PikaScript

Generated by Doxygen 1.7.3

Wed May 18 2011 02:17:25

Contents

Chapter 1

Copyright

PikaScript is released under the "New Simplified BSD License".

http://www.opensource.org/licenses/bsd-license.php

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Class Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

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

Chapter 6

Namespace Documentation

6.1 Pika Namespace Reference

The PikaScript namespace.

Classes

- class bound_mem_fun_t

  bound_mem_fun_t is a member functor bound to a specific C++ object through a pointer.

- class Dumb

  We use this dummy class to specialize member functions for arbitrary types (including void, references etc).

- class Exception

  The PikaScript exception class.

- class STLValue

  STLValue is the reference implementation of a PikaScript variable.

- struct Script

  Script is a meta-class that groups all the core classes of the PikaScript interpreter together (except for the value class).

- struct StdConfig

  StdConfig is a configuration class for Script that uses the reference implementations of STLValue and Script::STLVariables.

- class QuickVars

Typedefs

- typedef Script< StdConfig > StdScript

  This typedef exist for your convenience.

Enumerations

- enum Precedence {

  NO_TRACE = 0, TRACE_ERROR = 1, TRACE_CALL = 2, TRACE_LOOP = 3,

  STATEMENT = 4, BODY = 5, ARGUMENT = 6, BRACKETS = 7,

  ASSIGN = 8, LOGICAL_OR = 9, LOGICAL_AND = 10, BIT_OR = 11,

  BIT_XOR = 12, BIT_AND = 13, EQUALITY = 14, COMPARE = 15,

  CONCAT = 16, SHIFT = 17, ADD_SUB = 18, MUL_DIV = 19,

  PREFIX = 20, POSTFIX = 21, DEFINITION = 22 }

  Precedence levels are used both internally for the parser and externally for the tracing mechanism.

Functions

- template<class C , class A0 , class R >
  bound_mem_fun_t< C, A0, R > bound_mem_fun (R(C::∗m)(A0), C ∗o)

  bound_mem_fun creates a member functor bound to a specific C++ object through a pointer.

Conversion routines for string <-> other types.

For some of these we could use stdlib implementations yes, but:

1. Some of them (e.g. atof, strtod) behaves differently depending on global "locale" setting. We can't have that.
2. The stdlib implementations can be slow (e.g. my double->string conversion is about 3 times faster than MSVC CRT).
3. Pika requires high-precision string representation and proper handling of trailing 9's etc.

- template<class S >
  std::string toStdString (const S &s)

  Converts the string s to a standard C++ string.

- template<class S >
  ulong hexToLong (typename S::const_iterator &p, const typename S::const_iterator &e)

  Converts a string in hexadecimal form to an ulong integer.

- template<class S >
  long stringToLong (typename S::const_iterator &p, const typename S::const_iterator &e)

  Converts a string in decimal form to a signed long integer.

- template<class S , typename T >
  S intToString (T i, int radix=10, int minLength=1)

  Converts the integer `i` to a string with a radix and minimum length of your choice.

- template<class S >
  double stringToDouble (typename S::const_iterator &p, const typename S::const_iterator &e)

  Converts a string in scientific e notation (e.g. -12.34e-3) to a double floating point value.

- template<class S >
  bool stringToDouble (const S &s, double &d)

  A convenient utility routine that tries to convert the entire string `s` (in scientific e notation) to a double, returning true on success or false if the string is not in valid syntax.

- template<class S >
  S doubleToString (double d, int precision=14)

  Converts the double `d` to a string (in scientific e notation, e.g. -12.34e-3).

- template<class S >
  S unescape (typename S::const_iterator &p, const typename S::const_iterator &e)

  Converts a string that is either enclosed in single (' ') or double (" ") quotes.

- template<class S >
  S escape (const S &s)

  Depending on the contents of the source string `s` it is encoded either in single (' ') or double (" ") quotes.

6.1.1 Detailed Description

The PikaScript namespace.

6.1.2 Typedef Documentation

6.1.2.1 typedef Script<StdConfig> Pika::StdScript

This typedef exist for your convenience.

If you wish to use the reference implementation of PikaScript you can now simply instantiate Pika::StdScript::FullRoot and off you go.

Definition at line 581 of file PikaScript.h.

6.1.3 Enumeration Type Documentation

6.1.3.1 enum Pika::Precedence

Precedence levels are used both internally for the parser and externally for the tracing mechanism.

Enumerator:

NO_TRACE used only for tracing with tick()

TRACE_ERROR used only for tracing with tick()

TRACE_CALL used only for tracing with tick()

TRACE_LOOP used only for tracing with tick()

STATEMENT x; y;

BODY if () x, for () x

ARGUMENT (x, y)

BRACKETS (x) [x]

ASSIGN x=y x*=y x/=y x\=y x=y x+=y x-=y x<<=y x>>=y x#=y x&=y x^=y x|=y

LOGICAL_OR x||y

LOGICAL_AND x&&y

BIT_OR x|y

BIT_XOR x^y

BIT_AND x&y

EQUALITY x===y x==y x!==y x!=y

COMPARE x<y x<=y x>y x>=y

CONCAT x::y

SHIFT x<<y x>>y

ADD_SUB x+y x-y

MUL_DIV x*y x/y x xy

PREFIX @x !x ~x +x -x ++x --x

POSTFIX x() x.y x[y] x{y} x++ x--

DEFINITION function { }

Definition at line 228 of file PikaScript.h.

6.1.4 Function Documentation

6.1.4.1 template<class C , class A0 , class R > bound_mem_fun_t<C, A0, R> Pika::bound_mem_fun (R(C::*)(A0) *m,* C * *o*) [inline]

bound_mem_fun creates a member functor bound to a specific C++ object through a pointer.

You may use this function instead of "manually" binding a std::mem_fun functor to an object. For example the following code:

```
std::bind1st(std::mem_fun(&Dancer::tapDance), fredAstaire)));
```

can be replaced with

```
bound_mem_fun(&Dancer::tapDance, fredAstaire);
```

Furthermore, bound_mem_fun does not suffer from a problem that some STL implementations has which prevents you from using member functors with reference arguments.

bound_mem_fun is used in PikaScript to directly bind a native function to a member function of a certain C++ object.

Definition at line 119 of file PikaScript.h.

6.1.4.2 template<class S > S Pika::doubleToString (double *d,* int *precision* = 14)

Converts the double `d` to a string (in scientific e notation, e.g. -12.34e-3).

`precision` can be between 1 and 24 and is the number of digits to include in the output string (not counting any exponent of course). Any trailing decimal zeroes will be trimmed and only significant digits will be included.

Definition at line 148 of file PikaScriptImpl.h.

6.1.4.3 template<class S > S Pika::escape (const S & *s*)

Depending on the contents of the source string `s` it is encoded either in single (' ') or double (" ") quotes.

If the string contains only printable ASCII chars (ASCII values between 32 and 126 inclusively) and no apostrophes (' '), it is enclosed in single quotes with no further processing. Otherwise it is enclosed in double quotes (" ") and any unprintable ASCII character, backslash (\) or quotation mark (") is encoded using C-style escape sequences (e.g. \code "line1 ").

Definition at line 217 of file PikaScriptImpl.h.

6.1.4.4 template<class S > ulong Pika::hexToLong (typename S::const_iterator & *p,* const typename S::const_iterator & *e*)

Converts a string in hexadecimal form to an ulong integer.

p is updated on return to point to the first unparsed (e.g. invalid) character. If p == e, the full string was successfully converted.

Definition at line 95 of file PikaScriptImpl.h.

6.1.4.5 template<class S , typename T > S Pika::intToString (T *i*, int *radix* = 10, int *minLength* = 1)

Converts the integer i to a string with a radix and minimum length of your choice.

radix can be anything between 1 (binary) and 16 (hexadecimal).

Definition at line 111 of file PikaScriptImpl.h.

6.1.4.6 template<class S > double Pika::stringToDouble (typename S::const_iterator & *p*, const typename S::const_iterator & *e*)

Converts a string in scientific e notation (e.g. -12.34e-3) to a double floating point value.

Spaces before 'e' are not accepted. Uppercase 'E' is allowed. Positive and negative 'infinity' is supported (provided the compiler allows it).p is updated on return to point to the first unparsed (e.g. invalid) character. If p == e, the full string was successfully converted.

Definition at line 123 of file PikaScriptImpl.h.

6.1.4.7 template<class S > long Pika::stringToLong (typename S::const_iterator & *p*, const typename S::const_iterator & *e*)

Converts a string in decimal form to a signed long integer.

p is updated on return to point to the first unparsed (e.g. invalid) character. If p == e, the full string was successfully converted.

Definition at line 103 of file PikaScriptImpl.h.

6.1.4.8 template<class S > std::string Pika::toStdString (const S & *s*)

Converts the string s to a standard C++ string.

The default implementation is std::string(s.begin(), s.end()). You should specialize this template if necessary.

Definition at line 91 of file PikaScriptImpl.h.

6.1.4.9 template<class S > S Pika::unescape (typename S::const_iterator & *p*, const typename S::const_iterator & *e*)

Converts a string that is either enclosed in single (' ') or double (" ") quotes.

The routine expects the string beginning at `p` to be of one of these forms, but `e` can point beyond the terminating quote. If the single (' ') quote is used, the string between the quotes is simply extracted "as is" with the exception of pairs of apostrophes (") that are used to represent single apostrophes. If the string is enclosed in double quotes (" ") it can use C-style escape sequences. The supported escape sequences are:

```
\\ \" \' \a \b \f \n \r \t \v \xHH \uHHHH \<decimal>
```

. On return, `p` will point to the first unparsed (e.g. invalid) character. If `p` == `e`, the full string was successfully converted.

Definition at line 190 of file PikaScriptImpl.h.

Chapter 7

Class Documentation

7.1 QStrings::QString< C, PS >::_iterator< E, Q > Class Template Reference

7.1.1 Detailed Description

template<typename C, size_t PS = (64 - 12)>template<typename E, class Q> class QStrings::QString< C, PS >::_iterator< E, Q >

Definition at line 179 of file QStrings.h.

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

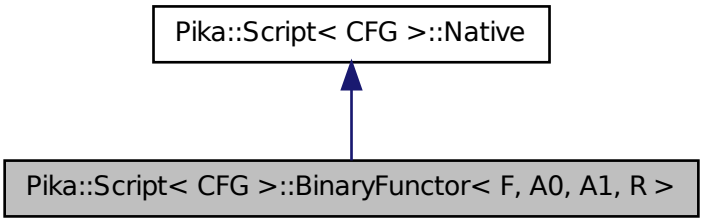
- /Users/Magnus/projects/PikaScript/src/QStrings.h

7.2 Pika::Script< CFG >::BinaryFunctor< F, A0, A1, R > Class Template Reference

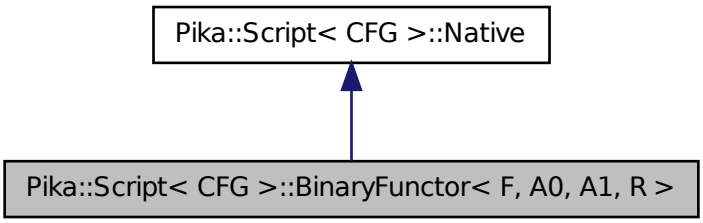
See UnaryFunctor for documentation.

`#include <PikaScript.h>`

Inheritance diagram for Pika::Script< CFG >::BinaryFunctor< F, A0, A1, R >:



Collaboration diagram for Pika::Script< CFG >::BinaryFunctor< F, A0, A1, R >:



Protected Member Functions

- template<class T >
  Value call (const Value &a0, const Value &a1, const T &)

  Call the functor with the arguments `a0` and `a1`.

- Value call (const Value &a0, const Value &a1, const Dumb< void > &)

  Overloaded to return the `void` value for functors that returns the `void` type.

7.2.1 Detailed Description

template<class CFG>template<class F, class A0 = typename F::first_argument_type, class A1 = typename F::second_argument_type, class R = typename F::result_type> class Pika::Script< CFG >::BinaryFunctor< F, A0, A1, R >

See UnaryFunctor for documentation.

Definition at line 500 of file PikaScript.h.

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

- /Users/Magnus/projects/PikaScript/src/PikaScript.h

7.3 Pika::bound_mem_fun_t< C, A0, R > Class Template Reference

bound_mem_fun_t is a member functor bound to a specific C++ object through a pointer.

`#include <PikaScript.h>`

7.3.1 Detailed Description

template<class C, class A0, class R> class Pika::bound_mem_fun_t< C, A0, R >

bound_mem_fun_t is a member functor bound to a specific C++ object through a pointer. You may use this class instead of "manually" binding a std::mem_fun functor to an object. Besides being more convenient, this class solves a problem in some STL implementations that prevents you from having reference arguments in the functor.

You would normally use the helper function bound_mem_fun() to automatically instantiate the correct template.

Definition at line 101 of file PikaScript.h.

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

- /Users/Magnus/projects/PikaScript/src/PikaScript.h

7.4 QStrings::QString< C, PS >::Buffer Class Reference

7.4.1 Detailed Description

template<typename C, size_t PS = (64 - 12)> class QStrings::QString< C, PS >::Buffer

Definition at line 133 of file QStrings.h.

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

- /Users/Magnus/projects/PikaScript/src/QStrings.h

7.5 Pika::Dumb< T > Class Template Reference

We use this dummy class to specialize member functions for arbitrary types (including void, references etc).

`#include <PikaScript.h>`

7.5.1 Detailed Description

template<class T> class Pika::Dumb< T >

We use this dummy class to specialize member functions for arbitrary types (including void, references etc).

Definition at line 126 of file PikaScript.h.

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

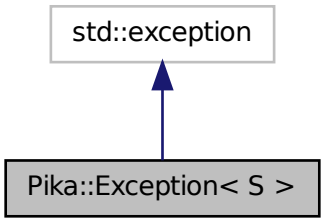
- /Users/Magnus/projects/PikaScript/src/PikaScript.h

7.6 Pika::Exception< S > Class Template Reference

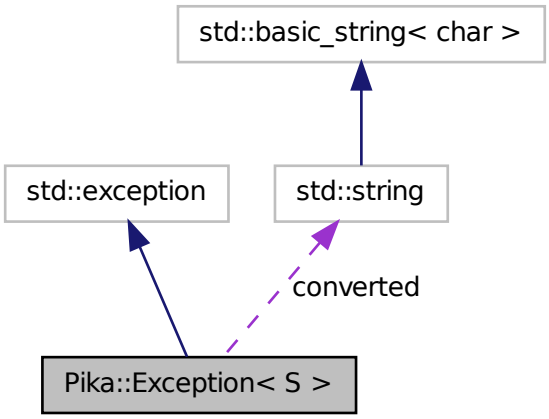
The PikaScript exception class.

`#include <PikaScript.h>`

Inheritance diagram for Pika::Exception< S >:



Collaboration diagram for Pika::Exception< S >:



Public Member Functions

- Exception (const S &error)

  Simply constructs the exception with error string `error`.

- virtual S getError () const throw ()

  Return the error string for this exception.

- virtual const char * what () const throw ()

  Returns the error as a null-terminated char string.

Protected Attributes

- S error

  The error string.

- std::string converted

  Since what() is defined to return a pointer only, we need storage for the converted string within this class too.

7.6.1 Detailed Description

template<class S> class Pika::Exception< S >

The PikaScript exception class. It is based on `std::exception` and stores a simple error string of type `S`. The standard what() is provided, and therefore conversion to a `const char*` string must be performed in case the `S` class is not directly compatible. Since what() is defined to return a pointer only, we need storage for the converted string within this class too (Exception::converted).

Definition at line 134 of file PikaScript.h.

7.6.2 Member Function Documentation

7.6.2.1 template<class S> virtual const char* Pika::Exception< S >::what ( ) const throw () [inline, virtual]

Returns the error as a null-terminated char string.

Use getError() if you can since it is faster and permits other character types than char.

Definition at line 137 of file PikaScript.h.

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

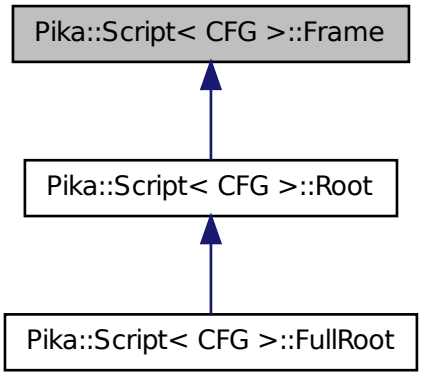
- /Users/Magnus/projects/PikaScript/src/PikaScript.h

7.7 Pika::Script< CFG >::Frame Class Reference

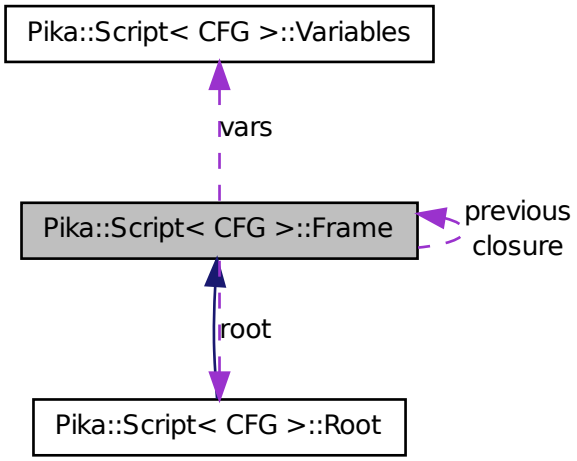
The execution context and interpreter for PikaScript.

`#include <PikaScript.h>`

Inheritance diagram for Pika::Script< CFG >::Frame:



Collaboration diagram for Pika::Script< CFG >::Frame:



Protected Types

- typedef std::pair< bool, Value > XValue

  The XValue differentiates lvalues and rvalues and is used internally in the interpreter.

Construction.

- Frame (Variables &vars, Root &root, Frame *previous)

  Constructs the Frame and associates it with the variable space `vars`.

Properties.

- Variables & getVariables () const throw ()

  Returns a reference to the Variable instance associated with this Frame. Simple as that.

- Root & getRoot () const throw ()

  Returns a reference to the "root frame" for this Frame. (No brainer.)

- Frame & getPrevious () const throw ()

  Returns a reference to the previous frame (i.e. the frame of the caller of this frame). Must not be called on the root frame (will assert).

Getting, setting and referencing variables.

- Value get (const String &identifier, bool fallback=false) const

  Gets a variable value.

- Value getOptional (const String &identifier, const Value &defaultValue=Value()) const

  Tries to get the variable value as with get() (but never "falls back").

- const Value & set (const String &identifier, const Value &v)

  Sets a variable value.

- Value reference (const String &identifier) const

  Creates a reference to the variable identified by `identifier` *by prefixing it with a "frame label".*

- std::pair< Frame *, String > resolveFrame (const String &identifier) const

  Resolves the frame for `identifier` *and returns it together with* `identifier` *stripped of any prefixed "frame identifier".*

Calling functions and evaluating source code.

- Value call (const String &callee, const Value &body, long argc, const Value *argv=0)

  Calls a Pika function (by setting up a new "sub-frame" and executing the function body).

- Value execute (const String &body)

  A low-level function that executes `body` directly on the Frame instance.

- Value evaluate (const String source)

  Evaluates the PikaScript expression in `source` directly on this Frame.

- StringIt parse (const StringIt &begin, const StringIt &end, bool literal)

  Parses a PikaScript expression or literal (without evaluating it) and returns an iterator pointing at the end of the expression.

Registering native functions (or objects).

- void registerNative (const String &identifier, Native *native)

  Registers the native function (or object) `native` with `identifier` in the appropriate variable space (determined by any "frame identifier" present in `identifier`).

- template<class A0 , class R >
  void registerNative (const String &i, R(*f)(A0))

  Helper template for easily registering a unary C++ function.

- template<class A0 , class A1 , class R >
  void registerNative (const String &i, R(*f)(A0, A1))

  Helper template for easily registering a binary C++ function.

- template<class C , class A0 , class R >
  void registerNative (const String &i, C *o, R(C::*m)(A0))

  Helper template for easily registering a unary C++ member function in the C++ object pointed to by o.

- void unregisterNative (const String &identifier)

  Helper function for unregistering a native function / object.

Destruction.

- virtual ~Frame ()

  The default destructor does nothing, but it is always good practice to have a virtual destructor.

7.7.1 Detailed Description

template<class CFG> class Pika::Script< CFG >::Frame

The execution context and interpreter for PikaScript. This is where the magic happens. A Frame represents an execution context for a PikaScript function and it contains the source code interpreter. Normally you do not create instances of Frame yourself. They are created on stack whenever a function call is made. Notice that this implementation of PikaScript does not run in a virtual machine, instead it is interpreted directly and it shares calling stack etc with your C++ application.

Definition at line 311 of file PikaScript.h.

7.7.2 Member Typedef Documentation

7.7.2.1 template<class CFG > typedef std::pair<bool, Value> Pika::Script< CFG >::Frame::XValue [protected]

The XValue differentiates lvalues and rvalues and is used internally in the interpreter.

first = lvalue or not, second = symbol (for lvalue) or actual value (for rvalue).

Definition at line 356 of file PikaScript.h.

7.7.3 Constructor & Destructor Documentation

7.7.3.1 template<class CFG > TMPL Pika::Script< CFG >::Frame::Frame (Variables & *vars*, Root & *root*, Frame * *previous*)

Constructs the Frame and associates it with the variable space `vars`.

All frames on the calling stack have direct access to the "root frame" which is designated by `root` (will be = `*this` for the Root). `previous` should point to the caller Frame (or 0 for the Root). The "frame label" of a root frame is always :: . Root::generateLabel() is called to create unique labels for other frames.

Definition at line 286 of file PikaScriptImpl.h.

7.7.4 Member Function Documentation

7.7.4.1 template<class CFG > Value Pika::Script< CFG >::Frame::call (const String & *callee*, const Value & *body*, long *argc*, const Value * *argv* = `0`)

Calls a Pika function (by setting up a new "sub-frame" and executing the function body).

You may pass `Value()` (the `void` value) for `callee` or `body`. If only `callee` is specified, it will be used to retrieve the function body (through get()). If only `body` is specified, the called function will not have a `$callee` variable (`$callee` is used for debugging and object-oriented solutions). If both are present, the `$callee` variable

will be set to `callee`, and `body` will be executed. `argc` is the number of arguments and if this is not zero, the `argv` parameter should point to an array of arguments (of at least `argc` elements in size). The return value is that of the PikaScript function.

7.7.4.2 template<class CFG > Value Pika::Script< CFG >::Frame::evaluate (const String *source*)

Evaluates the PikaScript expression in `source` directly on this Frame.

This differs from execute() in that `source` is not expected to be in the format of a "function body" of an "ordinary", "lambda" or "native" function. (Notice that we are not passing a reference to the `source` string here so that we are safe in case the PikaScript code manipulates the very string it is running on.) The return value is that of the evaluated expression.

7.7.4.3 template<class CFG > Value Pika::Script< CFG >::Frame::execute (const String & *body*)

A low-level function that executes `body` directly on the Frame instance.

This means that unlike call(), you need to setup a "sub-frame" yourself, populate it with argument variables and then use this function. The return value is that of the PikaScript function.

7.7.4.4 template<class CFG > Value Pika::Script< CFG >::Frame::get (const String & *identifier*, bool *fallback* = `false`) const

Gets a variable value.

If `identifier` is prefixed with a "frame identifier" (e.g. a "frame label" or ^), it will be "resolved" and used for retrieving the variable. Otherwise the variable space associated with this Frame instance will be checked and if the variable cannot be found and `fallback` is true, the global variable space will also be checked. If the variable cannot be found in any of the checked locations, an exception will be thrown.

7.7.4.5 template<class CFG > Value Pika::Script< CFG >::Frame::getOptional (const String & *identifier*, const Value & *defaultValue* = Value `()`) const

Tries to get the variable value as with get() (but never "falls back").

If the variable cannot be found, `defaultValue` will be returned instead.

7.7.4.6 template<class CFG > Value Pika::Script< CFG >::Frame::reference (const String & *identifier*) const

Creates a reference to the variable identified by `identifier` by prefixing it with a "frame label".

If the identifier is already prefixed with a "frame identifier" (such as ^) it will be resolved to determine the frame. Otherwise, the label of this Frame instance is used.

7.7.4.7 template<class CFG > template<class C , class A0 , class R > void Pika::Script< CFG >::Frame::registerNative (const String & *i*, C * *o*, R(C::*)(A0) *m*) [inline]

Helper template for easily registering a unary C++ member function in the C++ object pointed to by `o`.

The C++ member function should take a single argument of either Frame& or any of the native types that are convertible from Script::Value. It should return a value of any type that is convertible to Script::Value or void. Normally, this registration technique is used for bridging Pika function calls to methods of a C++ object which is guaranteed to live as long as the target Frame.

Definition at line 347 of file PikaScript.h.

7.7.4.8 template<class CFG > template<class A0 , class A1 , class R > void Pika::Script< CFG >::Frame::registerNative (const String & *i*, R(*)(A0, A1) *f*) [inline]

Helper template for easily registering a binary C++ function.

The C++ function should take two arguments of any of the native types that are convertible from Script::Value. It should return a value of any type that is convertible to Script::Value or void.

Definition at line 344 of file PikaScript.h.

7.7.4.9 template<class CFG > template<class A0 , class R > void Pika::Script< CFG >::Frame::registerNative (const String & *i*, R(*)(A0) *f*) [inline]

Helper template for easily registering a unary C++ function.

The C++ function should take a single argument of either Frame& or any of the native types that are convertible from Script::Value. It should return a value of any type that is convertible to Script::Value or void.

Definition at line 341 of file PikaScript.h.

7.7.4.10 template<class CFG > TMPL void Pika::Script< CFG >::Frame::registerNative (const String & *identifier*, Native * *native*)

Registers the native function (or object) `native` with `identifier` in the appropriate variable space (determined by any "frame identifier" present in `identifier`).

Once registered, the native is considered "owned" by the variable space. In other words, all registered natives will be deleted by the Variables destructor. Also, if you register a new native on an already used identifier, the old native for that identifier will be

deleted automatically. Besides assigning the native with Variables::assignNative() this method also sets the variable `identifier` to <identifier> (unless `native` is a null-pointer).

Definition at line 767 of file PikaScriptImpl.h.

7.7.4.11 template<class CFG > std::pair< Frame*, String > Pika::Script< CFG >::Frame::resolveFrame (const String & *identifier*) const

Resolves the frame for `identifier` and returns it together with `identifier` stripped of any prefixed "frame identifier".

The rules are as follows: 1) If the identifier has a leading ::, the "root frame" is returned. 2) If the identifier begins with an existing frame label, this frame is used for resolving the rest of the identifier. (If a frame label cannot be found an exception is thrown.) 3) For each leading '^' the previous frame is used for resolving the rest of the identifier. 4) Finally, if the identifier does not begin with the '$' character, the "closure" of the current frame is returned, otherwise the current frame is returned.

7.7.4.12 template<class CFG > const Value& Pika::Script< CFG >::Frame::set (const String & *identifier,* const Value & *v*)

Sets a variable value.

Just as with get(), `identifier` may be prefixed with a "frame identifier" to address a different Frame.

7.7.4.13 template<class CFG > void Pika::Script< CFG >::Frame::unregisterNative (const String & *identifier*) [inline]

Helper function for unregistering a native function / object.

Unregistering a native is the same as registering a null-pointer to the identifier. Any PikaScript variable referring to <identifier> will still do so (including the one created automatically by registerNative()). However, performing a function call on such a variable will generate a run-time exception.

Definition at line 350 of file PikaScript.h.

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

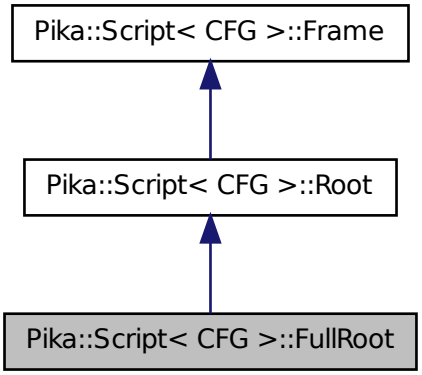
- /Users/Magnus/projects/PikaScript/src/PikaScript.h
- /Users/Magnus/projects/PikaScript/src/PikaScriptImpl.h

7.8 Pika::Script< CFG >::FullRoot Class Reference

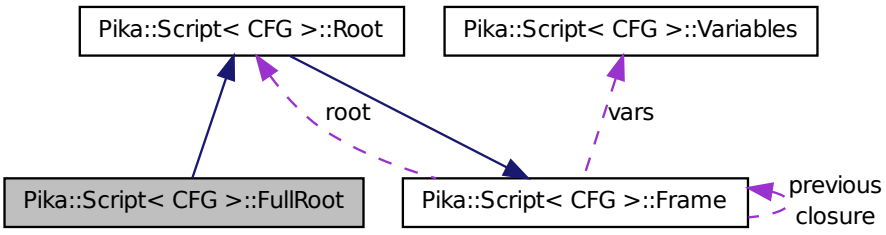
FullRoot inherits from both Root and CFG::Globals (which should be a descendant to Variable).

`#include <PikaScript.h>`

Inheritance diagram for Pika::Script< CFG >::FullRoot:



Collaboration diagram for Pika::Script< CFG >::FullRoot:



Public Member Functions

- FullRoot (bool includeIONatives=true)

7.8.1 Detailed Description

template<class CFG> class Pika::Script< CFG >::FullRoot

FullRoot inherits from both Root and CFG::Globals (which should be a descendant to Variable). Its constructor adds the natives of the standard library. This means that by instantiating this class you will get a full execution environment for PikaScript ready to go.

Definition at line 422 of file PikaScript.h.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 template<class CFG > Pika::Script< CFG >::FullRoot::FullRoot (bool *includeIONatives* = `true`) [inline]

< If `includeIO` is false, 'load', 'save', 'input', 'print' and 'system' will not be registered.

Definition at line 423 of file PikaScript.h.

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

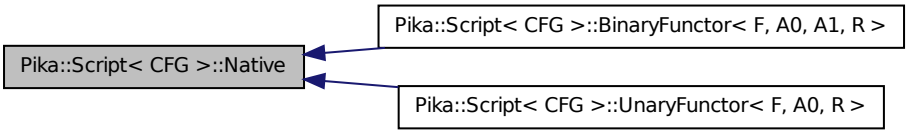
- /Users/Magnus/projects/PikaScript/src/PikaScript.h

7.9 Pika::Script< CFG >::Native Class Reference

Native is the base class for the native functions and objects that can be accessed from PikaScript.

`#include <PikaScript.h>`

Inheritance diagram for Pika::Script< CFG >::Native:



Public Member Functions

- virtual Value pikaCall (Frame &)

  Process the PikaScript call.

7.9.1 Detailed Description

template<class CFG> class Pika::Script< CFG >::Native

Native is the base class for the native functions and objects that can be accessed from PikaScript. It has a single virtual member function which should process a call to the native. Since natives are owned by the variable space once they are registered (and destroyed when the variable space destructs), they often act as simple bridges to other C++ functions and objects.

The easiest way to register a native is by calling one of the Frame::registerNative() member functions (typically on the "root frame"). You will find a couple of template functions there that allows you to register functors directly. They will create the necessary Native classes for you in the background.

Definition at line 459 of file PikaScript.h.

7.9.2 Member Function Documentation

7.9.2.1 template<class CFG > virtual Value Pika::Script< CFG >::Native::pikaCall (Frame &) [inline, virtual]

Process the PikaScript call.

Arguments can be retrieved by getting `$0`, `$1` etc from `frame` (via Frame::get() or Frame::getOptional()). If the function should not return a value, return `Value()` (which constructs a `void` value).

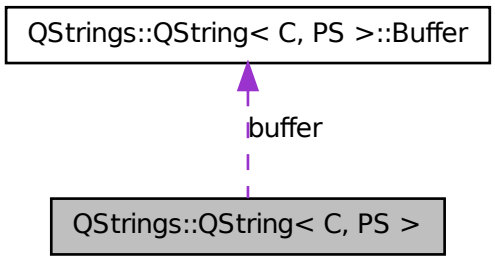
Definition at line 460 of file PikaScript.h.

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

- /Users/Magnus/projects/PikaScript/src/PikaScript.h
- /Users/Magnus/projects/PikaScript/src/PikaScriptImpl.h

7.10 QStrings::QString< C, PS > Class Template Reference

Collaboration diagram for QStrings::QString< C, PS >:



Classes

- class _iterator
- class Buffer

7.10.1 Detailed Description

template<typename C, size_t PS = (64 - 12)> class QStrings::QString< C, PS >

Definition at line 65 of file QStrings.h.

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

- /Users/Magnus/projects/PikaScript/src/QStrings.h

7.11 Pika::QuickVars< Super, CACHE_SIZE > Class Template Reference

7.11.1 Detailed Description

template<class Super, unsigned int CACHE_SIZE = 11> class Pika::QuickVars< Super, CACHE_SIZE >

Definition at line 52 of file QuickVars.h.

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

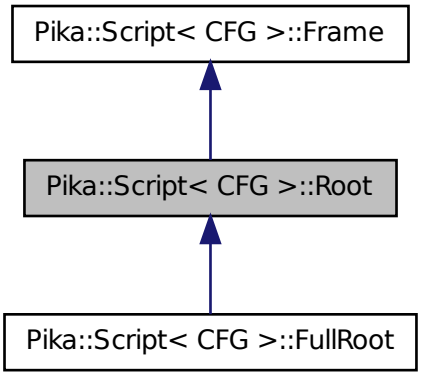
- /Users/Magnus/projects/PikaScript/src/QuickVars.h

7.12 Pika::Script< CFG >::Root Class Reference

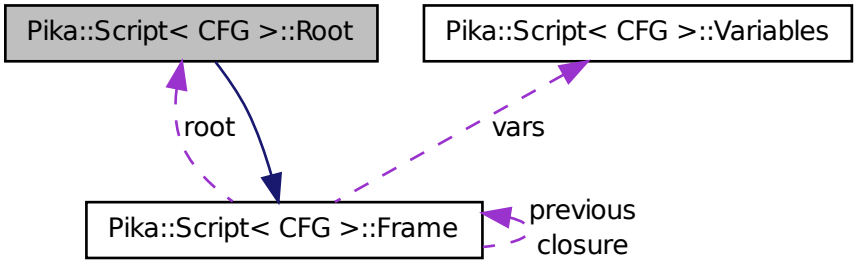
The Root is the first Frame you instantiate.

`#include <PikaScript.h>`

Inheritance diagram for Pika::Script< CFG >::Root:



Collaboration diagram for Pika::Script< CFG >::Root:



Public Member Functions

- virtual void trace (Frame &frame, const String &source, SizeType offset, bool lvalue, const Value &value, Precedence level, bool exit)

  Overload this member function if you want to customize the tracing mechanism in PikaScript.

- virtual void setTracer (Precedence traceLevel, const Value &tracerFunction) throw ()

  Called by the standard library function "trace" to assign a PikaScript tracer function and a trace level. (Also called by the standard trace() on exceptions.)

- bool doTrace (Precedence level) const throw ()
- String generateLabel ()

  Each "sub-frame" requires a unique "frame label".

Protected Attributes

- Precedence traceLevel

  Calls to trace() will only happen when the "precedence level" is less or equal to this.

- Value tracerFunction

  Pika-script tracer function (used by the default trace() implementation).

- bool isInsideTracer

  Set to prevent recursive calling of tracer (used by the default trace() implementation).

- Char autoLabel [32]

  The last generated frame label (padded with leading ':').

- Char * autoLabelStart

  The first character of the last generated frame label (begins at autoLabel + 30 and slowly moves backwards when necessary).

7.12.1 Detailed Description

template<class CFG> class Pika::Script< CFG >::Root

The Root is the first Frame you instantiate. It is the starting point for the execution of PikaScript code. Its variables can be accessed from any frame with the special "frame identifier" ::. Furthermore, its variable space is often checked as a "backup" for symbols that cannot be retrieved from local "sub-frames".

The class also offers a few functions out of which you may overload trace() and setTracer() if you want to customize the tracing mechanism in PikaScript. The default

implementation calls a PikaScript function that you can designate with the standard library function "trace".

In case you use PikaScript concurrently in different threads, you need a Root for every thread, but you could implement and share a sub-class of Variables that accesses shared data in a thread-safe manner.

If you just want to use the standard Root implementation with a standard variable space you may want to use FullRoot instead.

Definition at line 403 of file PikaScript.h.

7.12.2 Member Function Documentation

7.12.2.1 template<class CFG > bool Pika::Script< CFG >::Root::doTrace (Precedence *level*) const throw () [inline]

This function is called ∗a lot∗. For performance reasons it is good if it becomes inlined, so we are not declaring it virtual. If you want to customize which events that will be traced, try cleverly implementing your own trace() and setTracer() member functions instead.

Definition at line 408 of file PikaScript.h.

7.12.2.2 template<class CFG > String Pika::Script< CFG >::Root::generateLabel ( )

Each "sub-frame" requires a unique "frame label".

This function creates it by "incrementing" Root::autoLabel, character by character, using '0' to '9' and upper and lower case 'a' to 'z', growing the string when necessary.

7.12.2.3 template<class CFG > TMPL void Pika::Script< CFG >::Root::setTracer (Precedence *traceLevel,* const Value & *tracerFunction*) throw () [virtual]

Called by the standard library function "trace" to assign a PikaScript tracer function and a trace level. (Also called by the standard trace() on exceptions.)

You may want to overload this member function if you change the tracing mechanism and need control over the trace level for example.

Definition at line 795 of file PikaScriptImpl.h.

7.12.2.4 template<class CFG > TMPL void Pika::Script< CFG >::Root::trace (Frame & *frame,* const String & *source,* SizeType *offset,* bool *lvalue,* const Value & *value,* Precedence *level,* bool *exit*) [virtual]

Overload this member function if you want to customize the tracing mechanism in PikaScript.

The default implementation calls the PikaScript function Root::tracerFunction that you can assign with the standard library function "trace". See the standard library documentation on "trace" for more information on the arguments to this member function.

Definition at line 800 of file PikaScriptImpl.h.

7.12.3 Member Data Documentation

7.12.3.1 template<class CFG > Precedence Pika::Script< CFG >::Root::traceLevel [protected]

Calls to trace() will only happen when the "precedence level" is less or equal to this.

E.g. if traceLevel is CALL, only function calls and caught exceptions will be traced.

Definition at line 410 of file PikaScript.h.

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

- /Users/Magnus/projects/PikaScript/src/PikaScript.h
- /Users/Magnus/projects/PikaScript/src/PikaScriptImpl.h

7.13 Pika::Script< CFG > Struct Template Reference

Script is a meta-class that groups all the core classes of the PikaScript interpreter together (except for the value class).

`#include <PikaScript.h>`

Classes

- class BinaryFunctor

  See UnaryFunctor for documentation.

- class Frame

  The execution context and interpreter for PikaScript.

- class FullRoot

  FullRoot inherits from both Root and CFG::Globals (which should be a descendant to Variable).

- class Native

  Native is the base class for the native functions and objects that can be accessed from PikaScript.

- class Root

  The Root is the first Frame you instantiate.

- class STLVariables

  STLVariables is the reference implementation of a variable space.

- class UnaryFunctor

  We provide two Native template classes for bridging PikaScript calls to C++ "functors".

- class Variables

  Variables is an abstract base class which implements the interface to the variable space that a Frame works on.

Public Types

- typedef CFG Config

  The configuration meta-class. E.g. StdConfig.

- typedef CFG::Value Value

  The class used for all values and variables (defined by the configuration meta-class). E.g. STLValue.

- typedef Value::String String

  The class used for strings (defined by the string class). E.g. `std::string`.

- typedef String::value_type Char

  The character type for all strings (defined by the string class). E.g. `char`.

- typedef String::size_type SizeType

  The length type for all strings (defined by the string class). E.g. `size_t`.

- typedef String::const_iterator StringIt

  The const_iterator of the string is used so frequently it deserves its own typedef.

- typedef Exception< String > Xception

  The exception type.

Static Public Member Functions

- template<class F >
  static UnaryFunctor< F > * newUnaryFunctor (const F &f)

  Helper function to create a UnaryFunctor class with correct template parameters.

- template<class F >
  static BinaryFunctor< F > * newBinaryFunctor (const F &f)

  Helper function to create a BinaryFunctor class with correct template parameters.

- static std::pair< Value, String > getThisAndMethod (Frame &frame)

  getThisAndMethod splits the `$callee` variable of `frame` into object ("this") and method.

- static Value elevate (Frame &frame)

  Used to "aggregate" different method calls into a single function call.

- static Value getThis (Frame &frame)

  Returns only the "this" value as descripted in getThisAndMethod().

- static Value getMethod (Frame &frame)

  Returns only the "method" value as descripted in getThisAndMethod().

- static void addStandardNatives (Frame &frame, bool includeIO=true)

  Registers the standard native functions to `frame`. If `includeIO` is false, 'load', 'save', 'input', 'print' and 'system' will not be registered. Please, refer to the PikaScript standard library reference guide for more info on individual native functions.

7.13.1 Detailed Description

template<class CFG> struct Pika::Script< CFG >

Script is a meta-class that groups all the core classes of the PikaScript interpreter together (except for the value class). The benefit of having a class like this is that we can declare types that are common to all sub-classes.

The class is a template that takes another meta-class for configuring PikaScript. The configuration class should contain the following typedefs:

1. `Value` (use this class for all PikaScript values, e.g. STLValue<std::string>)
2. `Locals` (when a function call occurs, this sub-class of Variables will be instantiated for the callee)
3. `Globals` (this sub-class of Variables is used for the FullRoot class)

Definition at line 265 of file PikaScript.h.

7.13.2 Member Function Documentation

7.13.2.1 template<class CFG > TMPL std::pair< T_TYPE(Value), T_TYPE(String)> Pika::Script< CFG >::getThisAndMethod (Frame & *frame*) [static]

getThisAndMethod splits the `$callee` variable of `frame` into object ("this") and method.

The returned value is a pair, where the `first` value ("this") is a reference to the object and the `second` value is the "method" name as a string.

Notice that if the $callee variable does not begin with a "frame specifier", it is assumed that the object belongs to the previous frame (e.g. the caller of the method). This holds true even if the method is actually defined in the root frame. For example

```
function { obj.meth() }
```

would trigger an error even if `::obj` is defined since `obj` isn't defined in our function. While

```
function { ::obj.meth() }
```

works.

One common use for this function is in a PikaScript object constructor for extracting the "this" reference that should be constructed. Another situation where this routine is useful is if you use the "elevate" function to aggregate various methods into a single C++ function. You may then use this function to extract the method name.

Definition at line 858 of file PikaScriptImpl.h.

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

- /Users/Magnus/projects/PikaScript/src/PikaScript.h
- /Users/Magnus/projects/PikaScript/src/PikaScriptImpl.h

7.14 Pika::StdConfig Struct Reference

StdConfig is a configuration class for Script that uses the reference implementations of STLValue and Script::STLVariables.

`#include <PikaScript.h>`

7.14.1 Detailed Description

StdConfig is a configuration class for Script that uses the reference implementations of STLValue and Script::STLVariables.

Definition at line 571 of file PikaScript.h.

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

- /Users/Magnus/projects/PikaScript/src/PikaScript.h

7.15 Pika::STLValue< S > Class Template Reference

STLValue is the reference implementation of a PikaScript variable.

`#include <PikaScript.h>`

Typedefs.

- typedef S String

  The class to use for all strings (i.e. the super-class).

- typedef S::value_type Char

  The character type for all strings (e.g. char or wchar_t).

Constructors.

- STLValue ()

  The default constructor initializes the value to the empty string (or "void").

- STLValue (double d)

  Constructs a value representing the double precision floating point `d`.

- STLValue (float f)

  Constructs a value representing the single precision floating point `f`.

- STLValue (long i)

  Constructs a value representing the signed long integer `l`.

- STLValue (ulong i)

  Constructs a value representing the ulong integer `l`.

- STLValue (int i)

  Constructs a value representing the signed integer `i`.

- STLValue (uint i)

  Constructs a value representing the unsigned integer `i`.

- STLValue (bool b)

  Constructs a value representing the boolean `b`.

- template<class T >
  STLValue (const T &s)

  Pass other types of construction onwards to the super-class `S`.

Conversion to native C++ types.

- operator bool () const

  Converts the value to a boolean.

- operator long () const

Converts the value to a signed long integer.

- operator double () const

  Converts the value to a double precision floating point.

- operator float () const

  Converts the value to a single precision floating point.

- operator ulong () const

  Converts the value to an ulong integer.

- operator int () const

  Converts the value to a signed integer.

- operator uint () const

  Converts the value to an unsigned integer.

Overloaded operators (comparisons and subscript).

- bool operator< (const STLValue &r) const

  Less than comparison operator.

- bool operator== (const STLValue &r) const

  Equality operator.

- bool operator!= (const STLValue &r) const

  Non-equality operator.

- bool operator> (const STLValue &r) const

  Greater than comparison operator.

- bool operator<= (const STLValue &r) const

  Less than or equal to comparison operator.

- bool operator>= (const STLValue &r) const

  Greater than or equal to comparison operator.

- const STLValue operator[ ] (const STLValue &i) const

  The subscript operator returns the concatenation of the value with the dot (.) separator (if necessary) and the value `i`.

Classification methods.

- bool isVoid () const

  Returns true if the value represents the empty string.

Helper templates to allow certain operations on any type that is convertible to a STLValue.

- template<class T >
  bool **operator<** (const T &r) const
- template<class T >
  bool **operator==** (const T &r) const
- template<class T >
  bool **operator!=** (const T &r) const
- template<class T >
  bool **operator>** (const T &r) const
- template<class T >
  bool **operator<=** (const T &r) const
- template<class T >
  bool **operator>=** (const T &r) const
- template<class T >
  const STLValue **operator[ ]** (const T &i) const

7.15.1 Detailed Description

template<class S> class Pika::STLValue< S >

STLValue is the reference implementation of a PikaScript variable. Internally, all values are represented by an STL compliant string class `S *`. This class actually inherits from `S` (with public inheritance) for optimization reasons. This way we avoid a lot of unnecessary temporary objects when we cast to and from strings. (Unfortunately it is not possible to make this inheritance private and add conversion operators to the `S` class. Explicit conversion operators in C++ have lower priority than implicit base class conversions.)

Although it may seem inefficient to store all variables in textual representation it makes PikaScript easy to interface with and debug for. With the custom value <-> text conversion routines in PikaScript the performance isn't too bad. It mainly depends on the performance of the string implementation which is the reason why this class is a template. The standard variant of STLValue uses std::string, but you may want to "plug in" a more efficient class.

STLValue supports construction from and casting to the following C++ types:

- `bool`
- `int`

- `uint`
- `long`
- `ulong`
- `float`
- `double`
- the template string class `S`

Note

PikaScript only requires a specific subset of the STL string features. Utility member functions (such as `find_first_of`) are never used (instead, the equivalent generic STL algorithms are utilized). Destructive functions such as `insert` and `erase` are not used either. Furthermore, PikaScript consider all string access through the subscript operator [] to be for reading only (therefore only a `const` function for this operator is required).

Definition at line 172 of file PikaScript.h.

7.15.2 Member Function Documentation

7.15.2.1 template<class S > Pika::STLValue< S >::operator bool ( ) const

Converts the value to a boolean.

If the value isn't "true" or "false" an exception is thrown.

Definition at line 243 of file PikaScriptImpl.h.

7.15.2.2 template<class S > Pika::STLValue< S >::operator double ( ) const

Converts the value to a double precision floating point.

If the value isn't in valid floating point format an exception is thrown.

Definition at line 256 of file PikaScriptImpl.h.

7.15.2.3 template<class S > Pika::STLValue< S >::operator float ( ) const [inline]

Converts the value to a single precision floating point.

If the value isn't in valid floating point format an exception is thrown.

Definition at line 196 of file PikaScript.h.

7.15.2.4 template<class S > Pika::STLValue< S >::operator int ( ) const [inline]

Converts the value to a signed integer.

If the value isn't in valid integer format an exception is thrown.

Definition at line 198 of file PikaScript.h.

7.15.2.5 template<class S > Pika::STLValue< S >::operator long ( ) const

Converts the value to a signed long integer.

If the value isn't in valid integer format an exception is thrown.

Definition at line 249 of file PikaScriptImpl.h.

7.15.2.6 template<class S > Pika::STLValue< S >::operator uint ( ) const [inline]

Converts the value to an unsigned integer.

If the value isn't in valid integer format an exception is thrown.

Definition at line 199 of file PikaScript.h.

7.15.2.7 template<class S > Pika::STLValue< S >::operator ulong ( ) const [inline]

Converts the value to an ulong integer.

If the value isn't in valid integer format an exception is thrown.

Definition at line 197 of file PikaScript.h.

7.15.2.8 template<class S > bool Pika::STLValue< S >::operator!= (const STLValue< S > & *r*) const [inline]

Non-equality operator.

Notice that numbers are compared numerically (e.g. '1.0' and '1' are considered identical) and strings are compared literally (character by character).

Definition at line 205 of file PikaScript.h.

7.15.2.9 template<class S > bool Pika::STLValue< S >::operator< (const STLValue< S > & *r*) const

Less than comparison operator.

Notice that numbers are compared numerically and a number is always considered less than any non-number string.

Definition at line 262 of file PikaScriptImpl.h.

7.15.2.10 template<class S > bool Pika::STLValue< S >::operator<= (const STLValue< S > & r) const [inline]

Less than or equal to comparison operator.

Notice that numbers are compared numerically and a number is always considered less than any non-number string.

Definition at line 207 of file PikaScript.h.

7.15.2.11 template<class S > bool Pika::STLValue< S >::operator== (const STLValue< S > & r) const

Equality operator.

Notice that numbers are compared numerically (e.g. '1.0' and '1' are considered identical) and strings are compared literally (character by character).

Definition at line 268 of file PikaScriptImpl.h.

7.15.2.12 template<class S > bool Pika::STLValue< S >::operator> (const STLValue< S > & r) const [inline]

Greater than comparison operator.

Notice that numbers are compared numerically and a number is always considered less than any non-number string.

Definition at line 206 of file PikaScript.h.

7.15.2.13 template<class S > bool Pika::STLValue< S >::operator>= (const STLValue< S > & r) const [inline]

Greater than or equal to comparison operator.

Notice that numbers are compared numerically and a number is always considered less than any non-number string.

Definition at line 208 of file PikaScript.h.

7.15.2.14 template<class S > const STLValue< S > Pika::STLValue< S >::operator[] (const STLValue< S > & i) const

The subscript operator returns the concatenation of the value with the dot (.) separator (if necessary) and the value `i`.

Use it on a reference value to create a reference to a subscript element of that reference.

Definition at line 274 of file PikaScriptImpl.h.

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

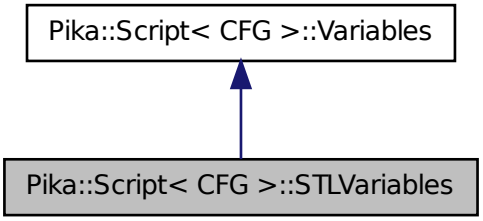
- /Users/Magnus/projects/PikaScript/src/PikaScript.h
- /Users/Magnus/projects/PikaScript/src/PikaScriptImpl.h

7.16 Pika::Script< CFG >::STLVariables Class Reference

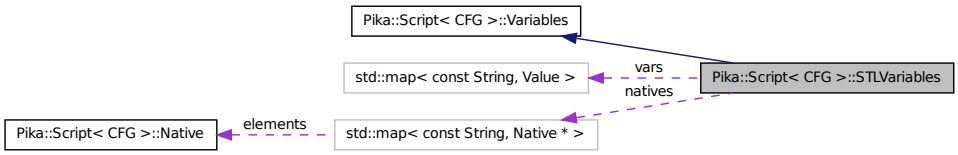
STLVariables is the reference implementation of a variable space.

`#include <PikaScript.h>`

Inheritance diagram for Pika::Script< CFG >::STLVariables:



Collaboration diagram for Pika::Script< CFG >::STLVariables:



7.16.1 Detailed Description

template<class CFG> class Pika::Script< CFG >::STLVariables

STLVariables is the reference implementation of a variable space. It simply uses two std::map's for the PikaScript variables and the natives respectively. All registered natives are deleted on the destruction of this class.

See Variables for descriptions on the overloaded member functions in this class.

Definition at line 435 of file PikaScript.h.

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

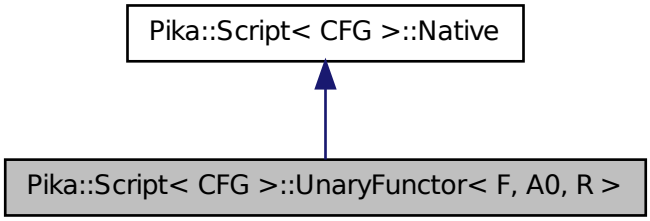
- /Users/Magnus/projects/PikaScript/src/PikaScript.h
- /Users/Magnus/projects/PikaScript/src/PikaScriptImpl.h

7.17 Pika::Script< CFG >::UnaryFunctor< F, A0, R > Class Template Reference

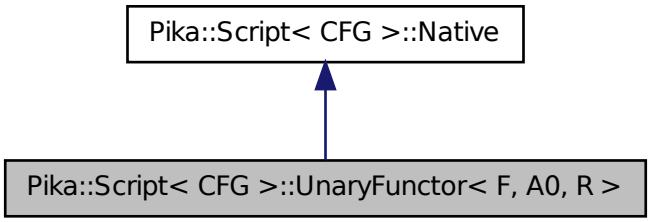
We provide two Native template classes for bridging PikaScript calls to C++ "functors".

`#include <PikaScript.h>`

Inheritance diagram for Pika::Script< CFG >::UnaryFunctor< F, A0, R >:



Collaboration diagram for Pika::Script< CFG >::UnaryFunctor< F, A0, R >:



Protected Member Functions

- template<class T >
  const Value arg (Frame &f, const T &)

  Get the first argument from the frame `f`.

- Frame & arg (Frame &f, const Dumb< Frame & > &)

  Overloaded arg() that returns the actual frame instead of getting the argument value if the functor argument references a Script::Frame& type.

- const Frame & arg (Frame &f, const Dumb< const Frame & > &)

  Overloaded arg() that returns the actual frame instead of getting the argument value if the functor argument references a const Script::Frame& type.

- template<class A , class T >
  Value call (A &a, const Dumb< T > &)

  Call the functor with the argument `a`.

- template<class A >
  Value call (A &a, const Dumb< void > &)

  Overloaded to return the `void` value for functors that returns the `void` type.

7.17.1 Detailed Description

template<class CFG>template<class F, class A0 = typename F::argument_type, class R = typename F::result_type> class Pika::Script< CFG >::UnaryFunctor< F, A0, R >

We provide two Native template classes for bridging PikaScript calls to C++ "functors". One that takes a single argument (UnaryFunctor) and one that takes two arguments (BinaryFunctor). A "functor" is either a class that has an overloaded operator() or a C / C++ function. It is a concept introduced to C++ with STL so please refer to your STL documentation of choice for more info. (For example: `http://www.sgi.com/tech/stl/functors.html`)

Thanks to some clever template tricks, these classes are very flexible when it comes to what type of arguments your functor can take and what type it may return. Here are your options:

- Any argument can be of a type that is convertible from Script::Value (e.g., `bool`, `long`, `double` etc).
- Likewise, the functor can return any type that is convertible to a Script::Value.
- You can also use a functor with `void` result type.
- The functor may take a single argument type of Script::Frame&. You can then retrieve all the arguments for the call by reading `$0`, `$1`, `$2` etc from the Frame (via Frame::get() or Frame::getOptional()).

In Frame you will find a template function (Frame::registerNative()) that allows you to register a native C++ function directly through a functor. It will construct the proper functor instance for you "in the background".

If you need use examples, please see the standard library implementation in PikaScriptImpl.h.

Definition at line 484 of file PikaScript.h.

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

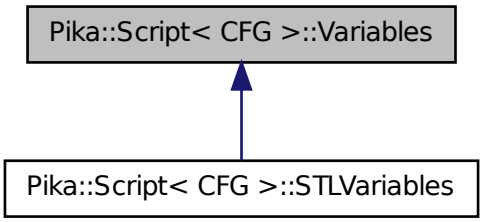
- /Users/Magnus/projects/PikaScript/src/PikaScript.h

7.18 Pika::Script< CFG >::Variables Class Reference

Variables is an abstract base class which implements the interface to the variable space that a Frame works on.

`#include <PikaScript.h>`

Inheritance diagram for Pika::Script< CFG >::Variables:



Public Member Functions

- virtual bool lookup (const String &symbol, Value &result)=0

  Lookup `symbol`.

- virtual bool assign (const String &symbol, const Value &value)=0

  Assign `value` to `symbol` and return true if the assignment succeeded.

- virtual bool erase (const String &symbol)=0

  Erase `symbol`. Return true if the symbol existed and was successfully erased.

- virtual void list (const String &key, VarList &list)=0

  Iterate all symbols that begins with `key` and push back names and values to `list`.

- virtual Native * lookupNative (const String &identifier)=0

  Lookup the native function (or object) with `identifier`.

- virtual bool assignNative (const String &identifier, Native *native)=0

  Assign the native function (or object) `native` *to* `identifier`*, replacing any already existing definition.*

- virtual ~Variables ()

  Destructor.

7.18.1 Detailed Description

template<class CFG> class Pika::Script< CFG >::Variables

Variables is an abstract base class which implements the interface to the variable space that a Frame works on. In the configuration meta-class class (Script::Config) two typedefs exist that determines which sub-classes of Variables should be used for the "root frame" (= Globals) and subsequently for the "sub-frames" (= Locals).

A standard Variables class is supplied in this header file (STLVariables). Custom sub-classes are useful for optimization and special integration needs.

Notice that the separation of Frames and Variables makes it possible to have more than one Frame referencing the same variable space. This could be useful for example in a threaded situation where several concurrent threads running PikaScript should share global variables. In this case each thread should still have a distinct "root frame" and you need to implement a sub-class of Variables that accesses its data in a thread-safe manner.

Definition at line 291 of file PikaScript.h.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 template<class CFG > TMPL Pika::Script< CFG >::Variables::~Variables ( ) [virtual]

Destructor.

Don't forget to delete all registered natives.

Definition at line 1048 of file PikaScriptImpl.h.

7.18.3 Member Function Documentation

7.18.3.1 template<class CFG > virtual bool Pika::Script< CFG >::Variables::assign (const String & *symbol,* const Value & *value*) [pure virtual]

Assign `value` to `symbol` and return true if the assignment succeeded.

If false is returned, the calling Frame::set() will throw an exception.

7.18.3.2 template<class CFG > virtual bool Pika::Script< CFG >::Variables::assignNative (const String & *identifier,* Native * *native*) [pure virtual]

Assign the native function (or object) `native` to `identifier`, replacing any already existing definition.

Once assigned, the native is considered "owned" by this variable space. This class is responsible for deleting its natives on destruction and also delete the existing definition when an identifier is being reassigned.

7.18.3.3 template<class CFG > virtual void Pika::Script< CFG >::Variables::list (const String & *key,* VarList & *list*) [pure virtual]

Iterate all symbols that begins with `key` and push back names and values to `list`.

There are no requirements on the order of the listed elements. You should not erase the list at the beginning.

7.18.3.4 template<class CFG > virtual bool Pika::Script< CFG >::Variables::lookup (const String & *symbol,* Value & *result*) [pure virtual]

Lookup `symbol`.

If found, store the found value in `result` and return true, otherwise return false.

7.18.3.5 template<class CFG > virtual Native* Pika::Script< CFG >::Variables::lookupNative (const String & *identifier*) [pure virtual]

Lookup the native function (or object) with `identifier`.

Return 0 if the native could not be found. In this case, the caller will throw an exception.

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

- /Users/Magnus/projects/PikaScript/src/PikaScript.h
- /Users/Magnus/projects/PikaScript/src/PikaScriptImpl.h

Chapter 8

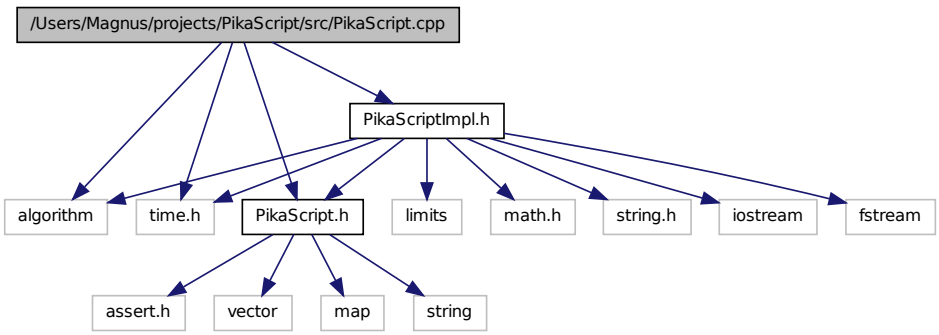
File Documentation

8.1 /Users/Magnus/projects/PikaScript/src/PikaScript.cpp File Reference

PikaScript is a high-level scripting language written in C++.

```
#include <algorithm>
#include <time.h>
#include "PikaScript.h"
#include "PikaScriptImpl.h"
```

Include dependency graph for PikaScript.cpp:



Namespaces

- namespace Pika

The PikaScript namespace.

8.1.1 Detailed Description

PikaScript is a high-level scripting language written in C++.

Version

Version 0.93

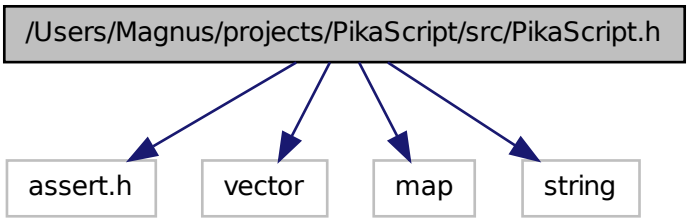
Definition in file PikaScript.cpp.

8.2 /Users/Magnus/projects/PikaScript/src/PikaScript.h File Reference

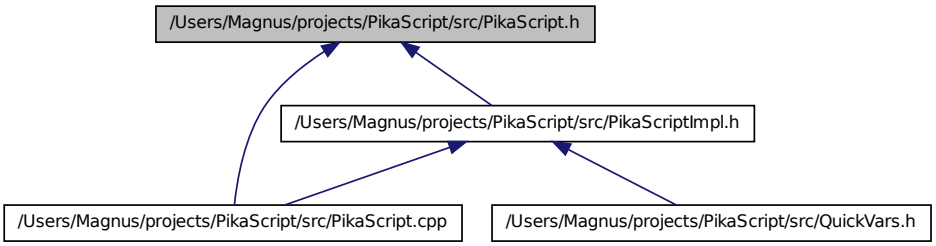
PikaScript is a high-level scripting language written in C++.

```
#include "assert.h"
#include <vector>
#include <map>
#include <string>
```

Include dependency graph for PikaScript.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Pika::bound_mem_fun_t< C, A0, R >

  bound_mem_fun_t is a member functor bound to a specific C++ object through a pointer.

- class Pika::Dumb< T >

  We use this dummy class to specialize member functions for arbitrary types (including void, references etc).

- class Pika::Exception< S >

  The PikaScript exception class.

- class Pika::STLValue< S >

  STLValue is the reference implementation of a PikaScript variable.

- struct Pika::Script< CFG >

  Script is a meta-class that groups all the core classes of the PikaScript interpreter together (except for the value class).

- class Pika::Script< CFG >::Variables

  Variables is an abstract base class which implements the interface to the variable space that a Frame works on.

- class Pika::Script< CFG >::Frame

  The execution context and interpreter for PikaScript.

- class Pika::Script< CFG >::Root

  The Root is the first Frame you instantiate.

- class Pika::Script< CFG >::FullRoot

FullRoot inherits from both Root and CFG::Globals (which should be a descendant to Variable).

- class Pika::Script< CFG >::STLVariables

  STLVariables is the reference implementation of a variable space.

- class Pika::Script< CFG >::Native

  Native is the base class for the native functions and objects that can be accessed from PikaScript.

- class Pika::Script< CFG >::UnaryFunctor< F, A0, R >

  We provide two Native template classes for bridging PikaScript calls to C++ "functors".

- class Pika::Script< CFG >::BinaryFunctor< F, A0, A1, R >

  See UnaryFunctor for documentation.

- struct Pika::StdConfig

  StdConfig is a configuration class for Script that uses the reference implementations of STLValue and Script::STLVariables.

Namespaces

- namespace Pika

  The PikaScript namespace.

Typedefs

- typedef Script< StdConfig > Pika::StdScript

  This typedef exist for your convenience.

Enumerations

- enum Pika::Precedence {

  Pika::NO_TRACE = 0, Pika::TRACE_ERROR = 1, Pika::TRACE_CALL = 2, Pika::TRACE_LOOP = 3,

  Pika::STATEMENT = 4, Pika::BODY = 5, Pika::ARGUMENT = 6, Pika::BRACKETS = 7,

  Pika::ASSIGN = 8, Pika::LOGICAL_OR = 9, Pika::LOGICAL_AND = 10, Pika::BIT_OR = 11,

  Pika::BIT_XOR = 12, Pika::BIT_AND = 13, Pika::EQUALITY = 14, Pika::COMPARE = 15,

Pika::CONCAT = 16, Pika::SHIFT = 17, Pika::ADD_SUB = 18, Pika::MUL_-DIV = 19,

Pika::PREFIX = 20, Pika::POSTFIX = 21, Pika::DEFINITION = 22 }

Precedence levels are used both internally for the parser and externally for the tracing mechanism.

Functions

- template<class C , class A0 , class R >
  bound_mem_fun_t< C, A0, R > Pika::bound_mem_fun (R(C::∗m)(A0), C ∗o)

  bound_mem_fun creates a member functor bound to a specific C++ object through a pointer.

Conversion routines for string <-> other types.

For some of these we could use stdlib implementations yes, but:

1. Some of them (e.g. atof, strtod) behaves differently depending on global "locale" setting. We can't have that.
2. The stdlib implementations can be slow (e.g. my double->string conversion is about 3 times faster than MSVC CRT).
3. Pika requires high-precision string representation and proper handling of trailing 9's etc.

- template<class S >
  std::string Pika::toStdString (const S &s)

  Converts the string `s` to a standard C++ string.

- template<class S >
  ulong Pika::hexToLong (typename S::const_iterator &p, const typename S::const_-iterator &e)

  Converts a string in hexadecimal form to an ulong integer.

- template<class S >
  long Pika::stringToLong (typename S::const_iterator &p, const typename S::const_-iterator &e)

  Converts a string in decimal form to a signed long integer.

- template<class S , typename T >
  S Pika::intToString (T i, int radix=10, int minLength=1)

  Converts the integer `i` to a string with a radix and minimum length of your choice.

- template<class S >
  double Pika::stringToDouble (typename S::const_iterator &p, const typename S::const_iterator &e)

  Converts a string in scientific e notation (e.g. -12.34e-3) to a double floating point value.

- template<class S >
  bool Pika::stringToDouble (const S &s, double &d)

  A convenient utility routine that tries to convert the entire string `s` *(in scientific e notation) to a double, returning true on success or false if the string is not in valid syntax.*

- template<class S >
  S Pika::doubleToString (double d, int precision=14)

  Converts the double `d` *to a string (in scientific e notation, e.g. -12.34e-3).*

- template<class S >
  S Pika::unescape (typename S::const_iterator &p, const typename S::const_iterator &e)

  Converts a string that is either enclosed in single (' ') or double (" ") quotes.

- template<class S >
  S Pika::escape (const S &s)

  Depending on the contents of the source string `s` *it is encoded either in single (' ') or double (" ") quotes.*

8.2.1 Detailed Description

PikaScript is a high-level scripting language written in C++.

Version

Version 0.93

Definition in file PikaScript.h.

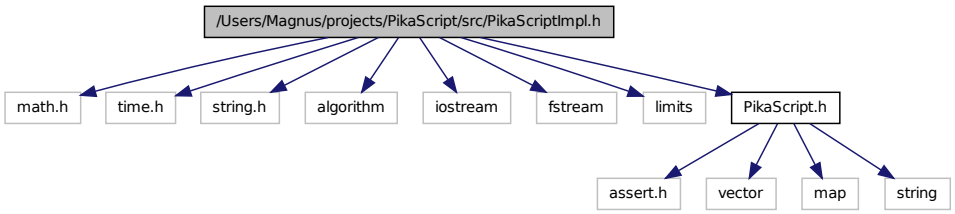
8.3 /Users/Magnus/projects/PikaScript/src/PikaScriptImpl.h File Reference

PikaScriptImpl contains the template definitions for PikaScript.

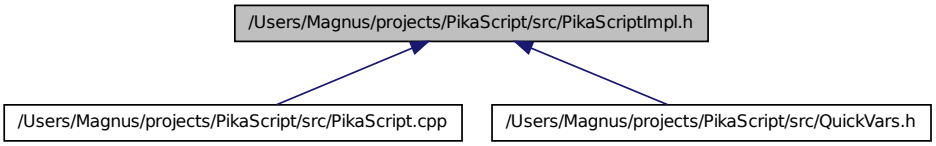
```
#include <math.h>
#include <time.h>
#include <string.h>
```

```
#include <algorithm>
#include <iostream>
#include <fstream>
#include <limits>
#include "PikaScript.h"
```

Include dependency graph for PikaScriptImpl.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Pika

  The PikaScript namespace.

Conversion routines for string <-> other types.

For some of these we could use stdlib implementations yes, but:

1. Some of them (e.g. atof, strtod) behaves differently depending on global "locale" setting. We can't have that.

2. The stdlib implementations can be slow (e.g. my double->string conversion is about 3 times faster than MSVC CRT).
3. Pika requires high-precision string representation and proper handling of trailing 9's etc.

- template<class S >
  std::string Pika::toStdString (const S &s)

  Converts the string `s` to a standard C++ string.

- template<class S >
  ulong Pika::hexToLong (typename S::const_iterator &p, const typename S::const_iterator &e)

  Converts a string in hexadecimal form to an ulong integer.

- template<class S >
  long Pika::stringToLong (typename S::const_iterator &p, const typename S::const_iterator &e)

  Converts a string in decimal form to a signed long integer.

- template<class S , typename T >
  S Pika::intToString (T i, int radix=10, int minLength=1)

  Converts the integer `i` to a string with a radix and minimum length of your choice.

- template<class S >
  double Pika::stringToDouble (typename S::const_iterator &p, const typename S::const_iterator &e)

  Converts a string in scientific e notation (e.g. -12.34e-3) to a double floating point value.

- template<class S >
  bool Pika::stringToDouble (const S &s, double &d)

  A convenient utility routine that tries to convert the entire string `s` (in scientific e notation) to a double, returning true on success or false if the string is not in valid syntax.

- template<class S >
  S Pika::doubleToString (double d, int precision=14)

  Converts the double `d` to a string (in scientific e notation, e.g. -12.34e-3).

- template<class S >
  S Pika::unescape (typename S::const_iterator &p, const typename S::const_iterator &e)

  Converts a string that is either enclosed in single (' ') or double (" ") quotes.

- template<class S >
  S Pika::escape (const S &s)

  Depending on the contents of the source string `s` it is encoded either in single (' ') or double (" ") quotes.

8.3.1 Detailed Description

PikaScriptImpl contains the template definitions for PikaScript. You only need to include this file if you want to instantiate a customization on the reference implementation on PikaScript. If you are satisfied with the reference implementation (StdScript), you only need to include PikaScript.h and add PikaScript.cpp to your project.

Version

Version 0.93

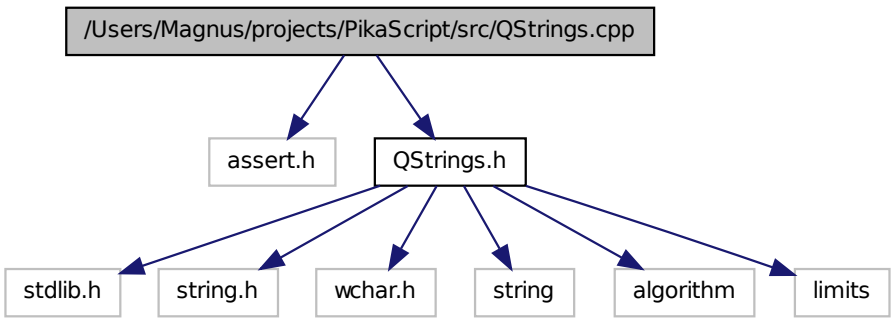
Definition in file PikaScriptImpl.h.

8.4 /Users/Magnus/projects/PikaScript/src/QStrings.cpp File Reference

QStrings is a high performance string class inspired by std::string.

```
#include <assert.h>
#include "QStrings.h"
```

Include dependency graph for QStrings.cpp:



8.4.1 Detailed Description

QStrings is a high performance string class inspired by std::string. QStrings implements a subset of std::string and can be used as a much optimized string implementation for PikaScript. It achieves its great performance by:

1) Maintaining a separate memory pool for smaller string buffers instead of using the slower standard heap. 2) Sharing string buffers not only for full strings, but even for substrings.

Warning

WARNING! The current implementation of the memory pool is ∗not∗ thread-safe due to unprotected use of shared global data. You must only use QStrings in single-threaded applications or in the case of a multi-threaded application you must only use QStrings from a single thread at a time!

Version

Version 0.92

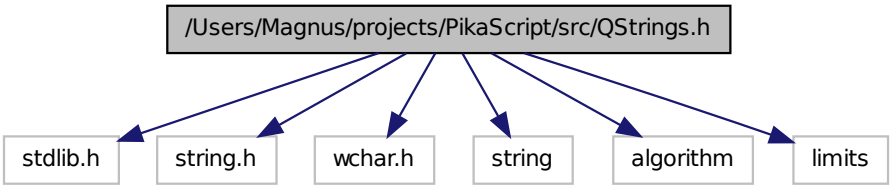
Definition in file QStrings.cpp.

8.5 /Users/Magnus/projects/PikaScript/src/QStrings.h File Reference

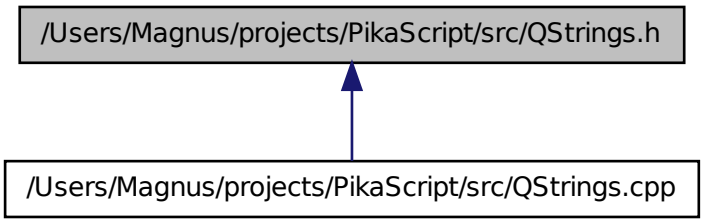
QStrings is a high performance string class inspired by std::string.

```
#include <stdlib.h>
#include <string.h>
#include <wchar.h>
#include <string>
#include <algorithm>
#include <limits>
```

Include dependency graph for QStrings.h:



This graph shows which files directly or indirectly include this file:



Classes

- class QStrings::QString< C, PS >
- class QStrings::QString< C, PS >::Buffer
- class QStrings::QString< C, PS >::_iterator< E, Q >

8.5.1 Detailed Description

QStrings is a high performance string class inspired by std::string. QStrings implements a subset of std::string and can be used as a much optimized string implementation for PikaScript. It achieves its great performance by:

1) Maintaining a separate memory pool for smaller string buffers instead of using the slower standard heap. 2) Sharing string buffers not only for full strings, but even for substrings.

Warning

WARNING! The current implementation of the memory pool is ∗not∗ thread-safe due to unprotected use of shared global data. You must only use QStrings in single-threaded applications or in the case of a multi-threaded application you must only use QStrings from a single thread at a time!

Version

Version 0.92

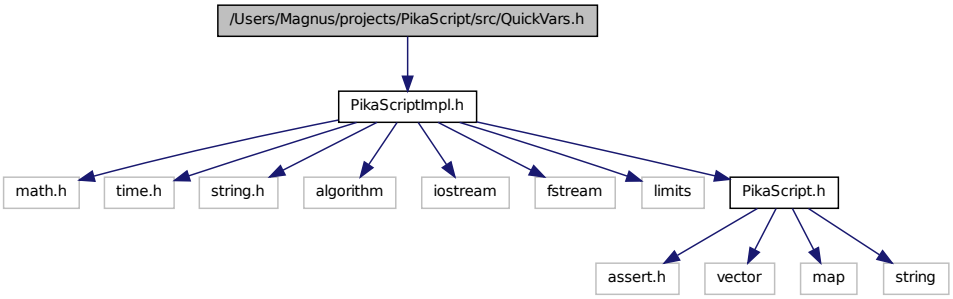
Definition in file QStrings.h.

8.6 /Users/Magnus/projects/PikaScript/src/QuickVars.h File Reference

QuickVars is a (generally) faster version of the reference implementation's STLVariable.

```
#include "PikaScriptImpl.h"
```

Include dependency graph for QuickVars.h:



Classes

- class Pika::QuickVars< Super, CACHE_SIZE >

Namespaces

- namespace Pika

  The PikaScript namespace.

8.6.1 Detailed Description

QuickVars is a (generally) faster version of the reference implementation's STLVariable. It achieves its better performance by caching the most recently used variables in a super-tiny hash table. The downside is that it uses more stack memory, especially if you have deep calling stacks with few local variables.

Version

Version 0.92

Definition in file QuickVars.h.

Index