jaula API Reference Reference Manual

version 1.2.0

Generated by Doxygen 1.5.1

Sat May 12 10:39:21 2007

Contents

Chapter 1

jaula API Reference Main Page

1.1 Introduction

JAULA means "JSON Analysis User Library Acronym" and is the name for a C++ library that parses and writes data in JSON format.

This library uses standard C++ streams for reading the data to parse and for writing output data so, it can be easily used not only for reading and writing JSON files as for serializing data throw sockets or other means, making it suitable (for instance) for JSON RPC.

Project is being hosted by SourceForge at http://sourceforge.net/projects/jaula

Further documentation can be found at project web site http://morongo.homelinux.net/jaula

1.1.1 Copyright

Author:

(c) 2007 Kombo Morongo <morongo@users.sourceforge.net>

1.2 GNU Free Documentation License

GNU Free Documentation License Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author

and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the

name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of

following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Chapter 2

jaula API Reference Module Index

2.1 jaula API Reference Modules

Here is a list of all modules:

Chapter 3

jaula API Reference Namespace Index

3.1 jaula API Reference Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 4

jaula API Reference Hierarchical Index

4.1 jaula API Reference Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

jaula API Reference Data Structure Index

5.1 jaula API Reference Data Structures

Here are the data structures with brief descriptions:

Chapter 6

jaula API Reference Module Documentation

6.1 JAULA: General definitions

Namespaces

- namespace JAULA

  Namespace for all library definitions.

6.2 JAULA: Error handling

Data Structures

- class JAULA::Bad_Data_Type
  class for incompatible data type exceptions
- class JAULA::Exception
  Base class for error handling exceptions.
- class JAULA::Lexan_Error
  class for lexical analysis exceptions
- class JAULA::Name_Duplicated
  Class for bad object property name exceptions.
- class JAULA::No_Error
  class for no error condition exceptions
- class JAULA::Syntax_Error
  class for syntax exceptions

Functions

- std::ostream & operator<< (std::ostream &ostr, JAULA::Exception const &ex)
  Insertion operator extension for Exceptions.

6.2.1 Function Documentation

6.2.1.1 std::ostream& operator<< (std::ostream & *ostr*, JAULA::Exception const & *ex*)

Insertion operator extension for Exceptions.

Parameters:

ostr Stream where the instance is to be displayed.

ex Instance to display

Returns:

a reference to the stream

Description

This method extends the standard insertion operation for streams to invoke JAULA::display() through this alternative sintax.

Author:

Kombo Morongo <morongo666@gmail.com>

6.3 JAULA: JSON lexical analysis

Data Structures

- class JAULA::Lexan

  Lexical Analysis implementation.

6.4 JAULA: JSON data parser

Data Structures

- class JAULA::Parser

  JSON Data Parser.

- class JAULA::Parser::Value_Parser

  JSON Value Parser.

6.5 JAULA: JSON Values containers

Data Structures

- class JAULA::Value
  Base class for handling values.
- class JAULA::Value_Array
  Class for handling array values.
- class JAULA::Value_Boolean
  Class for handling boolean values.
- class JAULA::Value_Complex
  Base class for handling complex values.
- class JAULA::Value_Null
  class for handling null values
- class JAULA::Value_Number
  Class for handling numeric values.
- class JAULA::Value_Number_Int
  Class for handling numeric (int) values.
- class JAULA::Value_Object
  Class for handling object values.
- class JAULA::Value_String
  Class for handling numeric values.

Functions

- std::ostream & operator<< (std::ostream &ostr, JAULA::Value const &val)
  Insertion operator extension for values.

6.5.1 Function Documentation

6.5.1.1 std::ostream& operator<< (std::ostream & *ostr*, JAULA::Value const & *val*)

Insertion operator extension for values.

Parameters:

ostr Stream where the instance is to be represented.

val Instance to represent

Returns:

a reference to the stream

Description

This method extends the standard insertion operation for streams to invoke JAULA::repr() through this alternative sintax wirting the contents in JSON notation.

Author:

Kombo Morongo <morongo666@gmail.com>

Chapter 7

jaula API Reference Namespace Documentation

7.1 JAULA Namespace Reference

Namespace for all library definitions.

Data Structures

- class Bad_Data_Type

  class for incompatible data type exceptions

- class Exception

  Base class for error handling exceptions.

- class Lexan

  Lexical Analysis implementation.

- class Lexan_Error

  class for lexical analysis exceptions

- class Name_Duplicated

  Class for bad object property name exceptions.

- class No_Error

  class for no error condition exceptions

- class Parser

  JSON Data Parser.

- class Syntax_Error

  class for syntax exceptions

- class Value

Base class for handling values.

- class Value_Array

  Class for handling array values.

- class Value_Boolean

  Class for handling boolean values.

- class Value_Complex

  Base class for handling complex values.

- class Value_Null

  class for handling null values

- class Value_Number

  Class for handling numeric values.

- class Value_Number_Int

  Class for handling numeric (int) values.

- class Value_Object

  Class for handling object values.

- class Value_String

  Class for handling numeric values.

7.1.1 Detailed Description

Namespace for all library definitions.

This namespace wraps all the library definitions in order to avoid name clashing with application symbols.

Author:

Kombo Morongo <morongo666@gmail.com>

Chapter 8

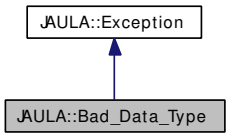
jaula API Reference Data Structure Documentation

8.1 JAULA::Bad_Data_Type Class Reference

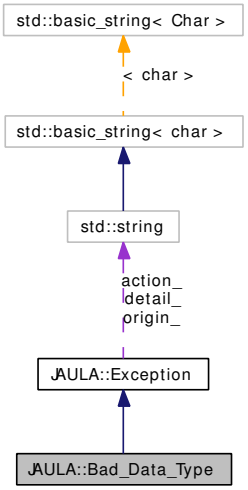
class for incompatible data type exceptions

`#include <jaula_bad_data_type.h>`

Inheritance diagram for JAULA::Bad_Data_Type:



Collaboration diagram for JAULA::Bad_Data_Type:



Public Member Functions

- Bad_Data_Type (std::string const &detail="", std::string const &action="", std::string const &origin="")

  Constructor.

- Bad_Data_Type & operator= (Bad_Data_Type const &orig)

  Assignment operator.

- virtual ~Bad_Data_Type ()

  Destructor.

8.1.1 Detailed Description

class for incompatible data type exceptions

This class defines the exceptions to be thrown in case of incompatible data types usage.

Author:

Kombo Morongo <morongo666@gmail.com>

8.1.2 Constructor & Destructor Documentation

8.1.2.1 JAULA::Bad_Data_Type::Bad_Data_Type (std::string const & *detail* = "", std::string const & *action* = "", std::string const & *origin* = "")

Constructor.

Parameters:

detail detailed description about the exception (what made the execution fail).

action action being performed when the exception arose.

origin Name of the method (and others methods that have called this) by the time the exception arose.

Description

This method construct a new exception instance with JAULA::BAD_DATA_TYPE code.

8.1.2.2 JAULA::Bad_Data_Type::~Bad_Data_Type () [virtual]

Destructor.

8.1.3 Member Function Documentation

8.1.3.1 Bad_Data_Type & JAULA::Bad_Data_Type::operator= (Bad_Data_Type const & *orig*)

Assignment operator.

Parameters:

orig Original instance to copy

Returns:

a reference to the destination instance

Description

Copies the contents of the original instance in the destination.

Note:

This method has been redefined from the base class so the instance type cannot be altered.

The documentation for this class was generated from the following files:

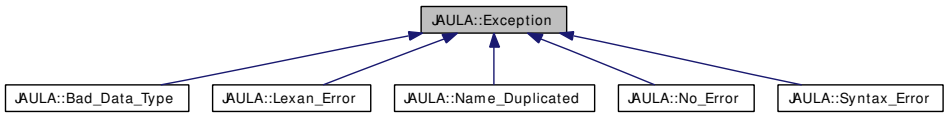
- jaula_bad_data_type.h
- jaula_bad_data_type.cc

8.2 JAULA::Exception Class Reference

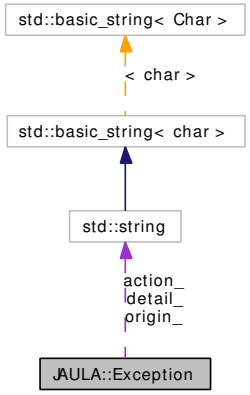
Base class for error handling exceptions.

`#include <jaula_exception.h>`

Inheritance diagram for JAULA::Exception:



Collaboration diagram for JAULA::Exception:



Public Types

- enum ExCode {
  NO_ERROR, BAD_DATA_TYPE, NAME_DUPLICATED, LEXAN_ERROR,
  SYNTAX_ERROR }
  Enumeration of available error codes.

Public Member Functions

- void addOrigin (std::string const &origin)
  Attachs a new origin after the existing ones.
- void display (std::ostream &ostr) const
  Represents the instance in a stream.
- Exception (Exception const &orig)
  Copy constructor.
- Exception (ExCode code=NO_ERROR, std::string const &detail="", std::string const &action="", std::string const &origin="")

Constructor.

- std::string const & getAction (void) const
  Retrieves the action that caused the exception.
- ExCode getCode (void) const
  Retrieves the error code for the exception.
- virtual std::string const & getDetail (void) const
  Retrieves the detailed description for the exception.
- std::string const & getOrigin (void) const
  Retrieves the method being run when the exception arose.
- Exception & operator= (Exception const &orig)
  Assignment operator.
- void setAction (std::string const &action)
  Establishes the action that caused the exception.
- void setDetail (std::string const &detail)
  Establishes the description for the exception.
- void setOrigin (std::string const &origin)
  Establishes the method being run when the exception arose.
- virtual ~Exception ()
  Destructor.

Protected Member Functions

- void setCode (ExCode code)
  Establishes the error code for the exception.

Private Attributes

- std::string action_
  Container action causing the exception.
- ExCode code_
  Container for error code.
- std::string detail_
  Container for exception textual detail.
- std::string origin_
  Container for where the exception was detected.

8.2.1 Detailed Description

Base class for error handling exceptions.

This class is the base for all kind of exceptions thrown by the methods defined in this library.

Author:

Kombo Morongo <morongo666@gmail.com>

8.2.2 Member Enumeration Documentation

8.2.2.1 enum JAULA::Exception::ExCode

Enumeration of available error codes.

Enumerator:

NO_ERROR No Error detected
BAD_DATA_TYPE The data provided for a method is from a wrong type
NAME_DUPLICATED Name repeated for an object property
LEXAN_ERROR Error detected in the lexical Analysis phase
SYNTAX_ERROR Error detected in the syntaxis Analysis phase

8.2.3 Constructor & Destructor Documentation

8.2.3.1 JAULA::Exception::Exception (ExCode *code* = NO_ERROR, std::string const & *detail* = "", std::string const & *action* = "", std::string const & *origin* = "")

Constructor.

Parameters:

code Code for the exception as defined in ExCode
detail detailed description about the exception (what made the execution fail).
action action being performed when the exception arose.
origin Name of the method (and others methods that have called this) by the time the exception arose.

Description

This method construct a new exception instance detailing each available property.

8.2.3.2 JAULA::Exception::Exception (Exception const & *orig*)

Copy constructor.

Parameters:

orig Original instance to copy

Description

Creates a new instance copying the contents from another

8.2.3.3 JAULA::Exception::~Exception () [virtual]

Destructor.

8.2.4 Member Function Documentation

8.2.4.1 void JAULA::Exception::addOrigin (std::string const & *origin*)

Attachs a new origin after the existing ones.

Parameters:

origin Name of the method to add

Description

Adds the name of a method who is supposed to be the next method in the calling stack from where the exception occured so a basic trace can be performed.

8.2.4.2 void JAULA::Exception::display (std::ostream & *ostr*) const

Represents the instance in a stream.

Parameters:

ostr Stream where the instance is to be displayed.

Description

Puts a text representation of the instance contents in a stream.

8.2.4.3 std::string const & JAULA::Exception::getAction (void) const

Retrieves the action that caused the exception.

8.2.4.4 Exception::ExCode JAULA::Exception::getCode (void) const

Retrieves the error code for the exception.

8.2.4.5 std::string const & JAULA::Exception::getDetail (void) const [virtual]

Retrieves the detailed description for the exception.

Note:

This method has been declared as virtual so it can be overrwritten to generate automatic detailed descriptions for specific (derived) exception types.

Reimplemented in JAULA::Name_Duplicated.

8.2.4.6 std::string const & JAULA::Exception::getOrigin (void) const

Retrieves the method being run when the exception arose.

Description

Retrieves the method being run when the exception arose and the methods that called it if available.

8.2.4.7 Exception & JAULA::Exception::operator= (Exception const & *orig*)

Assignment operator.

Parameters:

orig Original instance to copy

Returns:

a reference to the destination instance

Description

Copies the contents of the original instance in the destination.

Note:

This method controls if destination and origin instances are the same so there is no trouble in a = a asignments.

8.2.4.8 void JAULA::Exception::setAction (std::string const & *action*)

Establishes the action that caused the exception.

Parameters:

action action being performed when the exception arose.

8.2.4.9 void JAULA::Exception::setCode (ExCode *code*) [protected]

Establishes the error code for the exception.

Parameters:

code Code for the exception as defined in ExCode

8.2.4.10 void JAULA::Exception::setDetail (std::string const & *detail*)

Establishes the description for the exception.

Parameters:

detail detailed description about the exception (what made the execution fail).

8.2.4.11 void JAULA::Exception::setOrigin (std::string const & *origin*)

Establishes the method being run when the exception arose.

Parameters:

origin Name of the method (and others methods that have called this) by the time the exception arose.

8.2.5 Field Documentation

8.2.5.1 std::string JAULA::Exception::action_ [private]

Container action causing the exception.

8.2.5.2 ExCode JAULA::Exception::code_ [private]

Container for error code.

8.2.5.3 std::string JAULA::Exception::detail_ [private]

Container for exception textual detail.

Reimplemented in JAULA::Name_Duplicated.

8.2.5.4 std::string JAULA::Exception::origin_ [private]

Container for where the exception was detected.

The documentation for this class was generated from the following files:

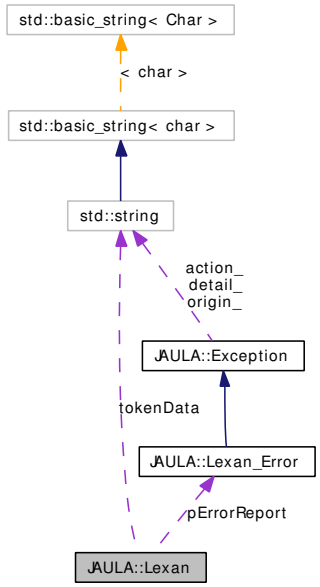
- jaula_exception.h
- jaula_exception.cc

8.3 JAULA::Lexan Class Reference

Lexical Analysis implementation.

`#include <jaula_lexan.h>`

Collaboration diagram for JAULA::Lexan:



Public Member Functions

- Lexan_Error const * getErrorReport (void) const
  Retrieves details for the last error detected.
- std::string const & getTokenData (void) const
  Retrieves last token associated data.
- Lexan (std::istream &in_stream, bool comments_allowed=false)
  Constructor.
- virtual void LexerError (const char *detail)
  Error report.
- virtual int yylex ()
  Retrieves tokens from the input.
- virtual ~Lexan ()
  Destructor.

Private Attributes

- bool commented

Flag for extending language to accept # comments.

- Lexan_Error * pErrorReport

  Pointer to the last exception detected.

- std::string tokenData

  Container for the token associated data.

8.3.1 Detailed Description

Lexical Analysis implementation.

This class implements the lexical analysis for JSON as specified by RFC 4627.

Author:

Kombo Morongo <morongo666@gmail.com>

8.3.2 Constructor & Destructor Documentation

8.3.2.1 JAULA::Lexan::Lexan (std::istream & *in_stream*, bool *comments_allowed* = `false`)

Constructor.

Parameters:

in_stream stream for the input data to analyze

comments_allowed flag to extend basic format and allow for # starting comments in input.

Description

Creates a lexical analysis instance for the specified input stream.

8.3.2.2 virtual JAULA::Lexan::~Lexan () `[virtual]`

Destructor.

8.3.3 Member Function Documentation

8.3.3.1 Lexan_Error const* JAULA::Lexan::getErrorReport (void) const

Retrieves details for the last error detected.

Returns:

a pointer to an instance containing the details for the last error detected during the lexical analysis process or a null pointer in case no error have ocurred during the lexical analysis so far.

8.3.3.2 std::string const& JAULA::Lexan::getTokenData (void) const

Retrieves last token associated data.

Returns:

the data associated for the last token returned for yylex() in case there is such kind of data (token corresponds to a property name or a single value) or undefined otherwise.

8.3.3.3 virtual void JAULA::Lexan::LexerError (const char * *detail*) [virtual]

Error report.

Parameters:

detail Text for the error received

Description

This method is a callback used by the analysis routines to indicate an error condition.

Its current implementation consists of generating the exception instance to be retrieved by getErrorReport().

8.3.3.4 virtual int JAULA::Lexan::yylex () [virtual]

Retrieves tokens from the input.

Returns:

the code for a token read or 0 if the end of data has been reached.

8.3.4 Field Documentation

8.3.4.1 bool JAULA::Lexan::commented [private]

Flag for extending language to accept # comments.

8.3.4.2 Lexan_Error* JAULA::Lexan::pErrorReport [private]

Pointer to the last exception detected.

8.3.4.3 std::string JAULA::Lexan::tokenData [private]

Container for the token associated data.

The documentation for this class was generated from the following file:

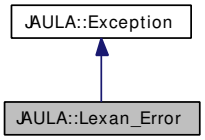
- jaula_lexan.h

8.4 JAULA::Lexan_Error Class Reference

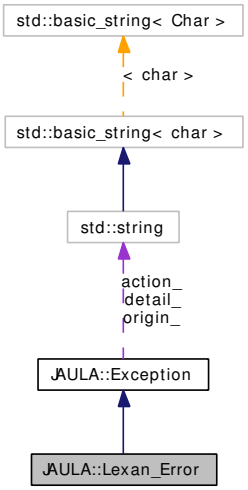
class for lexical analysis exceptions

`#include <jaula_lexan_error.h>`

Inheritance diagram for JAULA::Lexan_Error:



Collaboration diagram for JAULA::Lexan_Error:



Public Member Functions

- Lexan_Error (std::string const &detail="", std::string const &action="", std::string const &origin="")

  Constructor.

- Lexan_Error & operator= (Lexan_Error const &orig)

  Assignment operator.

- virtual ~Lexan_Error ()

  Destructor.

8.4.1 Detailed Description

class for lexical analysis exceptions

This class defines the exceptions to be thrown when errors are detected during the lexical analysis phase.

Author:

Kombo Morongo <morongo666@gmail.com>

8.4.2 Constructor & Destructor Documentation

8.4.2.1 JAULA::Lexan_Error::Lexan_Error (std::string const & *detail* = "", std::string const & *action* = "", std::string const & *origin* = "")

Constructor.

Parameters:

detail detailed description about the exception (what made the execution fail).

action action being performed when the exception arose.

origin Name of the method (and others methods that have called this) by the time the exception arose.

Description

This method construct a new exception instance with JAULA::LEXAN_ERROR code.

8.4.2.2 JAULA::Lexan_Error::~Lexan_Error () [virtual]

Destructor.

8.4.3 Member Function Documentation

8.4.3.1 Lexan_Error & JAULA::Lexan_Error::operator= (Lexan_Error const & *orig*)

Assignment operator.

Parameters:

orig Original instance to copy

Returns:

a reference to the destination instance

Description

Copies the contents of the original instance in the destination.

Note:

This method has been redefined from the base class so the instance type cannot be altered.

The documentation for this class was generated from the following files:

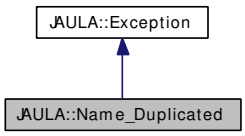
- jaula_lexan_error.h
- jaula_lexan_error.cc

8.5 JAULA::Name_Duplicated Class Reference

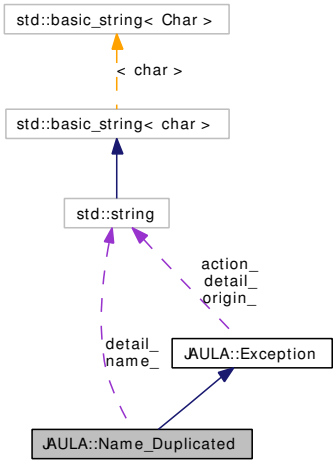
Class for bad object property name exceptions.

`#include <jaula_name_duplicated.h>`

Inheritance diagram for JAULA::Name_Duplicated:



Collaboration diagram for JAULA::Name_Duplicated:



Public Member Functions

- virtual std::string const & getDetail (void) const

  Retrieves the detailed description for the exception.

- std::string const & getName (void) const

  Retrieves the name of the duplicated property.

- Name_Duplicated (std::string const &name="", std::string const &action="", std::string const &origin="")

  Constructor.

- Name_Duplicated & operator= (Name_Duplicated const &orig)

  Assignment operator.

- void setName (std::string const &name)

  Establishes the name for the duplicated property.

- virtual ~Name_Duplicated ()

Destructor.

Private Attributes

- std::string detail_
- std::string name_

8.5.1 Detailed Description

Class for bad object property name exceptions.

This class defines the exceptions to be thrown in case a property name has been used more than once in an object.

Author:

Kombo Morongo <morongo666@gmail.com>

8.5.2 Constructor & Destructor Documentation

8.5.2.1 JAULA::Name_Duplicated::Name_Duplicated (std::string const & *name* = "", std::string const & *action* = "", std::string const & *origin* = "")

Constructor.

Parameters:

name Invalid object property name (deuplicated).

action action being performed when the exception arose.

origin Name of the method (and others methods that have called this) by the time the exception arose.

Description

This method construct a new exception instance with JAULA::Name_DUPLICATED code.

8.5.2.2 JAULA::Name_Duplicated::~Name_Duplicated () [virtual]

Destructor.

8.5.3 Member Function Documentation

8.5.3.1 std::string const & JAULA::Name_Duplicated::getDetail (void) const [virtual]

Retrieves the detailed description for the exception.

Description

This method creates a standard detailed description for the exception based on the duplicated object property name.

Reimplemented from JAULA::Exception.

8.5.3.2 std::string const & JAULA::Name_Duplicated::getName (void) const

Retrieves the name of the duplicated property.

8.5.3.3 Name_Duplicated & JAULA::Name_Duplicated::operator= (Name_Duplicated const & *orig*)

Assignment operator.

Parameters:

orig Original instance to copy

Returns:

a reference to the destination instance

Description

Copies the contents of the original instance in the destination.

Note:

This method has been redefined from the base class so the instance type cannot be altered.

8.5.3.4 void JAULA::Name_Duplicated::setName (std::string const & *name*)

Establishes the name for the duplicated property.

Parameters:

name Name of the object property that launched the exception.

8.5.4 Field Documentation

8.5.4.1 std::string JAULA::Name_Duplicated::detail_ [private]

workspace where to generate class specific detail

Reimplemented from JAULA::Exception.

8.5.4.2 std::string JAULA::Name_Duplicated::name_ [private]

Container for the repeated name

The documentation for this class was generated from the following files:

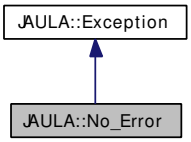
- jaula_name_duplicated.h
- jaula_name_duplicated.cc

8.6 JAULA::No_Error Class Reference

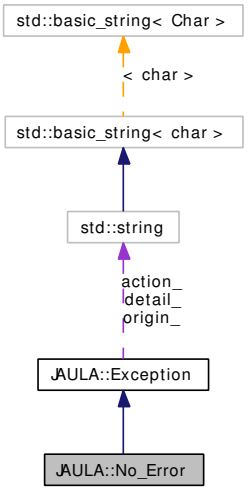
class for no error condition exceptions

`#include <jaula_no_error.h>`

Inheritance diagram for JAULA::No_Error:



Collaboration diagram for JAULA::No_Error:



Public Member Functions

- No_Error (void)
  Constructor.
- No_Error & operator= (No_Error const &orig)
  Assignment operator.
- virtual ~No_Error ()
  Destructor.

8.6.1 Detailed Description

class for no error condition exceptions

This class defines a null exception with no error.

Author:

Kombo Morongo <morongo666@gmail.com>

8.6.2 Constructor & Destructor Documentation

8.6.2.1 JAULA::No_Error::No_Error (void)

Constructor.

Description

This method construct a new exception instance with JAULA::NO_ERROR code.

8.6.2.2 JAULA::No_Error::~No_Error () [virtual]

Destructor.

8.6.3 Member Function Documentation

8.6.3.1 No_Error & JAULA::No_Error::operator= (No_Error const & *orig*)

Assignment operator.

Parameters:

orig Original instance to copy

Returns:

a reference to the destination instance

Description

Copies the contents of the original instance in the destination.

Note:

This method has been redefined from the base class so the instance type cannot be altered.

The documentation for this class was generated from the following files:

- jaula_no_error.h
- jaula_no_error.cc

8.7 JAULA::Parser Class Reference

JSON Data Parser.

`#include <jaula_parse.h>`

Public Member Functions

- Parser (void)

  Constructor.

- ~Parser (void)

  Destructor.

Static Public Member Functions

- static Value_Complex * parseStream (std::istream &inpStream, bool comments_allowed=false, bool full_read=true) throw (Exception)

  Parses JSON data from a stream.

Data Structures

- class Value_Parser

  JSON Value Parser.

8.7.1 Detailed Description

JSON Data Parser.

This class implements the JSON parser itself as specified by RFC 4627.

Author:

Kombo Morongo <morongo666@gmail.com>

8.7.2 Constructor & Destructor Documentation

8.7.2.1 JAULA::Parser::Parser (void)

Constructor.

8.7.2.2 JAULA::Parser::~Parser (void)

Destructor.

8.7.3 Member Function Documentation

8.7.3.1 Value_Complex * JAULA::Parser::parseStream (std::istream & *inpStream*, bool *comments_allowed* = `false`, bool *full_read* = `true`) throw (Exception) `[static]`

Parses JSON data from a stream.

Parameters:

inpStream stream from where to read the data to parse.

comments_allowed flag that if it is true means that the input can contain comments that begin with the hash '#' symbol and ends with eoln (as in bash). If it is false, it means that no comments are allowed in the input and if present will be considered as a syntax error.

full_read flag that if it is true means that the parser must analyze the input stream until the end of file is although it already had got a full array or object from it. In this situation, as JSON specification expects just only one array or object per input, any further data that is not a space for the syntax (or a comment if comments are allowed) will launch a syntax error exception.

If this flag is false, the parser will stop once a full array or object is taken from the input and, on exit, the stream will point to the start of the remaining data.

Returns:

a pointer to memory taken from the heap containing a complex value (array or object) with all the data from the stream parsed on individual / nested items.

Exceptions:

Exception An exception will be thrown as soon as a lexical or syntax error is found analyzing the stream. The result of printing th exception through a stream is a human readable text explaining the error found and an approximation of the error line where occurred.

Description

This method is the entry point for the JSON parser.

Warning:

As this method returns a pointer to memory from the heap, it is up to the user to free it when it is no longer needed in order to avoid leaks.

The documentation for this class was generated from the following files:

- jaula_parse.h
- jaula_parse.cc

8.8 JAULA::Parser::Value_Parser Class Reference

JSON Value Parser.

Public Member Functions

- Value_Parser (void)

  Constructor.

- ~Value_Parser (void)

  Destructor.

Static Public Member Functions

- static Value * parseValue (Lexan &lexan, unsigned int token) throw (Exception)

  reads a single JSON value

Private Types

- enum parser_states {
  START, array_addItem, array_nextItem, error,
  false_value, null_value, number_int_value, number_value,
  property_begin, property_name, property_value, property_next,
  string_value, true_value, END }

  Enumeration for the parser state machine.

Static Private Member Functions

- static void EOFError (Lexan &lexan, Syntax_Error const &ex) throw (Exception)

  Analyzes the reason for an EOF condition.

8.8.1 Detailed Description

JSON Value Parser.

Parses a value from the input stream.

Author:

Kombo Morongo <morongo666@gmail.com>

8.8.2 Member Enumeration Documentation

8.8.2.1 enum JAULA::Parser::Value_Parser::parser_states [private]

Enumeration for the parser state machine.

Description

This type defines a constant for each state that the parser may assume during the process.

Enumerator:

START initial state

array_addItem a new item for an array has been read

array_nextItem an inter-item delimiter for an array has been read

error error condition detected (pseudostate that launches an exception terminating the process)

false_value a boolean false value constant has been read (pseudostate)

null_value a null value constant has been read (pseudostate)

number_int_value a numeric (integer) value has been read (pseudostate)

number_value a numeric (float) value has been read (pseudostate)

property_begin an object initial delimiter has been read

property_name an object property name has been read

property_value an object property delimiter has been read

property_next an inter-property delimiter for an object has been read

string_value a string value has been read (pseudostate)

true_value a boolean true value constant has been read (pseudostate)

END final state (pseudostate)

8.8.3 Constructor & Destructor Documentation

8.8.3.1 JAULA::Parser::Value_Parser::Value_Parser (void)

Constructor.

8.8.3.2 JAULA::Parser::Value_Parser::~Value_Parser (void)

Destructor.

8.8.4 Member Function Documentation

8.8.4.1 void JAULA::Parser::Value_Parser::EOFError (Lexan & *lexan*, Syntax_Error const & *ex*) throw (Exception) [static, private]

Analyzes the reason for an EOF condition.

Parameters:

lexan Reference to the lexical analysis instance that reads from the input.

ex Syntax error to be thrown detailing why an EOF at this point is an error.

Exceptions:

Exception As a result of the execution of this method an exception is thrown with the data conatined in the input parameter or with a JAULA::LEXAN_ERROR type if the EOF is due to an error in the lexical analysis phase.

Description

This method is to be launched whenever an unexpected end of file is encountered. Its implementation includes analyzing if the EOF condition is real or a side effect from a lexical analysis error and chooses to send the Lexan::LexerError Exception or the one received in the input parameter based on this.

8.8.4.2 Value * JAULA::Parser::Value_Parser::parseValue (Lexan & *lexan*, unsigned int *token*) throw (Exception) [static]

reads a single JSON value

Parameters:

lexan Reference to the lexical analysis instance that reads from the input.

token Token read from the upper level. If this token does not belong to an starting value token, an error condition will arise.

Returns:

a pointer to memory taken from the heap containing the value read. If this value belongs to a complex type, all the items that it contains have been recursively parsed.

Exceptions:

Exception An exception will be thrown as soon as a lexical or syntax error is found analyzing the stream. The result of printing the exception retrurned through a stream is a human readable text explaining the error found and an approximation of the error line where it occurred.

Description

This method fully reads a whole value from the input or until a syntax or lexical error is found. Upon execution input stream is positioned so that a new token or EOF can be read from the input.

The documentation for this class was generated from the following files:

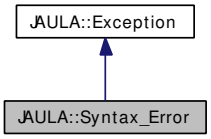
- jaula_parse.h
- jaula_parse.cc

8.9 JAULA::Syntax_Error Class Reference

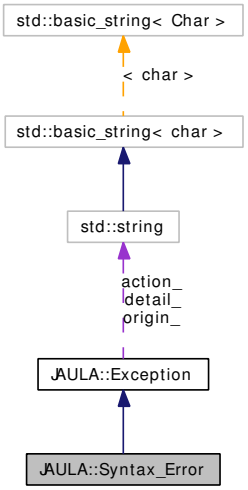
class for syntax exceptions

`#include <jaula_syntax_error.h>`

Inheritance diagram for JAULA::Syntax_Error:



Collaboration diagram for JAULA::Syntax_Error:



Public Member Functions

- Syntax_Error & operator= (Syntax_Error const &orig)

  Assignment operator.

- Syntax_Error (std::string const &detail="", std::string const &action="", std::string const &origin="")

  Constructor.

- virtual ~Syntax_Error ()

  Destructor.

8.9.1 Detailed Description

class for syntax exceptions

This class defines the exceptions to be thrown when errors are detected during the syntax analysis phase.

Author:

Kombo Morongo <morongo666@gmail.com>

8.9.2 Constructor & Destructor Documentation

8.9.2.1 JAULA::Syntax_Error::Syntax_Error (std::string const & *detail* = "", std::string const & *action* = "", std::string const & *origin* = "")

Constructor.

Parameters:

detail detailed description about the exception (what made the execution fail).

action action being performed when the exception arose.

origin Name of the method (and others methods that have called this) by the time the exception arose.

Description

This method construct a new exception instance with JAULA::SYNTAX_ERROR code.

8.9.2.2 JAULA::Syntax_Error::~Syntax_Error () [virtual]

Destructor.

8.9.3 Member Function Documentation

8.9.3.1 Syntax_Error & JAULA::Syntax_Error::operator= (Syntax_Error const & *orig*)

Assignment operator.

Parameters:

orig Original instance to copy

Returns:

a reference to the destination instance

Description

Copies the contents of the original instance in the destination.

Note:

This method has been redefined from the base class so the instance type cannot be altered.

The documentation for this class was generated from the following files:

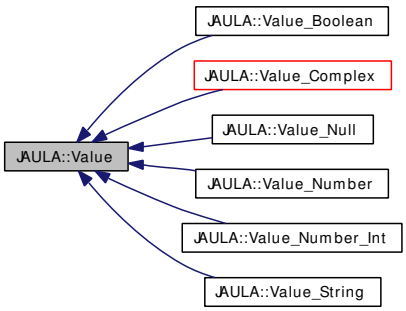
- jaula_syntax_error.h
- jaula_syntax_error.cc

8.10 JAULA::Value Class Reference

Base class for handling values.

`#include <jaula_value.h>`

Inheritance diagram for JAULA::Value:



Public Types

- enum ValueType {
  TYPE_NULL, TYPE_BOOLEAN, TYPE_STRING, TYPE_NUMBER,
  TYPE_NUMBER_INT, TYPE_ARRAY, TYPE_OBJECT }
  Enumeration of available value types.

Public Member Functions

- ValueType getType (void) const
  Retrieves the value type for the instance.
- Value & operator= (Value const &orig) throw (Bad_Data_Type)
  Assignment operator.
- virtual void repr (std::ostream &ostr) const=0
  Represents the instance in a stream.
- virtual void set (Value const &origin) throw (Bad_Data_Type)
  Copies the contents of one instance into anothe.
- virtual ~Value ()
  Destructor.

Static Public Member Functions

- static Value * duplicate (Value const &orig)
  Creates a duplicate of a value.

Protected Member Functions

- Value (ValueType Type)

  Constructor.

Private Attributes

- ValueType Type_

  Container for error code.

8.10.1 Detailed Description

Base class for handling values.

This class is the abstract base for all the containers for values according to the JSON specification.

Author:

Kombo Morongo <morongo666@gmail.com>

8.10.2 Member Enumeration Documentation

8.10.2.1 enum JAULA::Value::ValueType

Enumeration of available value types.

Enumerator:

TYPE_NULL JSON Null value

TYPE_BOOLEAN JSON Boolean value

TYPE_STRING JSON String value

TYPE_NUMBER JSON Number value

TYPE_NUMBER_INT JSON Number value adapted to hold integer quantities

TYPE_ARRAY JSON Array of values

TYPE_OBJECT JSON Object

8.10.3 Constructor & Destructor Documentation

8.10.3.1 JAULA::Value::~Value () [virtual]

Destructor.

8.10.3.2 JAULA::Value::Value (ValueType *Type*) [protected]

Constructor.

Parameters:

Type Type of value to be contained by the instance

Description

This method construct a new instance by specifying its type.

Note:

A ValueType for the instance is immutable during all the life cycle, this is the only method that permits specifying the value type.

8.10.4 Member Function Documentation

8.10.4.1 Value * JAULA::Value::duplicate (Value const & *orig*) [static]

Creates a duplicate of a value.

Parameters:

orig Original instance to duplicate.

Returns:

a pointer to memory taken from the heap (by means of the new operator) and containing a deep copy of the original value.

Warning:

As this method returns a pointer to memory allocated from the heap, it is up to the caller to release once it is no longer needed in order to avoid leaks.

8.10.4.2 Value::ValueType JAULA::Value::getType (void) const

Retrieves the value type for the instance.

Note:

Value types are immutable during instance's life cycle and can only be specified at construction time.

8.10.4.3 Value & JAULA::Value::operator= (Value const & *orig*) throw (Bad_Data_Type)

Assignment operator.

Parameters:

orig Original instance to copy

Returns:

a reference to the destination instance

Exceptions:

Bad_Data_Type This exception is launched in case that origin and destination value types are different.

Description

Copies the contents of the original instance in the destination.

Note:

This method controls if destination and origin instances are the same so there is no trouble in a = a asignments.

8.10.4.4 virtual void JAULA::Value::repr (std::ostream & *ostr*) const [pure virtual]

Represents the instance in a stream.

Parameters:

ostr Stream where the instance is to be represented.

Description

writes the instance content in JSON notation in a stream.

Implemented in JAULA::Value_Array, JAULA::Value_Boolean, JAULA::Value_Null, JAULA::Value_Number, JAULA::Value_Number_Int, JAULA::Value_Object, and JAULA::Value_String.

8.10.4.5 void JAULA::Value::set (Value const & *origin*) throw (Bad_Data_Type) [virtual]

Copies the contents of one instance into anothe.

Parameters:

origin Reference to the value to be copied.

Exceptions:

Bad_Data_Type This exception is launched in case that origin and destination value types are different.

Reimplemented in JAULA::Value_Array, JAULA::Value_Boolean, JAULA::Value_Null, JAULA::Value_Number, JAULA::Value_Number_Int, JAULA::Value_Object, and JAULA::Value_String.

8.10.5 Field Documentation

8.10.5.1 ValueType JAULA::Value::Type_ [private]

Container for error code.

The documentation for this class was generated from the following files:

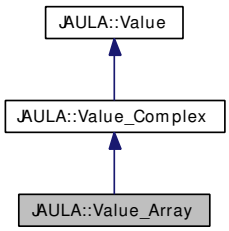
- jaula_value.h
- jaula_value.cc

8.11 JAULA::Value_Array Class Reference

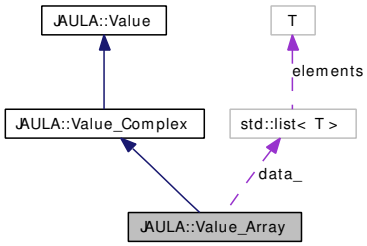
Class for handling array values.

`#include <jaula_value_array.h>`

Inheritance diagram for JAULA::Value_Array:



Collaboration diagram for JAULA::Value_Array:



Public Types

- typedef std::list< Value * > dataType

  Data type for value contents.

Public Member Functions

- void addItem (Value const &item)

  Appends one item to the array.

- virtual void clear (void)

  Empties the contents of an instance.

- virtual bool empty (void) const

  True if the instance is empty.

- dataType const & getData (void) const

  Retrieves the array of values contained by the instance.

- virtual void repr (std::ostream &ostr) const

  Represents the instance in a stream.

- virtual void set (Value const &origin) throw (Bad_Data_Type)

  Copies the contents of one instance into anothe.

- void set (dataType const &data)

  Establishes the contents of the instance.

- virtual size_t size (void) const

  Number of elements contained.

- Value_Array (dataType const &data)

  Data Constructor.

- Value_Array (void)

  Default Constructor.

- virtual ~Value_Array ()

  Destructor.

Private Attributes

- dataType data_

  Container to hold the value itself.

8.11.1 Detailed Description

Class for handling array values.

This class is a container for JSON arrays

Author:

Kombo Morongo <morongo666@gmail.com>

8.11.2 Member Typedef Documentation

8.11.2.1 typedef std::list<Value *> JAULA::Value_Array::dataType

Data type for value contents.

Description

Definition for the data container internal structure.

This data type defines a list of pointers to any kind of values (including arrays).

8.11.3 Constructor & Destructor Documentation

8.11.3.1 JAULA::Value_Array::Value_Array (void)

Default Constructor.

Description

The default constructor initializes an empty array.

8.11.3.2 JAULA::Value_Array::Value_Array (dataType const & *data*)

Data Constructor.

Parameters:

data Reference to the data to be copied

Description

This constructor generates a new instance by making a deep copy of the original data.

8.11.3.3 JAULA::Value_Array::~Value_Array () [virtual]

Destructor.

Note:

The destruction process releases all the memory associated to the data structure so any reference to the array or any element in it will be void.

8.11.4 Member Function Documentation

8.11.4.1 void JAULA::Value_Array::addItem (Value const & *item*)

Appends one item to the array.

Parameters:

item Item to be appended

Description

Appends a deep copy of the item value at the end of the array.

8.11.4.2 void JAULA::Value_Array::clear (void) [virtual]

Empties the contents of an instance.

Description

Erases the contained array.

Note:

As the array is destroyed by the process, any references to it or to its elements will be void.

Implements JAULA::Value_Complex.

8.11.4.3 bool JAULA::Value_Array::empty (void) const [virtual]

True if the instance is empty.

Returns:

true if there are no single elements contained in the array and false otherwise.

Implements JAULA::Value_Complex.

8.11.4.4 Value_Array::dataType const & JAULA::Value_Array::getData (void) const

Retrieves the array of values contained by the instance.

8.11.4.5 void JAULA::Value_Array::repr (std::ostream & *ostr*) const [virtual]

Represents the instance in a stream.

Parameters:

ostr Stream where the instance is to be represented.

Description

writes the instance content in JSON notation in a stream.

Implements JAULA::Value.

8.11.4.6 void JAULA::Value_Array::set (Value const & *origin*) throw (Bad_Data_Type) [virtual]

Copies the contents of one instance into anothe.

Parameters:

origin Reference to the value to be copied.

Exceptions:

Bad_Data_Type This exception is launched in case that origin and destination value types are different.

Note:

The destination array is destroyed during the asignment process so, any references to it or to its former elements will be void.

Reimplemented from JAULA::Value.

8.11.4.7 void JAULA::Value_Array::set (dataType const & *data*)

Establishes the contents of the instance.

Parameters:

data Array to assign to the instance

Note:

The destination array is destroyed during the asignment process so, any references to it or to its former elements will be void.

8.11.4.8 size_t JAULA::Value_Array::size (void) const [virtual]

Number of elements contained.

Returns:

the number of single elements contained by the array.

Implements JAULA::Value_Complex.

8.11.5 Field Documentation

8.11.5.1 dataType JAULA::Value_Array::data_ [private]

Container to hold the value itself.

The documentation for this class was generated from the following files:

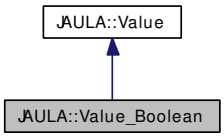
- jaula_value_array.h
- jaula_value_array.cc

8.12 JAULA::Value_Boolean Class Reference

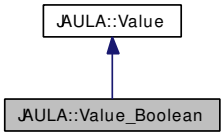
Class for handling boolean values.

`#include <jaula_value_boolean.h>`

Inheritance diagram for JAULA::Value_Boolean:



Collaboration diagram for JAULA::Value_Boolean:



Public Member Functions

- bool getData (void) const

  Retrieves the value contained by the instance.

- virtual void repr (std::ostream &ostr) const

  Represents the instance in a stream.

- virtual void set (Value const &origin) throw (Bad_Data_Type)

  Copies the contents of one instance into anothe.

- void set (bool data)

  Establishes the contents of the instance.

- Value_Boolean (bool data=false)

  Constructor.

- virtual ~Value_Boolean ()

  Destructor.

Private Attributes

- bool data_

  Container to hold the value itself.

8.12.1 Detailed Description

Class for handling boolean values.

This class is a container for JSON boolean values

Author:

Kombo Morongo <morongo666@gmail.com>

8.12.2 Constructor & Destructor Documentation

8.12.2.1 JAULA::Value_Boolean::Value_Boolean (bool *data* = `false`)

Constructor.

Parameters:

data Initial value to be hold by the conatiner

8.12.2.2 JAULA::Value_Boolean::~Value_Boolean () `[virtual]`

Destructor.

8.12.3 Member Function Documentation

8.12.3.1 bool JAULA::Value_Boolean::getData (void) const

Retrieves the value contained by the instance.

8.12.3.2 void JAULA::Value_Boolean::repr (std::ostream & *ostr*) const `[virtual]`

Represents the instance in a stream.

Parameters:

ostr Stream where the instance is to be represented.

Description

writes the instance content in JSON notation in a stream.

Implements JAULA::Value.

8.12.3.3 void JAULA::Value_Boolean::set (Value const & *origin*) throw (Bad_Data_Type) `[virtual]`

Copies the contents of one instance into anothe.

Parameters:

origin Reference to the value to be copied.

Exceptions:

Bad_Data_Type This exception is launched in case that origin and destination value types are different.

Reimplemented from JAULA::Value.

8.12.3.4 void JAULA::Value_Boolean::set (bool *data*)

Establishes the contents of the instance.

Parameters:

data Value to assign to the instance

8.12.4 Field Documentation

8.12.4.1 bool JAULA::Value_Boolean::data_ [private]

Container to hold the value itself.

The documentation for this class was generated from the following files:

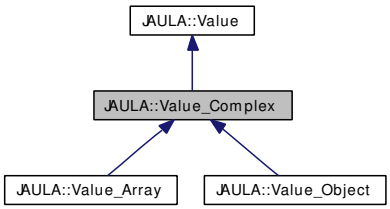
- jaula_value_boolean.h
- jaula_value_boolean.cc

8.13 JAULA::Value_Complex Class Reference

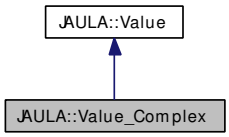
Base class for handling complex values.

`#include <jaula_value_complex.h>`

Inheritance diagram for JAULA::Value_Complex:



Collaboration diagram for JAULA::Value_Complex:



Public Member Functions

- virtual void clear (void)=0

  Empties the contents of an instance.

- virtual bool empty (void) const=0

  True if the instance is empty.

- virtual size_t size (void) const=0

  Number of elements contained.

- virtual ~Value_Complex ()

  Destructor.

Protected Member Functions

- Value_Complex (ValueType Type)

  Constructor.

8.13.1 Detailed Description

Base class for handling complex values.

This class defines an abstract base class for JSON complex values (arrays and objects)

Author:

Kombo Morongo <morongo666@gmail.com>

8.13.2 Constructor & Destructor Documentation

8.13.2.1 JAULA::Value_Complex::~Value_Complex () [virtual]

Destructor.

8.13.2.2 JAULA::Value_Complex::Value_Complex (ValueType *Type*) [protected]

Constructor.

Parameters:

Type Type of value to be contained by the instance

Description

This method construct a new instance by specifying its type.

Note:

A ValueType for the instance is immutable during all the life cycle, this is the only method that permits specifying the value type.

8.13.3 Member Function Documentation

8.13.3.1 virtual void JAULA::Value_Complex::clear (void) [pure virtual]

Empties the contents of an instance.

Description

Erases all the instance content.

Implemented in JAULA::Value_Array, and JAULA::Value_Object.

8.13.3.2 virtual bool JAULA::Value_Complex::empty (void) const [pure virtual]

True if the instance is empty.

Returns:

true if there are no single elements contained by the instance and false otherwise.

Implemented in JAULA::Value_Array, and JAULA::Value_Object.

8.13.3.3 virtual size_t JAULA::Value_Complex::size (void) const [pure virtual]

Number of elements contained.

Returns:

the number of single elements contained by the instance.

Implemented in JAULA::Value_Array, and JAULA::Value_Object.

The documentation for this class was generated from the following files:

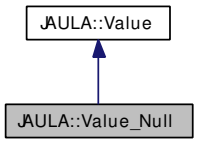
- jaula_value_complex.h
- jaula_value_complex.cc

8.14 JAULA::Value_Null Class Reference

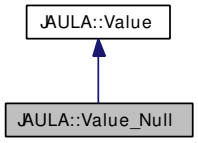
class for handling null values

`#include <jaula_value_null.h>`

Inheritance diagram for JAULA::Value_Null:



Collaboration diagram for JAULA::Value_Null:



Public Member Functions

- virtual void repr (std::ostream &ostr) const

  Represents the instance in a stream.

- virtual void set (Value const &origin) throw (Bad_Data_Type)

  Copies the contents of one instance into anothe.

- Value_Null (void)

  Constructor.

- virtual ~Value_Null ()

  Destructor.

8.14.1 Detailed Description

class for handling null values

This class is a container for JSON Null values

Author:

Kombo Morongo <morongo666@gmail.com>

8.14.2 Constructor & Destructor Documentation

8.14.2.1 JAULA::Value_Null::Value_Null (void)

Constructor.

8.14.2.2 JAULA::Value_Null::~Value_Null () [virtual]

Destructor.

8.14.3 Member Function Documentation

8.14.3.1 void JAULA::Value_Null::repr (std::ostream & *ostr*) const [virtual]

Represents the instance in a stream.

Parameters:

ostr Stream where the instance is to be represented.

Description

writes the instance content in JSON notation in a stream.

Implements JAULA::Value.

8.14.3.2 void JAULA::Value_Null::set (Value const & *origin*) throw (Bad_Data_Type) [virtual]

Copies the contents of one instance into anothe.

Parameters:

origin Reference to the value to be copied.

Exceptions:

Bad_Data_Type This exception is launched in case that origin and destination value types are different.

Reimplemented from JAULA::Value.

The documentation for this class was generated from the following files:

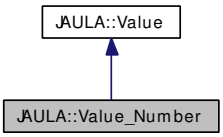
- jaula_value_null.h
- jaula_value_null.cc

8.15 JAULA::Value_Number Class Reference

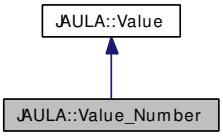
Class for handling numeric values.

`#include <jaula_value_number.h>`

Inheritance diagram for JAULA::Value_Number:



Collaboration diagram for JAULA::Value_Number:



Public Member Functions

- double getData (void) const

  Retrieves the value contained by the instance.

- virtual void repr (std::ostream &ostr) const

  Represents the instance in a stream.

- virtual void set (Value const &origin) throw (Bad_Data_Type)

  Copies the contents of one instance into anothe.

- void set (double data)

  Establishes the contents of the instance.

- Value_Number (double data=0)

  Constructor.

- virtual ~Value_Number ()

  Destructor.

Private Attributes

- double data_

  Container to hold the value itself.

8.15.1 Detailed Description

Class for handling numeric values.

This class is a container for JSON numeric values.

Author:

Kombo Morongo <morongo666@gmail.com>

8.15.2 Constructor & Destructor Documentation

8.15.2.1 JAULA::Value_Number::Value_Number (double *data* = 0)

Constructor.

Parameters:

data Initial value to be hold by the conatiner

8.15.2.2 JAULA::Value_Number::~Value_Number () [virtual]

Destructor.

8.15.3 Member Function Documentation

8.15.3.1 double JAULA::Value_Number::getData (void) const

Retrieves the value contained by the instance.

8.15.3.2 void JAULA::Value_Number::repr (std::ostream & *ostr*) const [virtual]

Represents the instance in a stream.

Parameters:

ostr Stream where the instance is to be represented.

Description

writes the instance content in JSON notation in a stream.

Implements JAULA::Value.

8.15.3.3 void JAULA::Value_Number::set (Value const & *origin*) throw (Bad_Data_Type) [virtual]

Copies the contents of one instance into anothe.

Parameters:

origin Reference to the value to be copied.

Exceptions:

Bad_Data_Type This exception is launched in case that origin and destination value types are different.

Reimplemented from JAULA::Value.

8.15.3.4 void JAULA::Value_Number::set (double *data*)

Establishes the contents of the instance.

Parameters:

data Value to assign to the instance

8.15.4 Field Documentation

8.15.4.1 double JAULA::Value_Number::data_ [private]

Container to hold the value itself.

The documentation for this class was generated from the following files:

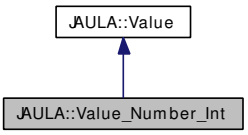
- jaula_value_number.h
- jaula_value_number.cc

8.16 JAULA::Value_Number_Int Class Reference

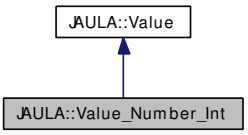
Class for handling numeric (int) values.

`#include <jaula_value_number_int.h>`

Inheritance diagram for JAULA::Value_Number_Int:



Collaboration diagram for JAULA::Value_Number_Int:



Public Member Functions

- long getData (void) const

  Retrieves the value contained by the instance.

- virtual void repr (std::ostream &ostr) const

  Represents the instance in a stream.

- virtual void set (Value const &origin) throw (Bad_Data_Type)

  Copies the contents of one instance into anothe.

- void set (long data)

  Establishes the contents of the instance.

- Value_Number_Int (long data=0)

  Constructor.

- virtual ~Value_Number_Int ()

  Destructor.

Private Attributes

- long data_

  Container to hold the value itself.

8.16.1 Detailed Description

Class for handling numeric (int) values.

This class is a container for JSON numeric values adapted to hold only integer values.

Author:

Kombo Morongo <morongo666@gmail.com>

8.16.2 Constructor & Destructor Documentation

8.16.2.1 JAULA::Value_Number_Int::Value_Number_Int (long *data* = 0)

Constructor.

Parameters:

data Initial value to be hold by the conatiner

8.16.2.2 JAULA::Value_Number_Int::~Value_Number_Int () [virtual]

Destructor.

8.16.3 Member Function Documentation

8.16.3.1 long JAULA::Value_Number_Int::getData (void) const

Retrieves the value contained by the instance.

8.16.3.2 void JAULA::Value_Number_Int::repr (std::ostream & *ostr*) const [virtual]

Represents the instance in a stream.

Parameters:

ostr Stream where the instance is to be represented.

Description

writes the instance content in JSON notation in a stream.

Implements JAULA::Value.

8.16.3.3 void JAULA::Value_Number_Int::set (Value const & *origin*) throw (Bad_Data_Type) [virtual]

Copies the contents of one instance into anothe.

Parameters:

origin Reference to the value to be copied.

Exceptions:

Bad_Data_Type This exception is launched in case that origin and destination value types are different.

Reimplemented from JAULA::Value.

8.16.3.4 void JAULA::Value_Number_Int::set (long *data*)

Establishes the contents of the instance.

Parameters:

data Value to assign to the instance

8.16.4 Field Documentation

8.16.4.1 long JAULA::Value_Number_Int::data_ [private]

Container to hold the value itself.

The documentation for this class was generated from the following files:

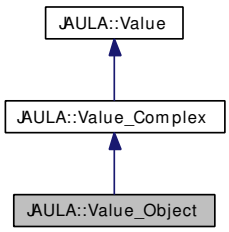
- jaula_value_number_int.h
- jaula_value_number_int.cc

8.17 JAULA::Value_Object Class Reference

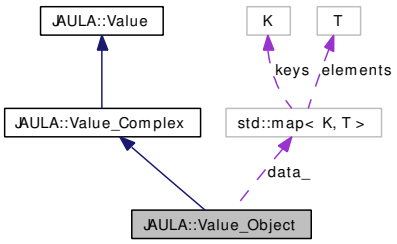
Class for handling object values.

`#include <jaula_value_object.h>`

Inheritance diagram for JAULA::Value_Object:



Collaboration diagram for JAULA::Value_Object:



Public Types

- typedef std::map< std::string, Value * > dataType

  Data type for value contents.

Public Member Functions

- virtual void clear (void)

  Empties the contents of an instance.

- virtual bool empty (void) const

  True if the instance is empty.

- dataType const & getData (void) const

  Retrieves the map of values contained by the instance.

- void insertItem (std::string const &name, Value const &item) throw (Name_Duplicated)

  Inserts one item to the object.

- virtual void repr (std::ostream &ostr) const

  Represents the instance in a stream.

- virtual void set (Value const &origin) throw (Bad_Data_Type)

  Copies the contents of one instance into anothe.

- void set (dataType const &data)

  Establishes the contents of the instance.

- virtual size_t size (void) const

  Number of elements contained.

- Value_Object (dataType const &data)

  Data Constructor.

- Value_Object (void)

  Default Constructor.

- virtual ~Value_Object ()

  Destructor.

Private Attributes

- dataType data_

  Container to hold the value itself.

8.17.1 Detailed Description

Class for handling object values.

This class is a container for JSON objects

Author:

Kombo Morongo <morongo666@gmail.com>

8.17.2 Member Typedef Documentation

8.17.2.1 typedef std::map<std::string, Value *> JAULA::Value_Object::dataType

Data type for value contents.

Description

Definition for the data container internal structure.

This data type defines a map of pointers to any kind of values (including objects) indexed by a property name that is always a string.

8.17.3 Constructor & Destructor Documentation

8.17.3.1 JAULA::Value_Object::Value_Object (void)

Default Constructor.

Description

The default constructor initializes an empty object.

8.17.3.2 JAULA::Value_Object::Value_Object (dataType const & *data*)

Data Constructor.

Parameters:

data Reference to the data to be copied

Description

This constructor generates a new instance by making a deep copy of the original data.

8.17.3.3 JAULA::Value_Object::~Value_Object () [virtual]

Destructor.

Note:

The destruction process releases all the memory associated to the data structure so any reference to the object or any element in it will be void.

8.17.4 Member Function Documentation

8.17.4.1 void JAULA::Value_Object::clear (void) [virtual]

Empties the contents of an instance.

Description

Erases the contained array.

Note:

As the array is destroyed by the process, any references to it or to its elements will be void.

Implements JAULA::Value_Complex.

8.17.4.2 bool JAULA::Value_Object::empty (void) const [virtual]

True if the instance is empty.

Returns:

true if there are no single elements contained in the object and false otherwise.

Implements JAULA::Value_Complex.

8.17.4.3 Value_Object::dataType const & JAULA::Value_Object::getData (void) const

Retrieves the map of values contained by the instance.

8.17.4.4 void JAULA::Value_Object::insertItem (std::string const & *name*, Value const & *item*) throw (Name_Duplicated)

Inserts one item to the object.

Parameters:

name Name for the property to insert

item Item value to be inserted

Exceptions:

Name_Duplicated This exception is thrown in case the object already has a property with the same name as the one to insert.

Description

Inserts a deep copy of the item value at the specified name fot the object.

8.17.4.5 void JAULA::Value_Object::repr (std::ostream & *ostr*) const [virtual]

Represents the instance in a stream.

Parameters:

ostr Stream where the instance is to be represented.

Description

writes the instance content in JSON notation in a stream.

Implements JAULA::Value.

8.17.4.6 void JAULA::Value_Object::set (Value const & *origin*) throw (Bad_Data_Type) [virtual]

Copies the contents of one instance into anothe.

Parameters:

origin Reference to the value to be copied.

Exceptions:

Bad_Data_Type This exception is launched in case that origin and destination value types are different.

Note:

The destination object is destroyed during the asignment process so, any references to it or to its former elements will be void.

Reimplemented from JAULA::Value.

8.17.4.7 void JAULA::Value_Object::set (dataType const & *data*)

Establishes the contents of the instance.

Parameters:

data map of values to assign to the instance

Note:

The destination object is destroyed during the asignment process so, any references to it or to its former elements will be void.

8.17.4.8 size_t JAULA::Value_Object::size (void) const [virtual]

Number of elements contained.

Returns:

the number of single elements contained by the object.

Implements JAULA::Value_Complex.

8.17.5 Field Documentation

8.17.5.1 dataType JAULA::Value_Object::data_ [private]

Container to hold the value itself.

The documentation for this class was generated from the following files:

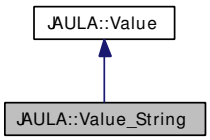
- jaula_value_object.h
- jaula_value_object.cc

8.18 JAULA::Value_String Class Reference

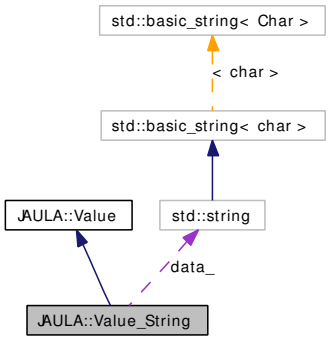
Class for handling numeric values.

`#include <jaula_value_string.h>`

Inheritance diagram for JAULA::Value_String:



Collaboration diagram for JAULA::Value_String:



Public Member Functions

- std::string const & getData (void) const

  Retrieves the value contained by the instance.

- virtual void repr (std::ostream &ostr) const

  Represents the instance in a stream.

- virtual void set (Value const &origin) throw (Bad_Data_Type)

  Copies the contents of one instance into anothe.

- void set (std::string const &data)

  Establishes the contents of the instance.

- Value_String (std::string const &data="")

  Constructor.

- virtual ~Value_String ()

  Destructor.

Static Public Member Functions

- static void stringRepr (std::ostream &ostr, std::string const &str)

  Represents a string in JSON Notation.

Private Attributes

- std::string data_

  Container to hold the value itself.

8.18.1 Detailed Description

Class for handling numeric values.

This class is a container for JSON numeric values.

Author:

Kombo Morongo <morongo666@gmail.com>

8.18.2 Constructor & Destructor Documentation

8.18.2.1 JAULA::Value_String::Value_String (std::string const & *data* = "")

Constructor.

Parameters:

data Initial value to be hold by the conatiner

8.18.2.2 JAULA::Value_String::~Value_String () [virtual]

Destructor.

8.18.3 Member Function Documentation

8.18.3.1 std::string const & JAULA::Value_String::getData (void) const

Retrieves the value contained by the instance.

8.18.3.2 void JAULA::Value_String::repr (std::ostream & *ostr*) const [virtual]

Represents the instance in a stream.

Parameters:

ostr Stream where the instance is to be represented.

Description

writes the instance content in JSON notation in a stream.

Implements JAULA::Value.

8.18.3.3 void JAULA::Value_String::set (Value const & *origin*) throw (Bad_Data_Type) [virtual]

Copies the contents of one instance into anothe.

Parameters:

origin Reference to the value to be copied.

Exceptions:

Bad_Data_Type This exception is launched in case that origin and destination value types are different.

Reimplemented from JAULA::Value.

8.18.3.4 void JAULA::Value_String::set (std::string const & *data*)

Establishes the contents of the instance.

Parameters:

data Value to assign to the instance

8.18.3.5 void JAULA::Value_String::stringRepr (std::ostream & *ostr*, std::string const & *str*) [static]

Represents a string in JSON Notation.

Parameters:

ostr stream where the representation is to be performed.

str String to represent

Description

Represents a string in JSON notation, surrounding it with quotes and changing all the non-printable symbols by its control codes or unicode value.

8.18.4 Field Documentation

8.18.4.1 std::string JAULA::Value_String::data_ [private]

Container to hold the value itself.

The documentation for this class was generated from the following files:

- jaula_value_string.h
- jaula_value_string.cc

Index