PyFranca Documentation

Release 0.4.1

Kaloyan Tenchov

Oct 06, 2017

Contents

Contents:

CHAPTER 1

pyfranca package

pyfranca.franca_processor module

class pyfranca.franca_processor.**Processor**
Bases: object

Franca IDL processor.

static basename (*namespace*)
Extract the type or namespace name from a Franca FQN.

import_file (*fspec*, *references=None*, *package_path=None*)
Parse an FIDL file and import it into the processor as package.

Parameters

- **fspec** – File specification.
- **references** – A list of package references.
- **package_path** – Additional model path to search for imports.

Returns The parsed ast.Package.

import_package (*fspec*, *package*, *references=None*)
Import an ast.Package into the processor.

Parameters

- **fspec** – File specification of the package.
- **package** – ast.Package object.
- **references** – A list of package references.

import_string (*fspec*, *fidl*, *references=None*)
Parse an FIDL string and import it into the processor as package.

Parameters

- **fspec** – File specification of the package.
- **fidl** – FIDL string.
- **references** – A list of package references.

Returns The parsed ast.Package.

static is_fqn(*string*)
Defines whether a Franca name is an ID or an FQN.

static packagename(*namespace*)
Extract the package name from a Franca FQN.

static resolve(*namespace*, *fqn*)
Resolve type references.

Parameters

- **namespace** – context ast.Namespace object.
- **fqn** – FQN or ID string.

Returns Dereferenced ast.Type object.

static resolve_namespace(*package*, *fqn*)
Resolve namespace references.

Parameters

- **package** – context ast.Package object.
- **fqn** – FQN or ID string.

Returns Dereferenced ast.Namespace object.

static split_fqn(*fqn*)
Split a Franca FQN into a tuple - package, namespace, and name.

exception pyfranca.franca_processor.**ProcessorException**(*message*)
Bases: exceptions.Exception

pyfranca.ast module

Franca abstract syntax tree representation.

exception pyfranca.ast.**ASTException**(*message*)
Bases: exceptions.Exception

class pyfranca.ast.**Argument**(*name*, *arg_type*, *comments=None*)
Bases: object

class pyfranca.ast.**Array**(*name*, *element_type*, *comments=None*)
Bases: *pyfranca.ast.ComplexType*

class pyfranca.ast.**Attribute**(*name*, *attr_type*, *flags=None*, *comments=None*)
Bases: *pyfranca.ast.Type*

class pyfranca.ast.**Boolean**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**BooleanValue**(*value*)
Bases: *pyfranca.ast.Value*

class pyfranca.ast.**Broadcast**(*name*, *flags=None*, *out_args=None*, *comments=None*)
Bases: *pyfranca.ast.Type*

class pyfranca.ast.**ByteBuffer**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**ComplexType**(*comments=None*)
Bases: *pyfranca.ast.Type*

class pyfranca.ast.**Constant**(*name*, *element_type*, *element_value*, *comments=None*)
Bases: *pyfranca.ast.ComplexType*

class pyfranca.ast.**Double**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**DoubleValue**(*value*)
Bases: *pyfranca.ast.Value*

class pyfranca.ast.**Enumeration**(*name*, *enumerators=None*, *extends=None*, *flags=None*, *comments=None*)
Bases: *pyfranca.ast.ComplexType*

class pyfranca.ast.**Enumerator**(*name*, *value=None*, *comments=None*)
Bases: object

class pyfranca.ast.**Float**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**FloatValue**(*value*)
Bases: *pyfranca.ast.Value*

class pyfranca.ast.**Import**(*file_name*, *namespace=None*)
Bases: object

class pyfranca.ast.**Int16**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**Int32**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**Int64**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**Int8**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**IntegerValue**(*value*, *base=10*)
Bases: *pyfranca.ast.Value*

BINARY = 2

DECIMAL = 10

HEXADECIMAL = 16

class pyfranca.ast.**Interface**(*name*, *flags=None*, *members=None*, *extends=None*, *comments=None*)
Bases: *pyfranca.ast.Namespace*

class pyfranca.ast.**Map**(*name*, *key_type*, *value_type*, *comments=None*)
Bases: *pyfranca.ast.ComplexType*

class pyfranca.ast.**Method**(*name*, *flags=None*, *in_args=None*, *out_args=None*, *errors=None*, *comments=None*)
Bases: *pyfranca.ast.Type*

class pyfranca.ast.**Namespace**(*name*, *flags=None*, *members=None*, *comments=None*)
Bases: object

class pyfranca.ast.**Package**(*name*, *file_name=None*, *imports=None*, *interfaces=None*, *typecollections=None*, *comments=None*)
Bases: object

AST representation of a Franca package.

class pyfranca.ast.**PrimitiveType**
Bases: *pyfranca.ast.Type*

class pyfranca.ast.**Reference**(*name*)
Bases: *pyfranca.ast.Type*

class pyfranca.ast.**String**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**StringValue**(*value*)
Bases: *pyfranca.ast.Value*

class pyfranca.ast.**Struct**(*name*, *fields=None*, *extends=None*, *flags=None*, *comments=None*)
Bases: *pyfranca.ast.ComplexType*

class pyfranca.ast.**StructField**(*name*, *field_type*, *comments=None*)
Bases: object

class pyfranca.ast.**Type**(*name=None*, *comments=None*)
Bases: object

class pyfranca.ast.**TypeCollection**(*name*, *flags=None*, *members=None*, *comments=None*)
Bases: *pyfranca.ast.Namespace*

class pyfranca.ast.**Typedef**(*name*, *base_type*, *comments=None*)
Bases: *pyfranca.ast.Type*

class pyfranca.ast.**UInt16**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**UInt32**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**UInt64**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**UInt8**
Bases: *pyfranca.ast.PrimitiveType*

class pyfranca.ast.**Value**(*value*, *value_type=None*)
Bases: *pyfranca.ast.Type*

class pyfranca.ast.**Version**(*major*, *minor*)
Bases: object

pyfranca.franca_parser module

Franca parser.

class pyfranca.franca_parser.**ArgumentGroup**(*arguments=None*)
Bases: object

class pyfranca.franca_parser.**ErrorArgumentGroup**(*arguments=None*)
Bases: *pyfranca.franca_parser.ArgumentGroup*

class pyfranca.franca_parser.**InArgumentGroup**(*arguments=None*)
Bases: *pyfranca.franca_parser.ArgumentGroup*

class pyfranca.franca_parser.**OutArgumentGroup**(*arguments=None*)
Bases: *pyfranca.franca_parser.ArgumentGroup*

class pyfranca.franca_parser.**Parser**(*the_lexer=None*, ***kwargs*)
Bases: object

Franca IDL PLY parser.

static p_arg_def(*p*)
arg_def : structured_comment type ID

static p_arg_defs_1(*p*)
arg_defs : arg_defs arg_def

static p_arg_defs_2(*p*)
arg_defs : arg_def

static p_arg_group_def_1(*p*)
arg_group_def : IN '{' arg_defs '}'

static p_arg_group_def_2(*p*)
arg_group_def : OUT '{' arg_defs '}'

static p_arg_group_def_3(*p*)
arg_group_def : ERROR '{' enumerators '}'

static p_arg_group_def_4(*p*)
arg_group_def : ERROR type

static p_arg_group_defs_1(*p*)
arg_group_defs : arg_group_defs arg_group_def

static p_arg_group_defs_2(*p*)
arg_group_defs : arg_group_def

static p_arg_group_defs_3(*p*)
arg_group_defs : empty

static p_array_def(*p*)
array_def : structured_comment ARRAY ID OF type

static p_attribute_def(*p*)
attribute_def : structured_comment ATTRIBUTE type ID flag_defs

static p_boolean_val(*p*)
boolean_val : BOOLEAN_VAL

static p_broadcast_def(*p*)
broadcast_def : structured_comment BROADCAST ID flag_defs '{' arg_group_defs '}'

static p_constant_def_1(*p*)

constant_def [structured_comment CONST INT8 ID '=' integer_val]

structured_comment CONST INT16 ID '=' integer_val
structured_comment CONST INT32 ID '=' integer_val
structured_comment CONST INT64 ID '=' integer_val
structured_comment CONST UINT8 ID '=' integer_val

structured_comment CONST UINT16 ID '=' integer_val
structured_comment CONST UINT32 ID '=' integer_val
structured_comment CONST UINT64 ID '=' integer_val

static **p_constant_def_2** (*p*)

constant_def [structured_comment CONST INT8 ID '=' boolean_val]

structured_comment CONST INT16 ID '=' boolean_val
structured_comment CONST INT32 ID '=' boolean_val
structured_comment CONST INT64 ID '=' boolean_val
structured_comment CONST UINT8 ID '=' boolean_val
structured_comment CONST UINT16 ID '=' boolean_val
structured_comment CONST UINT32 ID '=' boolean_val
structured_comment CONST UINT64 ID '=' boolean_val
structured_comment CONST INT8 ID '=' real_val
structured_comment CONST INT16 ID '=' real_val
structured_comment CONST INT32 ID '=' real_val
structured_comment CONST INT64 ID '=' real_val
structured_comment CONST UINT8 ID '=' real_val
structured_comment CONST UINT16 ID '=' real_val
structured_comment CONST UINT32 ID '=' real_val
structured_comment CONST UINT64 ID '=' real_val

static **p_constant_def_3** (*p*)

constant_def [structured_comment CONST FLOAT ID '=' integer_val]

structured_comment CONST FLOAT ID '=' boolean_val
structured_comment CONST FLOAT ID '=' real_val

static **p_constant_def_4** (*p*)

constant_def [structured_comment CONST DOUBLE ID '=' integer_val]

structured_comment CONST DOUBLE ID '=' boolean_val
structured_comment CONST DOUBLE ID '=' real_val

static **p_constant_def_5** (*p*)
constant_def : structured_comment CONST BOOLEAN ID '=' value

static **p_constant_def_6** (*p*)
constant_def : structured_comment CONST STRING ID '=' value

static **p_defs_1** (*p*)
defs : defs def

static **p_defs_2** (*p*)
defs : def

static **p_defs_3** (*p*)
defs : empty

static **p_empty** (*p*)
empty :

static **p_enumeration_def_1** (*p*)
enumeration_def : structured_comment ENUMERATION ID '{' enumerators '}'

static **p_enumeration_def_2** (*p*)
enumeration_def : structured_comment ENUMERATION ID EXTENDS fqn '{' enumerators '}'

static **p_enumerator_1** (*p*)
enumerator : structured_comment ID

static **p_enumerator_2** (*p*)
enumerator : structured_comment ID '=' integer_val

static **p_enumerators_1** (*p*)
enumerators : enumerators enumerator

static **p_enumerators_2** (*p*)
enumerators : enumerator

static **p_enumerators_3** (*p*)
enumerators : empty

static **p_error** (*p*)

static **p_flag_def** (*p*)

flag_def [SELECTIVE]

FIREANDFORGET
POLYMORPHIC
NOSUBSCRIPTIONS
READONLY

static **p_flag_defs_1** (*p*)
flag_defs : flag_defs flag_def

static **p_flag_defs_2** (*p*)
flag_defs : flag_def

static **p_flag_defs_3** (*p*)
flag_defs : empty

static **p_fqn_1** (*p*)
fqn : ID '.' fqn

static **p_fqn_2** (*p*)
fqn : ID

static **p_fqn_3** (*p*)
fqn : '*'

static **p_import_def_1** (*p*)
def : IMPORT fqn FROM STRING_VAL

static **p_import_def_2** (*p*)
def : IMPORT MODEL STRING_VAL

static **p_integer_val** (*p*)
integer_val : INTEGER_VAL

static **p_integer_val_2** (*p*)
integer_val : HEXADECIMAL_VAL

static **p_integer_val_3** (*p*)
integer_val : BINARY_VAL

static **p_interface_1** (*p*)
def : structured_comment INTERFACE ID '{' interface_members '}'

static **p_interface_2** (*p*)
def : structured_comment INTERFACE ID EXTENDS fqn '{' interface_members '}'

static **p_interface_member** (*p*)

interface_member [version_def]

attribute_def
method_def
broadcast_def
type_def
enumeration_def
struct_def
array_def
map_def
constant_def

static **p_interface_members_1** (*p*)
interface_members : interface_members interface_member

static **p_interface_members_2** (*p*)
interface_members : interface_member

static **p_interface_members_3** (*p*)
interface_members : empty

static **p_map_def** (*p*)
map_def : structured_comment MAP ID '{' type TO type '}'

static **p_method_def** (*p*)
method_def : structured_comment METHOD ID flag_defs '{' arg_group_defs '}'

static **p_package_def** (*p*)
package_def : structured_comment PACKAGE fqn defs

static **p_real_val** (*p*)
real_val : REAL_VAL

static **p_string_val** (*p*)
string_val : STRING_VAL

static **p_struct_def_1** (*p*)
struct_def : structured_comment STRUCT ID flag_defs '{' struct_fields '}'

static **p_struct_def_2** (*p*)
struct_def : structured_comment STRUCT ID EXTENDS fqn '{' struct_fields '}'

static **p_struct_field** (*p*)
struct_field : structured_comment type ID

static **p_struct_fields_1** (*p*)
struct_fields : struct_fields struct_field

static **p_struct_fields_2** (*p*)
struct_fields : struct_field

static **p_struct_fields_3** (*p*)
struct_fields : empty

static **p_structured_comment_1** (*p*)
structured_comment : STRUCTURED_COMMENT

static **p_structured_comment_2** (*p*)
structured_comment : empty

static **p_type_1** (*p*)

type [INT8]

INT16
INT32
INT64
UINT8
UINT16
UINT32
UINT64
BOOLEAN
FLOAT
DOUBLE
STRING
BYTEBUFFER

static **p_type_2** (*p*)

type [INT8 '[' ']']

INT16 '[' ']'
INT32 '[' ']'
INT64 '[' ']'
UINT8 '[' ']'
UINT16 '[' ']'
UINT32 '[' ']'
UINT64 '[' ']'
BOOLEAN '[' ']'
FLOAT '[' ']'
DOUBLE '[' ']'
STRING '[' ']'
BYTEBUFFER '[' ']'

static **p_type_3** (*p*)
type : fqn

static **p_type_4** (*p*)
type : fqn '[' ']'

static **p_type_def** (*p*)
type_def : structured_comment TYPEDEF ID IS type

static **p_typecollection** (*p*)
def : structured_comment TYPECOLLECTION ID '{' typecollection_members '}'

static **p_typecollection_member** (*p*)

typecollection_member [version_def]

type_def
enumeration_def
struct_def
array_def

map_def
constant_def

static **p_typecollection_members_1** (*p*)
typecollection_members : typecollection_members typecollection_member

static **p_typecollection_members_2** (*p*)
typecollection_members : typecollection_member

static **p_typecollection_members_3** (*p*)
typecollection_members : empty

static **p_value** (*p*)

value [boolean_val]

string_val
real_val
integer_val

static **p_version_def** (*p*)
version_def : VERSION '{' MAJOR INTEGER_VAL MINOR INTEGER_VAL '}'

parse (*fidl*)
Parse input text

Parameters **fidl** – Input text to parse.

Returns AST representation of the input.

parse_file (*fspec*)
Parse input file

Parameters **fspec** – Specification of a fidl to parse.

Returns AST representation of the input.

static **parse_structured_comment** (*comment*)
Parse a structured comment.

Parameters **comment** – Structured comment of an Franca-IDL symbol to parse.

Returns OrderedDict of all comments. Key is Franca-IDL keyword, e.g. @description, value conatins the text.

exception pyfranca.franca_parser.**ParserException** (*message*)
Bases: exceptions.Exception

pyfranca.franca_lexer module

Franca lexer.

class pyfranca.franca_lexer.**Lexer** (***kwargs*)
Bases: object

Franca IDL PLY lexer.

keyword = 'ByteBuffer'

keywords = ['package', 'import', 'from', 'model', 'typeCollection', 'version', 'major', 'minor', 'typedef', 'is', 'interface'

literals = ['.', '{', '}', '*', '=', '[', ']']

static t_BINARY_VAL(*t*)
0[bB][01]+

static t_BLOCK_COMMENT(*t*)
/*(.|n)*?*/

static t_BOOLEAN_VAL(*t*)
(true|false)

static t_HEXADECIMAL_VAL(*t*)
0[xX][0-9a-fA-F]+

static t_ID(*t*)
[A-Za-z][A-Za-z0-9_]*

static t_INTEGER_VAL(*t*)
[+-]?d+

static t_LINE_COMMENT(*t*)
//[^rn]*

static t_NEWLINE(*t*)
n+

static t_REAL_VAL(*t*)
[+-]?(((((([0-9]*.[0-9]+)|([0-9]+.))([eE][-+]?[0-9]+)?)|([0-9]+([eE][-+]?[0-9]+)))[fFdD]?)

static t_STRING_VAL(*t*)
“[^”]*”

static t_STRUCTURED_COMMENT(*t*)
<**(.|n)*?**>

static t_error(*t*)

t_ignore = ‘ \t’

tokenize(*data*)
Tokenize input data to stdout for testing purposes.

Parameters **data** – Input text to parse.

tokenize_data(*data*)
Tokenize input data to stdout for testing purposes.

Parameters **data** – Input text to parse.

tokenize_file(*fspec*)
Tokenize input file to stdout for testing purposes.

Parameters **fspec** – Input file to parse.

tokens = [’PACKAGE’, ‘IMPORT’, ‘FROM’, ‘MODEL’, ‘TYPECOLLECTION’, ‘VERSION’, ‘MAJOR’, ‘MINOR’, ‘

exception pyfranca.franca_lexer.**LexerException**(*message*)
Bases: exceptions.Exception

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

R

S

T

U

V