasu parser you can read to this article here: [Building advanced parsers using Kolasu]()

LICENCE

The standard license includes the right to redistribute the parser, compiled, as part of larger software developed by you.

It can also include a support period if agreed. During that period each problem reported will be fixed for free by Strumenta. After that optional extended guarantee can be acquired.

The full text of the license is provided in a separate document.



How I can learn more? Do you have questions? You can contact us by filling the form or write to us an email. If you don't like forms, then just send us an e-mail at **info@strumenta.com**

strumenta.com

Strumenta

Strumenta