Morpho 0.5

Generated by Doxygen 1.8.20

Chapter 1

file

[comment]: # File class help [version]: # 0.5

#File [tag]: # File

The File class provides the capability to read from and write to files, or to obtain the contents of a file in convenient formats.

To open a file, create a File object with the filename as the argument

```
var f = File("myfile.txt")
```

which opens `"myfile.txt"` for *reading*. To open a file for writing or appending, you need to provide a mode selector

```
var g = File("myfile.txt", "write")
```

or

```
var g = File("myfile.txt", "append")
```

Once the file is open, you can then read or write by calling appropriate methods:

```
f.lines()            // reads the contents of the file into an array of lines.
f.readline()         // reads a single line
f.readchar()         // reads a single character.
f.write(string)      // writes the arguments to the file.
```

After you're done with the file, close it with

```
f.close()
```

#File.lines [tag]: # lines

Returns the contents of a file as an array of strings; each element corresponds to a single line.

#File.readline [tag]: # readline

Reads a single line from a file; returns the result as a string.

#File.readchar [tag]: # readchar

Reads a single character from a file; returns the result as a string.

#File.write [tag]: # write

Writes to a file.

#File.close [tag]: # close

Closes an open file

Chapter 2

index

[comment]: # Morpho language help file [version]: # 0.5

#Help [tag]: # help

Welcome to the morpho interactive help system. To get help about a topic called `topicname`, type

```
help topicname
```

Possible topics include language keywords like `class`, `fn` and `for`, built in classes like `Graphics` and `File` or information about functions like `exp` and `random`.

Some topics have additional subtopics: to access these type

```
help topic subtopic
```

For example, to get help on a method for a particular class, you could type

```
help Classname methodname
```

You can also use `?` as a shorthand synonym for `help`

```
? topic
```

Chapter 3

language

[comment]: # Morpho language help file [version]: # 0.5

#Functions [tag]: # fn [tag]: # func

A function in Morpho is defined with the `fn` keyword, followed by the function's name, a list of parameters enclosed in parentheses, and the body of the function in curly braces. This example computes the square of a number:

```
fn sqr(x) {
  return x*x
}
```

#Return [tag]: # return

The `return` keyword is used to exit from a function, optionally passing a given value back to the caller. `return` can be used anywhere within a function. The below example calculates the `n` th Fibonacci number,

```
fn fib(n) {
  if (n<2) return n
  return fib(n-1) + fib(n-2)
}
```

by returning early if `n<2`, otherwise returning the result by recursively calling itself.

#Variables [tag]: # var

Variables are defined using the `var` keyword followed by the variable name:

```
var a
```

Optionally, an initial assignment may be given:

```
var a = 1
```

Variables defined in a block of code are visible only within that block, so

```
var greeting = "Hello"
{
    var greeting = "Goodbye"
    print greeting
}
print greeting
```

will print

Goodbye Hello

Multiple variables can be defined at once by separating them with commas

```
var a, b=2, c[2]=[1,2]
```

where each can have its own initializer (or not).

#Classes [tag]: # class

Classes are defined using the `class` keyword followed by the name of the class. The definition includes methods that the class responds to. The special `init` method is called whenever an object is created.

```
class Cake {
    init(type) {
        self.type = type
    }

    eat() {
        print "A delicious "+type+" cake"
    }
}
```

Objects are created by calling the class as if it was a function:

```
var c = Cake("carrot")
```

Methods are called using the . operator:

```
c.eat()
```

#If [tag]: # if

If allows you to selectively execute a section of code depending on whether a condition is met. The simplest version looks like this:

```
if (x<1) print x
```

where the body of the loop, `print x`, is only executed if x is less than 1. The body can be a code block to accomodate longer sections of code:

```
if (x<1) {
    ... // do something
}
```

If you want to choose between two alternatives, use `else`:

```
if (a==b) {
    // do something
} else {
    // this code is executed only if the condition is false
}
```

You can even chain multiple tests together like this:

```
if (a==b) {
    // option 1
} else if (a==c) {
    // option 2
} else {
    // something else
}
```

#While [tag]: # while

While loops

#For [tag]: # for

For loops allow you to repeatedly execute a section of code. They come in two versions: the simpler version looks like this,

```
for (i in 1..5) print i
```

which prints the numbers 1 to 5 in turn. The variable `i` is the *loop variable*, which takes on a different value each iteration. `1..5` is a range, which denotes a sequence of numbers. The *body* of the loop, `print i`, is the code to be repeatedly executed.

If you want your loop variable to count in increments other than 1, you can specify a stepsize in the range:

```
for (i in 1..5:2) print i
              ^step
```

Ranges need not be integer:

```
for (i in 0.1..0.5:0.1) print i
```

You can also replace the range with other kinds of collection object to loop over their contents:

```
var a = Matrix([1,2,3,4])
for (x in a) print x
```

Morpho also provides a second form of `for` loop similar to that in C:

```
for (var i=0; i<5; i+=1) { print i }
     ^start   ^test ^inc.  ^body
```

which is executed as follows: start: the variable `i` is initially set to zero. test: before each iteration, the test is evaluated. If the test is `false`, the loop terminates. body: the body of the loop is executed. inc: the variable `i` is increased by 1.

This kind of loop is very flexible as you can include any code that you like in each of the sections.

#Indexing [tag]: # [[tag]: #] [tag]: # index [tag]: # subscript

Indexing

#Example

This is *very* important, this is *underlined* and `var` is code.

Chapter 4

matrix

[comment]: # Matrix class help [version]: # 0.5

#Matrix [tag]: # Matrix

The Matrix class provides support for matrices. A matrix can be initialized with a given size,

```
var a = Matrix(nrows,ncols)
```

where all elements are initially set to zero. Alternatively, a matrix can be created from an array,

```
var a = Matrix([[1,2], [3,4]])
```

You can create a column vector like this,

```
var v = Matrix([1,2])
```

Once a matrix is created, you can use all the regular arithmetic operators with matrix operands, e.g.

```
a+b
a*b
```

The division operator is used to solve a linear system, e.g.

```
var a = Matrix([[1,2],[3,4]])
var b = Matrix([1,2])

print b/a
```

yields the solution to the system a∗x = b.

#MtrxIncmptbl [tag]: # MtrxIncmptbl

This error occurs when an arithmetic operation is performed on two 'incompatible' matrices. For example, two matrices must have the same dimensions, i.e. the same number of rows and columns, to be added or subtracted,

```
var a = Matrix([[1,2],[3,4]])
var b = Matrix([[1]])
print a+b // generates a `MtrxIncmptbl` error.
```

Or to be multiplied together, the number of columns of the left hand matrix must equal the number of rows of the right hand matrix.

```
var a = Matrix([[1,2],[3,4]])
var b = Matrix([1,2])
print a*b // ok
print b*a // generates a `MtrxIncmptbl` error.
```

Chapter 5

sparse

[comment]: # Sparse class help [version]: # 0.5

#Sparse [tag]: # Sparse

The Sparse class provides support for sparse matrices. An empty sparse matrix can be initialized with a given size,

```
var a = Sparse(nrows,ncols)
```

Alternatively, a matrix can be created from an array of triplets,

```
var a = Matrix([[row, col, value] ...])
```

For example

```
var a = Matrix([[0,0,2], [1,1,-2]])
```

creates the matrix

```
[ 2 0 ]
[ 0 -2 ]
```

Once a sparse matrix is created, you can use all the regular arithmetic operators with matrix operands, e.g.

```
a+b
a*b
```

Chapter 6

syntax

[comment]: # Morpho language help file [version]: # 0.5

#Syntax [tag]: # syntax

Morpho provides a flexible object oriented language similar to other languages in the C family (like C++, Java and Javascript) with a simplified syntax.

Morpho programs are stored as plain text with the .morpho file extension. A program can be run from the command line by typing

```
morpho program.morpho
```

#Comments [tag]: # comment [tag]: # Comments [tag]: # // [tag]: # /* [tag]: # */ Two types of comment are available. The first type is called a 'line comment' whereby text after // on the same line is ignored by the interpreter.

```
a.dosomething() // A comment
```

Longer 'block' comments can be created by placing text between /* and */. Newlines are ignored

```
/* This
   is
   a longer comment */
```

In contrast to C, these comments can be nested

```
/* A nested /* comment */ */
```

enabling the programmer to quickly comment out a section of code.

#Symbols [tag]: # symbols [tag]: # names

Symbols are used to refer to named entities, including variables, classes, functions etc. Symbols must begin with a letter or underscore _ as the first character and may include letters or numbers as the remainder. Symbols are case sensitive.

```
asymbol
_alsoasymbol
another_symbol
EvenThis123
YET_ANOTHER_SYMBOL
```

Classes are typically given names with an initial capital letter. Variable names are usually all lower case.

#Newlines [tag]: # newlines

Morpho accepts newlines in place of a semicolon to end a statement.

```
var a = 1; //
```

#Blocks

#Precedence

Chapter 7

Class Index

7.1 Class List

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

Chapter 8

File Index

8.1 File List

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

Chapter 9

Class Documentation

9.1 _syntaxtreenode Struct Reference

A node on the syntax tree is defined by a value and indices of the left and right elements.

`#include <syntaxtree.h>`

Public Attributes

- syntaxtreenodetype **type**
- value **content**
- int **line**
- int **posn**
- syntaxtreeindx **left**
- syntaxtreeindx **right**

9.1.1 Detailed Description

A node on the syntax tree is defined by a value and indices of the left and right elements.

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

- datastructures/syntaxtree.h

9.2 builtinclassentry Struct Reference

`#include <builtin.h>`

Public Types

- enum { **BUILTIN_METHOD**, **BUILTIN_PROPERTY** }

Public Attributes

- enum builtinclassentry:: { ... } **type**
- char * **name**
- builtinfunctionflags **flags**
- builtinfunction **function**

9.2.1 Detailed Description

A type used to store the entries of a built in class

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

- builtin/builtin.h

9.3 callframe Struct Reference

Public Attributes

- objectfunction * **function**
- objectclosure * **closure**
- value * **reg**
- instruction * **pc**
- unsigned int **stackcount**

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

- vm/core.h

9.4 codeinfo Struct Reference

Public Types

- enum { **REGISTER**, **CONSTANT**, **UPVALUE**, **GLOBAL** }

Public Attributes

- enum codeinfo:: { ... } **returntype**
- registerindx **dest**
- unsigned int **ninstructions**

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

- vm/compile.h

9.5 compilenoderule Struct Reference

A compilenoderule rule will be defined for each syntax tree node type, providing a function to compile the node.

`#include <compile.h>`

Public Attributes

- compiler_nodefn **nodefn**

9.5.1 Detailed Description

A compilenoderule rule will be defined for each syntax tree node type, providing a function to compile the node.

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

- vm/compile.h

9.6 debuginfo Struct Reference

Public Attributes

- unsigned int **ninstr**
- objectfunction * **function**
- unsigned int **line**
- unsigned int **posn**

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

- vm/core.h

9.7 dictionary Struct Reference

dictionary data structure that maps keys to values

`#include <dictionary.h>`

Public Attributes

- unsigned int **capacity**
- unsigned int count
- dictionaryentry * contents

9.7.1 Detailed Description

dictionary data structure that maps keys to values

9.7.2 Member Data Documentation

9.7.2.1 contents

```
dictionaryentry* dictionary::contents
```

number of items in the dictionary

9.7.2.2 count

```
unsigned int dictionary::count
```

capacity of the dictionary

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

- datastructures/dictionary.h

9.8 dictionaryentry Struct Reference

A single dictionary entry.

```
#include <dictionary.h>
```

Public Attributes

- value **key**
- value **val**

9.8.1 Detailed Description

A single dictionary entry.

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

- datastructures/dictionary.h

9.9 error Struct Reference

A static container for error messages.

`#include <error.h>`

Public Attributes

- errorcategory **cat**
- errorid **id**
- int **line**
- int **posn**
- char **msg** [MORPHO_ERRORSTRINGSIZE]

9.9.1 Detailed Description

A static container for error messages.

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

- utils/error.h

9.10 errordefinition Struct Reference

Definition of an error message.

`#include <error.h>`

Public Attributes

- errorcategory **cat**
- char * **msg**

9.10.1 Detailed Description

Definition of an error message.

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

- utils/error.h

9.11 functionstate Struct Reference

`#include <compile.h>`

Public Attributes

- objectfunction * **func**
- functiontype **type**
- varray_registeralloc **registers**
- varray_upvalue **upvalues**
- unsigned int **nreg**
- unsigned int **scopedepth**
- bool **inargs**

9.11.1 Detailed Description

This structure tracks compiler information for the current function.

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

- vm/compile.h

9.12 graylist Struct Reference

`#include <core.h>`

Public Attributes

- unsigned int **graycount**
- unsigned int **graycapacity**
- object ** **list**

9.12.1 Detailed Description

Gray list for garbage collection

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

- vm/core.h

9.13 keypress Struct Reference

Public Attributes

- keytype **type**
- char **c**

9.13.1 Detailed Description

A single keypress event obtained and processed by the terminal

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

- interface/linedit.c

9.14 lexer Struct Reference

Store the current configuration of a lexer.

```
#include <parse.h>
```

Public Attributes

- const char * **start**
- const char * current
- int line
- int posn

9.14.1 Detailed Description

Store the current configuration of a lexer.

9.14.2 Member Data Documentation

9.14.2.1 current

```
const char* lexer::current
```

Starting point to lex

9.14.2.2 line

```
int lexer::line
```

Current point

9.14.2.3 posn

```
int lexer::posn
```

Line number

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

- utils/parse.h

9.15 linedit_stringlist Struct Reference

```
#include <linedit.h>
```

Public Attributes

- int **posn**
- linedit_string * **first**

9.15.1 Detailed Description

A list of strings

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

- interface/linedit.h

9.16 linedit_syntaxcolordata Struct Reference

```
#include <linedit.h>
```

Public Attributes

- linedit_tokenizer **tokenizer**
- unsigned int **ncols**
- bool **lexwarning**
- linedit_color **col** [ ]

9.16.1 Detailed Description

Structure to hold all information related to syntax coloring

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

- interface/linedit.h

9.17 linedit_token Struct Reference

`#include <linedit.h>`

Public Attributes

- unsigned int **type**
- char * **start**
- size_t **length**

9.17.1 Detailed Description

lineditor tokens

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

- interface/linedit.h

9.18 lineditor Struct Reference

`#include <linedit.h>`

Public Attributes

- lineditormode **mode**
- int **posn**
- int **sposn**
- linedit_string **prompt**
- linedit_string **current**
- linedit_string **clipboard**
- linedit_stringlist **history**
- linedit_stringlist **suggestions**
- linedit_syntaxcolordata * **color**
- linedit_completer **completer**

9.18.1 Detailed Description

Holds all state information needed for a line editor

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

- interface/linedit.h

9.19 objectarray Struct Reference

Public Attributes

- object **obj**
- unsigned int **dimensions**
- unsigned int **nelements**
- value **data** [ ]

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

- datastructures/object.h

9.20 objectbuiltinfunction Struct Reference

```
#include <builtin.h>
```

Public Attributes

- object **obj**
- value **name**
- builtinfunctionflags **flags**
- builtinfunction **function**

9.20.1 Detailed Description

A built in function object

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

- builtin/builtin.h

9.21 objectclosure Struct Reference

Public Attributes

- object **obj**
- objectfunction * **func**
- int **nupvalues**
- objectupvalue * **upvalues** [ ]

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

- datastructures/object.h

9.22 objectdictionary Struct Reference

Public Attributes

- object **obj**
- dictionary **dict**

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

- datastructures/object.h

9.23 objectdokkey Struct Reference

```
#include <object.h>
```

Public Attributes

- object **obj**
- unsigned int **row**
- unsigned int **col**

9.23.1 Detailed Description

The dictionary of keys format uses this special object type to store indices, enabling use of the existing dictionary type.

Warning

These are for internal use only and should never be returned to user code

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

- datastructures/object.h

9.24 objectinstance Struct Reference

Public Attributes

- object **obj**
- objectclass * **klass**
- dictionary **fields**

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

- datastructures/object.h

9.25 objectinvocation Struct Reference

Public Attributes

- object **obj**
- value **receiver**
- value **method**

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

- datastructures/object.h

9.26 objectmatrix Struct Reference

`#include <object.h>`

Public Attributes

- object **obj**
- unsigned int **nrows**
- unsigned int **ncols**
- double * **elements**
- double **matrixdata** [ ]

9.26.1 Detailed Description

Matrices are a purely numerical collection type oriented toward linear algebra. Elements are stored in column-major format, i.e. [1 2] [3 4] is stored (1, 3, 2, 4) in memory. This is for compatibility with standard linear algebra packages

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

- datastructures/object.h

9.27 objectmesh Struct Reference

Public Attributes

- object **obj**
- unsigned int **dim**
- objectmatrix * **vert**
- objectarray * **conn**

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

- datastructures/object.h

9.28 objectrange Struct Reference

Public Attributes

- object **obj**
- unsigned int **nsteps**
- value **start**
- value **end**
- value **step**

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

- datastructures/object.h

9.29 objectsparse Struct Reference

Public Attributes

- object **obj**
- sparsedok **dok**
- sparseccs **ccs**

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

- datastructures/object.h

9.30 objectstring Struct Reference

```
#include <object.h>
```

Public Attributes

- object **obj**
- size_t **length**
- char * **string**
- char **stringdata** [ ]

9.30.1 Detailed Description

A string object

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

- datastructures/object.h

9.31 parser Struct Reference

A structure that defines the state of a parser.

`#include <parse.h>`

Public Attributes

- token **current**
- token previous
- syntaxtreeindx left
- lexer * **lex**
- syntaxtree * tree
- error * err
- bool nl

9.31.1 Detailed Description

A structure that defines the state of a parser.

9.31.2 Member Data Documentation

9.31.2.1 err

`error* parser::err`

Syntax tree receiving output

9.31.2.2 left

`syntaxtreeindx parser::left`

The previous token

9.31.2.3 nl

`bool parser::nl`

Error structure to output errors to

9.31.2.4 previous

```
token parser::previous
```

The current token

9.31.2.5 tree

```
syntaxtree* parser::tree
```

Lexer to use

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

- utils/parse.h

9.32 parserule Struct Reference

A parse rule will be defined for each token, providing functions to parse the token if it is encountered in the prefix or infix positions. The parse rule also defines the precedence.

```
#include <parse.h>
```

Public Attributes

- parsefunction **prefix**
- parsefunction **infix**
- precedence **precedence**

9.32.1 Detailed Description

A parse rule will be defined for each token, providing functions to parse the token if it is encountered in the prefix or infix positions. The parse rule also defines the precedence.

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

- utils/parse.h

9.33 registeralloc Struct Reference

```
#include <compile.h>
```

Public Attributes

- bool **isallocated**
- bool iscaptured
- unsigned int scopedepth
- value symbol

9.33.1 Detailed Description

This structure tracks the contents of each register as the function is being compiled.

9.33.2 Member Data Documentation

9.33.2.1 iscaptured

```
bool registeralloc::iscaptured
```

Whether the register has been allocated

9.33.2.2 scopedepth

```
unsigned int registeralloc::scopedepth
```

Whether the register becomes an upvalue

9.33.2.3 symbol

```
value registeralloc::symbol
```

Scope depth at which the register was allocated

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

- vm/compile.h

9.34 scompiler Struct Reference

A structure that stores the state of a compiler.

```
#include <compile.h>
```

Public Attributes

- lexer **lex**
- parser **parse**
- syntaxtree **tree**
- error **err**
- dictionary **globals**
- functionstate **fstack** [MORPHO_CALLFRAMESTACKSIZE]
- indx **fstackp**
- objectfunction * **prevfunction**
- objectclass * **currentclass**
- compilerlist * **currentlist**
- compilerlist * **prevlist**
- syntaxtreenode * **currentmethod**
- program * **out**
- struct scompiler * **parent**

9.34.1 Detailed Description

A structure that stores the state of a compiler.

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

- vm/compile.h

9.35 scompilerlist Struct Reference

`#include <compile.h>`

Public Attributes

- varray_value **entries**
- struct scompilerlist * **next**

9.35.1 Detailed Description

This structure holds a list as it is being created

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

- vm/compile.h

9.36 slinedit_string Struct Reference

`#include <linedit.h>`

Public Attributes

- size_t **capacity**
- size_t **length**
- char * **string**
- struct slinedit_string * **next**

9.36.1 Detailed Description

lineditor strings

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

- interface/linedit.h

9.37 sobject Struct Reference

`#include <object.h>`

Public Types

- enum { **OBJECT_ISUNMANAGED**, **OBJECT_ISUNMARKED**, **OBJECT_ISMARKED** }

Public Attributes

- objecttype **type**
- enum sobject:: { ... } **status**
- hash **hsh**
- struct sobject * **next**

9.37.1 Detailed Description

Simplest object

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

- datastructures/object.h

9.38 sobjectclass Struct Reference

Public Attributes

- object **obj**
- struct sobjectclass * **superclass**
- value **name**
- dictionary **methods**

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

- datastructures/object.h

9.39 sobjectfunction Struct Reference

`#include <object.h>`

Public Attributes

- object **obj**
- int **nargs**
- value **name**
- indx **entry**
- struct sobjectfunction * **parent**
- int **nupvalues**
- int **nregs**
- varray_value **konst**
- varray_varray_upvalue **prototype**

9.39.1 Detailed Description

A function object

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

- datastructures/object.h

9.40 sobjecthelptopic Struct Reference

Public Attributes

- object **obj**
- char * **topic**
- char * **file**
- long int **location**
- struct sobjecthelptopic * **parent**
- struct sobjecthelptopic * **next**
- dictionary **subtopics**

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

- interface/help.h

9.41 sobjectupvalue Struct Reference

Public Attributes

- object **obj**
- value * **location**
- value closed
- struct sobjectupvalue * next

9.41.1 Member Data Documentation

9.41.1.1 closed

```
value sobjectupvalue::closed
```

Pointer to the location of the upvalue

9.41.1.2 next

```
struct sobjectupvalue* sobjectupvalue::next
```

Closed value of the upvalue

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

- datastructures/object.h

9.42 sparseccs Struct Reference

Public Attributes

- int **nentries**
- int **nrows**
- int **ncols**
- int * **cptr**
- int * **rix**
- double * **values**

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

- datastructures/object.h

9.43 sparsedok Struct Reference

Public Attributes

- int **nrows**
- int **ncols**
- dictionary **dict**
- objectdokkey * **keys**

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

- datastructures/object.h

9.44 sprogram Struct Reference

Morpho code program and associated data.

`#include <core.h>`

Public Attributes

- varray_instruction **code**
- varray_debuginfo info
- objectfunction * global
- unsigned int nglobals
- object * **boundlist**
- dictionary symboltable

9.44.1 Detailed Description

Morpho code program and associated data.

9.44.2 Member Data Documentation

9.44.2.1 global

`objectfunction* sprogram::global`

Information about how the code connects to the source

9.44.2.2 info

`varray_debuginfo sprogram::info`

Compiled instructions

9.44.2.3 nglobals

`unsigned int sprogram::nglobals`

Pseudofunction containing global data

9.44.2.4 symboltable

```
dictionary sprogram::symboltable
```

Linked list of static objects bound to this program

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

- vm/core.h

9.45 svm Struct Reference

A Morpho virtual machine and its current state.

```
#include <core.h>
```

Public Attributes

- program * **current**
- varray_value globals
- varray_value stack
- callframe frame [MORPHO_CALLFRAMESTACKSIZE]
- value * sp
- callframe * **fp**
- error **err**
- object * objects
- graylist gray
- size_t bound
- size_t nextgc
- objectupvalue * openupvalues

9.45.1 Detailed Description

A Morpho virtual machine and its current state.

9.45.2 Member Data Documentation

9.45.2.1 bound

```
size_t svm::bound
```

Graylist for garbage collection

9.45.2.2 frame

```
callframe svm::frame[MORPHO_CALLFRAMESTACKSIZE]
```

The stack

9.45.2.3 globals

```
varray_value svm::globals
```

The current program being executed

9.45.2.4 gray

```
graylist svm::gray
```

Linked list of objects

9.45.2.5 nextgc

```
size_t svm::nextgc
```

Estimated size of bound bytes

9.45.2.6 objects

```
object* svm::objects
```

An error struct that will be filled out when an error occurs

9.45.2.7 openupvalues

```
objectupvalue* svm::openupvalues
```

Next garbage collection threshold

9.45.2.8 sp

```
value* svm::sp
```

The call frame stack

9.45.2.9 stack

```
varray_value svm::stack
```

Global variables

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

- vm/core.h

9.46 syntaxtree Struct Reference

Public Attributes

- varray_syntaxtreenode **tree**
- syntaxtreeindx **entry**

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

- datastructures/syntaxtree.h

9.47 token Struct Reference

```
#include <parse.h>
```

Public Attributes

- tokentype **type**
- const char * start
- unsigned int length
- int line
- int posn

9.47.1 Detailed Description

A token

9.47.2 Member Data Documentation

9.47.2.1 length

```
unsigned int token::length
```

Start of the token

9.47.2.2 line

```
int token::line
```

Its length

9.47.2.3 posn

```
int token::posn
```

Source line

9.47.2.4 start

```
const char* token::start
```

Type of the token

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

- utils/parse.h

9.48 upvalue Struct Reference

```
#include <object.h>
```

Public Attributes

- bool **islocal**
- indx reg

9.48.1 Detailed Description

Each upvalue

9.48.2 Member Data Documentation

9.48.2.1 reg

`indx upvalue::reg`

Set if the upvalue is local to this function

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

- datastructures/object.h

9.49 value Struct Reference

The unboxed value type.

`#include <value.h>`

Public Attributes

- valuetype **type**
- union {
  int **integer**
  double **real**
  bool **boolean**
  struct sobject * **obj**
  } **as**

9.49.1 Detailed Description

The unboxed value type.

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

- datastructures/value.h

Chapter 10

File Documentation

10.1 build.h File Reference

Define constants that choose how Morpho is built.

Macros

- #define **MORPHO_HELPDIRECTORY** "/usr/local/share/morpho/help"
- #define MORPHO_COLORTERMINAL

  Use coloring in output.
- #define MORPHO_STRINGINTERPOLATION

  Support string interpolation.
- #define MORPHO_NEWLINETERMINATORS

  Newlines as statement terminators.
- #define MORPHO_EPS 1e-16

  Enable compatibility with Lox language.
- #define MORPHO_ERRORSTRINGSIZE 255

  Maximum length of a Morpho error string.
- #define MORPHO_INPUTBUFFERDEFAULTSIZE 255

  Default size of input buffer.
- #define MORPHO_MAXIMUMFILENAMELENGTH 255

  Maximum file name length.
- #define MORPHO_CALLFRAMESTACKSIZE 64

  Size of the call frame stack.
- #define MORPHO_MAXARGS 255

  Maximum number of arguments.
- #define MORPHO_COMPUTED_GOTO

  Build Morpho VM with computed gotos.
- #define MORPHO_NAN_BOXING

  Build Morpho VM with small but hacky value type [NaN boxing].
- #define MORPHO_GCINITIAL 1024∗1024;

  Number of bytes to bind before GC first runs.
- #define MORPHO_GCGROWTHFACTOR 2

  Controls how rapidly the GC tries to collect garbage.
- #define MORPHO_MAXIMUMSTACKALLOC 256

Limits size of statically allocated arrays on the C stack.

- #define MORPHO_LINALG_USE_ACCELERATE

  Avoid using global variables (suitable for small programs only)
- #define MORPHO_LINALG_USE_CSPARSE
- #define MORPHO_DEBUG

  Include debugging features.

10.1.1 Detailed Description

Define constants that choose how Morpho is built.

Author

T J Atherton

10.1.2 Macro Definition Documentation

10.1.2.1 MORPHO_EPS

```
#define MORPHO_EPS 1e-16
```

Enable compatibility with Lox language.

Turn off features incompatible with lox Value used to detect zero

10.1.2.2 MORPHO_LINALG_USE_ACCELERATE

```
#define MORPHO_LINALG_USE_ACCELERATE
```

Avoid using global variables (suitable for small programs only)

Use Apple's accelerate library for dense linear algebra

10.1.2.3 MORPHO_LINALG_USE_CSPARSE

```
#define MORPHO_LINALG_USE_CSPARSE
```

Use the LAPACKE library for dense linear algebra Use CSparse for sparse matrix

10.2 builtin/builtin.c File Reference

Morpho built in functions and classes.

```
#include "builtin.h"
#include "common.h"
#include "object.h"
#include "functions.h"
#include "file.h"
#include "matrix.h"
#include "sparse.h"
#include "mesh.h"
```

Macros

- #define **MORPHO_CLASS_METHOD** "class"
- #define **MORPHO_SUPER_METHOD** "superclass"
- #define **MORPHO_SERIALIZE_METHOD** "serialize"
- #define **MORPHO_CLONE_METHOD** "clone"

Functions

- value Object_init (vm *v, int nargs, value *args)
- value Object_class (vm *v, int nargs, value *args)
- value Object_super (vm *v, int nargs, value *args)
- value Object_serialize (vm *v, int nargs, value *args)
- value Object_clone (vm *v, int nargs, value *args)
- **MORPHO_METHOD** (MORPHO_INITIALIZER_METHOD, Object_init, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_CLASS_METHOD, Object_class, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_SUPER_METHOD, Object_super, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_SERIALIZE_METHOD, Object_serialize, BUILTIN_FLAGSEMPTY)
- MORPHO_ENDCLASS value String_length (vm *v, int nargs, value *args)
- MORPHO_ENDCLASS value **dictionary_constructor** (vm *v, int nargs, value *args)
- value Dictionary_getindex (vm *v, int nargs, value *args)
- value Dictionary_setindex (vm *v, int nargs, value *args)
- **MORPHO_METHOD** (MORPHO_GETINDEX_METHOD, Dictionary_getindex, BUILTIN_FLAGSEMPTY)
- MORPHO_ENDCLASS int range_count (objectrange *range)
- value range_iterate (objectrange *range, unsigned int i)
- objectrange * object_newrange (value start, value end, value step)
- value range_constructor (vm *v, int nargs, value *args)
- value Range_enumerate (vm *v, int nargs, value *args)
- void builtin_setveneerclass (objecttype type, value class)

  Sets the veneer class for a particular object type.
- objectclass * builtin_getveneerclass (objecttype type)

  Gets the veneer for a particular object type.
- value builtin_addfunction (char *name, builtinfunction func, builtinfunctionflags flags)
- value builtin_findfunction (value name)
- void builtin_printfunction (objectbuiltinfunction *f)
- value builtin_addclass (char *name, builtinclassentry desc[ ], value superclass)
- value builtin_findclass (value name)
- void builtin_copysymboltable (dictionary *out)
- value builtin_internsymbol (value symbol)
- value builtin_internsymbolascstring (char *symbol)
- void **builtin_bindobjects** (vm *v, unsigned int nobj, value *obj)
- void **builtin_initialize** (void)
- void **builtin_finalize** (void)

Variables

- objectclass * objectveneer [OBJECT_EXTERN+1]

10.2.1 Detailed Description

Morpho built in functions and classes.

Author

T J Atherton

10.2.2 Function Documentation

10.2.2.1 builtin_addclass()

```
value builtin_addclass (
            char * name,
            builtinclassentry desc[],
            value superclass )
```

Defines a built in class

Parameters

in	*name*	the name of the class
in	*desc*	class description; use MORPHO_GETCLASSDEFINITION(name) to obtain this
in	*superclass*	the class's superclass

Returns

the class object

10.2.2.2 builtin_addfunction()

```
value builtin_addfunction (
            char * name,
            builtinfunction func,
            builtinfunctionflags flags )
```

Add a builtin function.

Parameters

name	name of the function
func	the corresponding C function
flags	flags to define the function

Returns

value referring to the objectbuiltinfunction

10.2.2.3 builtin_copysymboltable()

```
void builtin_copysymboltable (
            dictionary * out )
```

Copies the built in symbol table into a new dictionary

10.2.2.4 builtin_findclass()

```
value builtin_findclass (
            value name )
```

Finds a builtin class from its name

10.2.2.5 builtin_findfunction()

```
value builtin_findfunction (
            value name )
```

Finds a builtin function from its name

10.2.2.6 builtin_internsymbol()

```
value builtin_internsymbol (
            value symbol )
```

Interns a given symbol.

10.2.2.7 builtin_internsymbolascstring()

```
value builtin_internsymbolascstring (
            char * symbol )
```

Interns a symbo given as a C string.

10.2.2.8 builtin_printfunction()

```
void builtin_printfunction (
            objectbuiltinfunction * f )
```

Prints a builtin function

10.2.2.9 Dictionary_getindex()

```
value Dictionary_getindex (
            vm * v,
            int nargs,
            value * args )
```

Find a string's length

10.2.2.10 Dictionary_setindex()

```
value Dictionary_setindex (
            vm * v,
            int nargs,
            value * args )
```

Find a string's length

10.2.2.11 Object_class()

```
value Object_class (
            vm * v,
            int nargs,
            value * args )
```

Find the object's class

10.2.2.12 Object_clone()

```
value Object_clone (
            vm * v,
            int nargs,
            value * args )
```

Generic initializer

10.2.2.13 Object_init()

```
value Object_init (
            vm * v,
            int nargs,
            value * args )
```

Generic initializer

10.2.2.14 object_newrange()

```
objectrange* object_newrange (
             value start,
             value end,
             value step )
```

Create a new range. Step may be set to MORPHO_NIL to use the default value of 1

10.2.2.15 Object_serialize()

```
value Object_serialize (
             vm * v,
             int nargs,
             value * args )
```

Generic initializer

10.2.2.16 Object_super()

```
value Object_super (
             vm * v,
             int nargs,
             value * args )
```

Find the object's superclass

10.2.2.17 range_constructor()

```
value range_constructor (
             vm * v,
             int nargs,
             value * args )
```

Constructor function for ranges

10.2.2.18 range_count()

```
MORPHO_ENDCLASS int range_count (
             objectrange * range )
```

Calculate the number of steps in a range

10.2.2.19 Range_enumerate()

```
value Range_enumerate (
             vm * v,
             int nargs,
             value * args )
```

Enumerate members of a range

10.2.2.20 range_iterate()

```
value range_iterate (
            objectrange * range,
            unsigned int i )
```

Find the ith value of a range object

10.2.2.21 String_length()

```
MORPHO_ENDCLASS value String_length (
            vm * v,
            int nargs,
            value * args )
```

Find a string's length

10.2.3 Variable Documentation

10.2.3.1 objectveneer

```
objectclass* objectveneer[OBJECT_EXTERN+1]
```

Core object types can be provided with a 'veneer' class enabling the user to call methods on it, e.g. <string>.length(). This list provides easy access.

10.3 builtin/builtin.h File Reference

Morpho built in functions and classes.

```
#include "object.h"
#include "morpho.h"
```

Classes

- struct objectbuiltinfunction
- struct builtinclassentry

Macros

- #define **BUILTIN_FLAGSEMPTY** 0
- #define MORPHO_GETBUILTINFUNCTION(val) ((objectbuiltinfunction *) MORPHO_GETOBJECT(val))
- #define MORPHO_ISBUILTINFUNCTION(val) object_istype(val, OBJECT_BUILTINFUNCTION)
- #define MORPHO_BEGINCLASS(name) builtinclassentry builtinclass_##name[] = {
- #define **MORPHO_PROPERTY**(label) ((builtinclassentry) { .type=(BUILTIN_PROPERTY), .name=(label), .flags=BUILTIN_FLAGSEMPTY, .function=NULL})
- #define **MORPHO_METHOD**(label, func, flg) ((builtinclassentry) { .type=(BUILTIN_METHOD), .name=(label), .flags=flg, .function=func})
- #define **MORPHO_ENDCLASS**
- #define **MORPHO_GETCLASSDEFINITION**(name) (builtinclass_##name)
- #define MORPHO_GETARG(args, n) (args[n+1])
- #define MORPHO_SELF(args) (args[0])
- #define MORPHO_RAISE(v, err) { morpho_runtimeerror(v, err); return MORPHO_NIL; }
- #define **MORPHO_RAISEVARGS**(v, err, ...)
- #define **OBJECT_CLASSNAME** "Object"
- #define **STRING_CLASSNAME** "String"
- #define **DICTIONARY_CLASSNAME** "Dictionary"
- #define **RANGE_CLASSNAME** "Range"
- #define **RANGE_ARGS** "RngArgs"
- #define **RANGE_ARGS_MSG** "Range expects numerical arguments: a start, an end and an optional step-size."
- #define **ENUMERATE_ARGS** "EnmrtArgs"
- #define **ENUMERATE_ARGS_MSG** "Enumerate method expects a single integer argument."

Typedefs

- typedef unsigned int builtinfunctionflags
- typedef value(* builtinfunction) (vm *v, int nargs, value *args)

Functions

- value builtin_addfunction (char *name, builtinfunction func, builtinfunctionflags flags)
- value builtin_findfunction (value name)
- void builtin_printfunction (objectbuiltinfunction *f)
- value builtin_addclass (char *name, builtinclassentry desc[], value superclass)
- value builtin_findclass (value name)
- void builtin_copysymboltable (dictionary *out)
- value builtin_internsymbol (value symbol)
- value builtin_internsymbolascstring (char *symbol)
- void builtin_setveneerclass (objecttype type, value class)

  Sets the veneer class for a particular object type.
- objectclass * builtin_getveneerclass (objecttype type)

  Gets the veneer for a particular object type.
- void **builtin_bindobjects** (vm *v, unsigned int nobj, value *obj)
- void **builtin_initialize** (void)
- void **builtin_finalize** (void)

10.3.1 Detailed Description

Morpho built in functions and classes.

Author

T J Atherton

10.3.2 Macro Definition Documentation

10.3.2.1 MORPHO_BEGINCLASS

```
#define MORPHO_BEGINCLASS(
              name ) builtinclassentry builtinclass_##name[] = {
```

The following macros help to define a built in class. They should be used outside of any function declaration. To use: MORPHO_BEGINCLASS(Object) - Starts the declaration MORPHO_PROPERTY("test") - Adds a property called "test" to the definition MORPHO_METHOD("init", object_init, BUILTIN_FLAGSEMPTY) - Adds a method called "init" to the definition MORPHO_ENDCLASS - Ends the declaration

10.3.2.2 MORPHO_ENDCLASS

```
#define MORPHO_ENDCLASS
```

Value:

```
, MORPHO_PROPERTY(NULL) \
};
```

10.3.2.3 MORPHO_GETARG

```
#define MORPHO_GETARG(
              args,
              n ) (args[n+1])
```

Macros and functions for built in classes

10.3.2.4 MORPHO_GETBUILTINFUNCTION

```
#define MORPHO_GETBUILTINFUNCTION(
              val ) ((objectbuiltinfunction *) MORPHO_GETOBJECT(val))
```

Gets an objectfunction from a value

10.3.2.5 MORPHO_ISBUILTINFUNCTION

```
#define MORPHO_ISBUILTINFUNCTION(
              val ) object_istype(val, OBJECT_BUILTINFUNCTION)
```

Tests whether an object is a function

10.3.2.6 MORPHO_RAISE

```
#define MORPHO_RAISE(
              v,
              err ) { morpho_runtimeerror(v, err ); return MORPHO_NIL; }
```

Raise an error and return nil

10.3.2.7 MORPHO_RAISEVARGS

```
#define MORPHO_RAISEVARGS(
              v,
              err,
              ...  )
```

Value:

```
{ morpho_runtimeerror(v, err, __VA_ARGS__); \
  return MORPHO_NIL; }
```

10.3.2.8 MORPHO_SELF

```
#define MORPHO_SELF(
              args ) (args[0])
```

This macro gets self

10.3.3 Typedef Documentation

10.3.3.1 builtinfunction

```
typedef value(* builtinfunction) (vm *v, int nargs, value *args)
```

Type of C function that implements a built in Morpho function

10.3.3.2 builtinfunctionflags

```
typedef unsigned int builtinfunctionflags
```

Flags that describe properties of the built in function

10.3.4 Function Documentation

10.3.4.1 builtin_addclass()

```
value builtin_addclass (
            char * name,
            builtinclassentry desc[],
            value superclass )
```

Defines a built in class

Parameters

in	*name*	the name of the class
in	*desc*	class description; use MORPHO_GETCLASSDEFINITION(name) to obtain this
in	*superclass*	the class's superclass

Returns

the class object

10.3.4.2 builtin_addfunction()

```
value builtin_addfunction (
            char * name,
            builtinfunction func,
            builtinfunctionflags flags )
```

Add a builtin function.

Parameters

name	name of the function
func	the corresponding C function
flags	flags to define the function

Returns

value referring to the objectbuiltinfunction

10.3.4.3 builtin_copysymboltable()

```
void builtin_copysymboltable (
            dictionary * out )
```

Copies the built in symbol table into a new dictionary

10.3.4.4 builtin_findclass()

```
value builtin_findclass (
            value name )
```

Finds a builtin class from its name

10.3.4.5 builtin_findfunction()

```
value builtin_findfunction (
            value name )
```

Finds a builtin function from its name

10.3.4.6 builtin_internsymbol()

```
value builtin_internsymbol (
            value symbol )
```

Interns a given symbol.

10.3.4.7 builtin_internsymbolascstring()

```
value builtin_internsymbolascstring (
            char * symbol )
```

Interns a symbo given as a C string.

10.3.4.8 builtin_printfunction()

```
void builtin_printfunction (
            objectbuiltinfunction * f )
```

Prints a builtin function

10.4 builtin/file.c File Reference

Built in class to provide file input and output.

```
#include "file.h"
#include "builtin.h"
#include "error.h"
#include "object.h"
#include "morpho.h"
#include "common.h"
#include <stdio.h>
#include <limits.h>
#include <errno.h>
```

Functions

- bool **file_getsize** (FILE *f, size_t *s)
- FILE * **file_getfile** (value obj)
- void **file_setfile** (value obj, FILE *f)
- value File_init (vm *v, int nargs, value *args)
- value File_close (vm *v, int nargs, value *args)
- int file_readlineintovarray (FILE *f, varray_char *string)
- bool file_readintovarray (FILE *f, varray_char *string)
- value file_readlineusingvarray (FILE *f, varray_char *string)
- value File_readline (vm *v, int nargs, value *args)
- value File_lines (vm *v, int nargs, value *args)
- value File_readchar (vm *v, int nargs, value *args)
- value File_write (vm *v, int nargs, value *args)
- value File_eof (vm *v, int nargs, value *args)
- value File_free (vm *v, int nargs, value *args)
- **MORPHO_METHOD** (MORPHO_INITIALIZER_METHOD, File_init, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (FILE_CLOSE, File_close, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (FILE_LINES, File_lines, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (FILE_READLINE, File_readline, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (FILE_READCHAR, File_readchar, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (FILE_WRITE, File_write, BUILTIN_FLAGSEMPTY)
- MORPHO_ENDCLASS void **file_initialize** (void)

10.4.1 Detailed Description

Built in class to provide file input and output.

Author

T J Atherton

10.4.2 Function Documentation

10.4.2.1 File_close()

```
value File_close (
            vm * v,
            int nargs,
            value * args )
```

Close a file

10.4.2.2 File_eof()

```
value File_eof (
            vm * v,
            int nargs,
            value * args )
```

Detects whether we're at the end of the file

10.4.2.3 File_free()

```
value File_free (
            vm * v,
            int nargs,
            value * args )
```

Called when the file object is freed

10.4.2.4 File_init()

```
value File_init (
            vm * v,
            int nargs,
            value * args )
```

Initializer In: 1. a file name

1. (optional) a string giving the requested status, e.g. "wr+"

10.4.2.5 File_lines()

```
value File_lines (
            vm * v,
            int nargs,
            value * args )
```

Get the contents of a file as an array

10.4.2.6 File_readchar()

```
value File_readchar (
            vm * v,
            int nargs,
            value * args )
```

Reads a single character

10.4.2.7 file_readintovarray()

```
bool file_readintovarray (
            FILE * f,
            varray_char * string )
```

Reads a whole file into a buffer

10.4.2.8 File_readline()

```
value File_readline (
            vm * v,
            int nargs,
            value * args )
```

Read a line

10.4.2.9 file_readlineintovarray()

```
int file_readlineintovarray (
            FILE * f,
            varray_char * string )
```

Reads a line using a given buffer

10.4.2.10 file_readlineusingvarray()

```
value file_readlineusingvarray (
            FILE * f,
            varray_char * string )
```

Reads a line using a given buffer

10.4.2.11 File_write()

```
value File_write (
            vm * v,
            int nargs,
            value * args )
```

Write to a file

10.5 builtin/file.h File Reference

Built in class to provide file input and output.

```
#include <stdio.h>
#include "morpho.h"
```

Macros

- #define **FILE_CLASSNAME** "File"
- #define **FILE_FILEPROPERTY** "@file"
- #define **FILE_CLOSE** "close"
- #define **FILE_LINES** "lines"
- #define **FILE_READLINE** "readline"
- #define **FILE_READCHAR** "readchar"
- #define **FILE_WRITE** "write"
- #define **FILE_EOF** "eof"
- #define **FILE_READMODE** "read"
- #define **FILE_WRITEMODE** "write"
- #define **FILE_APPENDMODE** "append"
- #define **FILE_OPENFAILED** "FlOpnFld"
- #define **FILE_OPENFAILED_MSG** "Couldn't open file '%s'."
- #define **FILE_FILENAMEARG** "FlNmArgs"
- #define **FILE_FILENAMEARG_MSG** "First argument to File must be a filename."
- #define **FILE_NEEDSFILENAME** "FlNmMssng"
- #define **FILE_NEEDSFILENAME_MSG** "Filename missing."
- #define **FILE_MODE** "FlMode"
- #define **FILE_MODE_MSG** "Second argument to File should be 'read', 'write' or 'append'."
- #define **FILE_WRITEARGS** "FlWrtArgs"
- #define **FILE_WRITEARGS_MSG** "Arguments to File.write must be strings."
- #define **FILE_WRITEFAIL** "FlWrtFld"
- #define **FILE_WRITEFAIL_MSG** "Write to file failed."

Functions

- bool **file_getsize** (FILE *f, size_t *s)
- int file_readlineintovarray (FILE *f, varray_char *string)
- bool file_readintovarray (FILE *f, varray_char *string)
- void **file_initialize** (void)

10.5.1 Detailed Description

Built in class to provide file input and output.

Author

T J Atherton

10.5.2 Function Documentation

10.5.2.1 file_readintovarray()

```
bool file_readintovarray (
            FILE * f,
            varray_char * string )
```

Reads a whole file into a buffer

10.5.2.2 file_readlineintovarray()

```
int file_readlineintovarray (
            FILE * f,
            varray_char * string )
```

Reads a line using a given buffer

10.6 builtin/functions.c File Reference

Built in functions.

```
#include <time.h>
#include <stdlib.h>
#include "functions.h"
#include "random.h"
#include "builtin.h"
```

Functions

- value builtin_random (vm *v, int nargs, value *args)
- value builtin_exp (vm *v, int nargs, value *args)
- value builtin_randomint (vm *v, int nargs, value *args)
- value builtin_system (vm *v, int nargs, value *args)
- value builtin_clock (vm *v, int nargs, value *args)
- void **functions_initialize** (void)

10.6.1 Detailed Description

Built in functions.

Author

T J Atherton

10.6.2 Function Documentation

10.6.2.1 builtin_clock()

```
value builtin_clock (
            vm * v,
            int nargs,
            value * args )
```

Clock

10.6.2.2 builtin_exp()

```
value builtin_exp (
            vm * v,
            int nargs,
            value * args )
```

Exponential function

10.6.2.3 builtin_random()

```
value builtin_random (
            vm * v,
            int nargs,
            value * args )
```

Generate a random float between 0 and 1

10.6.2.4 builtin_randomint()

```
value builtin_randomint (
            vm * v,
            int nargs,
            value * args )
```

Generate a random integer with a bound. Efficient and unbiased algorithm from: https://www.↩ pcg-random.org/posts/bounded-rands.html

10.6.2.5 builtin_system()

```
value builtin_system (
            vm * v,
            int nargs,
            value * args )
```

Call the operating system

10.7 builtin/functions.h File Reference

Built in functions.

```
#include <stdio.h>
```

Macros

- #define **FUNCTION_RANDOM** "random"
- #define **FUNCTION_RANDOMINT** "randomint"
- #define **FUNCTION_EXP** "exp"
- #define **FUNCTION_CLOCK** "clock"
- #define **FUNCTION_SYSTEM** "system"

Functions

- void **functions_initialize** (void)

10.7.1 Detailed Description

Built in functions.

Author

T J Atherton

10.8 datastructures/dictionary.h File Reference

Dictionary (hashtable) data structure.

```
#include "value.h"
```

Classes

- struct dictionaryentry

  A single dictionary entry.
- struct dictionary

  dictionary data structure that maps keys to values

Macros

- #define **HASH_EMPTY** 0

Typedefs

- typedef uint32_t **hash**

Functions

- void dictionary_init (dictionary *dict)

  Initializes a dictionary.
- void dictionary_clear (dictionary *dict)

  Clears a dictionary structure, freeing attached memory.
- void dictionary_freecontents (dictionary *dict, bool freekeys, bool freevals)

  Frees a dictionary's contents.
- bool dictionary_insert (dictionary *dict, value key, value val)

  Inserts a value in a hashtable given a key.
- bool dictionary_insertintern (dictionary *dict, value key, value val)

  Inserts a value in a hashtable given a key, assuming the key has been interned.
- value dictionary_intern (dictionary *dict, value key)

  Interns a new key @detail Looks to see if a similar key [one that passes MORPHO_ISEQUAL] is already in the dictionary, and if so returns it. Otherwise, inserts this key into the dictionary.
- bool dictionary_get (dictionary *dict, value key, value *val)

  Retrieves a value from a dictionary given a key.
- bool dictionary_getintern (dictionary *dict, value key, value *val)

  Retrieves a value from a dictionary given a key assuming that key has been interned.
- bool dictionary_remove (dictionary *dict, value key)

  Removes a key from a dictionary given a key.
- bool dictionary_copy (dictionary *src, dictionary *dest)

  Copies the entries of one dictionary to another.

10.8.1 Detailed Description

Dictionary (hashtable) data structure.

Author

T J Atherton

10.8.2 Function Documentation

10.8.2.1 dictionary_clear()

```
void dictionary_clear (
            dictionary * dict )
```

Clears a dictionary structure, freeing attached memory.

Parameters

dict	the dictionary to clear

Warning

This doens't free keys or values in the dictionary.

10.8.2.2 dictionary_freecontents()

```
void dictionary_freecontents (
            dictionary * dict,
            bool freekeys,
            bool freevals )
```

Frees a dictionary's contents.

Parameters

dict	the dictionary to clear
freekeys	whether to free the keys
freevals	whether to free the vals

10.8.2.3 dictionary_get()

```
bool dictionary_get (
            dictionary * dict,
            value key,
            value * val )
```

Retrieves a value from a dictionary given a key.

Parameters

in	*dict*	the dictionary to get
in	*key*	key to locate
out	*val*	Stores the result in this value if found.

Returns

true if found, false otherwise

10.8.2.4 dictionary_getintern()

```
bool dictionary_getintern (
            dictionary * dict,
```

```
value key,
value * val )
```

Retrieves a value from a dictionary given a key assuming that key has been interned.

Parameters

in	*dict*	the dictionary to get
in	*key*	key to locate
out	*val*	Stores the result in this value if found.

Returns

true if found, false otherwise

10.8.2.5 dictionary_init()

```
void dictionary_init (
            dictionary * dict )
```

Initializes a dictionary.

Parameters

dict	the dictionary to initialize

10.8.2.6 dictionary_insert()

```
bool dictionary_insert (
            dictionary * dict,
            value key,
            value val )
```

Inserts a value in a hashtable given a key.

Parameters

in	*dict*	the dictionary
in	*key*	key to insert
in	*val*	value to insert

Returns

true if successful, false otherwise

Warning

If an entry already exists, it is overwritten. Caller should check for existing keys if this is necessary.

10.8.2.7 dictionary_insertintern()

```
bool dictionary_insertintern (
            dictionary * dict,
            value key,
            value val )
```

Inserts a value in a hashtable given a key, assuming the key has been interned.

Parameters

in	*dict*	the dictionary
in	*key*	key to insert
in	*val*	value to insert

Returns

true if successful, false otherwise

Warning

If an entry already exists, it is overwritten. Caller should check for existing keys if this is necessary.

10.8.2.8 dictionary_intern()

```
value dictionary_intern (
            dictionary * dict,
            value key )
```

Interns a new key @detail Looks to see if a similar key [one that passes MORPHO_ISEQUAL] is already in the dictionary, and if so returns it. Otherwise, inserts this key into the dictionary.

Parameters

in	*dict*	the dictionary
in	*key*	a new key to intern

Returns

the internalized key, or MORPHO_NIL on failure.

10.8.2.9 dictionary_remove()

```
bool dictionary_remove (
            dictionary * dict,
            value key )
```

Removes a key from a dictionary given a key.

Parameters

in	*dict*	the dictionary to initialize
in	*key*	key to remove

Returns

true if the key was found, false otherwise

10.9 datastructures/matrix.c File Reference

Veneer class over the objectmatrix type that interfaces with blas and lapack.

```
#include <string.h>
#include "object.h"
#include "matrix.h"
#include "sparse.h"
#include "morpho.h"
#include "builtin.h"
#include "common.h"
```

Functions

- objectmatrix * object_newmatrix (unsigned int nrows, unsigned int ncols, bool zero)
- bool matrix_getarraydimensions (objectarray *array, unsigned int dim[], unsigned int maxdim, unsigned int *ndim)
- value matrix_getarrayelement (objectarray *array, unsigned int ndim, unsigned int *indx)
- objectmatrix * object_matrixfromarray (objectarray *array)
- bool matrix_setelement (objectmatrix *matrix, unsigned int row, unsigned int col, double value)

  Sets a matrix element.
- bool matrix_getelement (objectmatrix *matrix, unsigned int row, unsigned int col, double *value)

  Gets a matrix element.
- objectmatrixerror matrix_add (objectmatrix *a, objectmatrix *b, objectmatrix *out)
- objectmatrixerror matrix_sub (objectmatrix *a, objectmatrix *b, objectmatrix *out)
- objectmatrixerror matrix_mul (objectmatrix *a, objectmatrix *b, objectmatrix *out)
- objectmatrixerror matrix_divs (objectmatrix *a, objectmatrix *b, objectmatrix *out)
- objectmatrixerror matrix_divl (objectmatrix *a, objectmatrix *b, objectmatrix *out)
- objectmatrixerror matrix_transpose (objectmatrix *a, objectmatrix *out)
- objectmatrixerror matrix_trace (objectmatrix *a, double *out)
- value matrix_constructor (vm *v, int nargs, value *args)
- value **Matrix_getindex** (vm *v, int nargs, value *args)
- value **Matrix_setindex** (vm *v, int nargs, value *args)
- value **Matrix_print** (vm *v, int nargs, value *args)
- value **Matrix_add** (vm *v, int nargs, value *args)
- value **Matrix_sub** (vm *v, int nargs, value *args)
- value **Matrix_mul** (vm *v, int nargs, value *args)
- value **Matrix_div** (vm *v, int nargs, value *args)
- value **Matrix_transpose** (vm *v, int nargs, value *args)
- value **Matrix_trace** (vm *v, int nargs, value *args)
- value **Matrix_enumerate** (vm *v, int nargs, value *args)

- **MORPHO_METHOD** (MORPHO_GETINDEX_METHOD, Matrix_getindex, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_SETINDEX_METHOD, Matrix_setindex, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_PRINT_METHOD, Matrix_print, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_ADD_METHOD, Matrix_add, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_SUB_METHOD, Matrix_sub, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_MUL_METHOD, Matrix_mul, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_DIV_METHOD, Matrix_div, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MATRIX_TRANSPOSE_METHOD, Matrix_transpose, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MATRIX_TRACE_METHOD, Matrix_trace, BUILTIN_FLAGSEMPTY)
- MORPHO_ENDCLASS void **matrix_initialize** (void)

10.9.1 Detailed Description

Veneer class over the objectmatrix type that interfaces with blas and lapack.

Author

T J Atherton

10.9.2 Function Documentation

10.9.2.1 matrix_add()

```
objectmatrixerror matrix_add (
            objectmatrix * a,
            objectmatrix * b,
            objectmatrix * out )
```

Performs a + b -> out.

10.9.2.2 matrix_constructor()

```
value matrix_constructor (
            vm * v,
            int nargs,
            value * args )
```

Constructs a Matrix object

10.9.2.3 matrix_divl()

```
objectmatrixerror matrix_divl (
            objectmatrix * a,
            objectmatrix * b,
            objectmatrix * out )
```

Solves the system a.x = b for large matrices (test with MATRIX_ISSMALL)

10.9.2.4 matrix_divs()

```
objectmatrixerror matrix_divs (
            objectmatrix * a,
            objectmatrix * b,
            objectmatrix * out )
```

Solves the system a.x = b for small matrices (test with MATRIX_ISSMALL)

Warning

Uses the C stack for storage, which avoids malloc but can cause stack overflow

10.9.2.5 matrix_getarraydimensions()

```
bool matrix_getarraydimensions (
            objectarray * array,
            unsigned int dim[],
            unsigned int maxdim,
            unsigned int * ndim )
```

Recurses into an objectarray to find the dimensions of the array and all child arrays

Parameters

in	*array*	- to search
out	*dim*	- array of dimensions to be filled out (must be zero'd before initial call)
in	*maxdim*	- maximum number of dimensions
out	*ndim*	- number of dimensions of the array

10.9.2.6 matrix_getarrayelement()

```
value matrix_getarrayelement (
            objectarray * array,
            unsigned int ndim,
            unsigned int * indx )
```

Looks up an array element recursively if necessary

10.9.2.7 matrix_getelement()

```
bool matrix_getelement (
            objectmatrix * matrix,
            unsigned int row,
            unsigned int col,
            double * value )
```

Gets a matrix element.

Returns

true if the element is in the range of the matrix, false otherwise

10.9.2.8 matrix_mul()

```
objectmatrixerror matrix_mul (
            objectmatrix * a,
            objectmatrix * b,
            objectmatrix * out )
```

Performs a * b -> out

10.9.2.9 matrix_setelement()

```
bool matrix_setelement (
            objectmatrix * matrix,
            unsigned int row,
            unsigned int col,
            double value )
```

Sets a matrix element.

Returns

true if the element is in the range of the matrix, false otherwise

10.9.2.10 matrix_sub()

```
objectmatrixerror matrix_sub (
            objectmatrix * a,
            objectmatrix * b,
            objectmatrix * out )
```

Performs a - b -> out

10.9.2.11 matrix_trace()

```
objectmatrixerror matrix_trace (
            objectmatrix * a,
            double * out )
```

Calculate the trace of a matrix

10.9.2.12 matrix_transpose()

```
objectmatrixerror matrix_transpose (
            objectmatrix * a,
            objectmatrix * out )
```

Transpose a matrix

10.9.2.13 object_matrixfromarray()

```
objectmatrix* object_matrixfromarray (
            objectarray * array )
```

Creates a new array from a list of values

10.9.2.14 object_newmatrix()

```
objectmatrix* object_newmatrix (
            unsigned int nrows,
            unsigned int ncols,
            bool zero )
```

Creates a matrix object

10.10 datastructures/matrix.h File Reference

Veneer class over the objectmatrix type that interfaces with blas and lapack.

```
#include <stdio.h>
#include <cblas.h>
#include <lapacke.h>
```

Macros

- #define MORPHO_LINALG_USE_LAPACKE
- #define **MATRIX_LAPACK_PRESENT**
- #define **MATRIX_CLASSNAME** "Matrix"
- #define **MATRIX_TRANSPOSE_METHOD** "transpose"
- #define **MATRIX_TRACE_METHOD** "trace"
- #define **MATRIX_DET_METHOD** "det"
- #define **MATRIX_EIGENVALUES_METHOD** "eigenvalues"
- #define **MATRIX_EIGENSYSTEM_METHOD** "eigensystem"
- #define **MATRIX_INDICESOUTSIDEBOUNDS** "MtrxBnds"
- #define **MATRIX_INDICESOUTSIDEBOUNDS_MSG** "Matrix index out of bounds."
- #define **MATRIX_INVLDINDICES** "MtrxInvldIndx"
- #define **MATRIX_INVLDINDICES_MSG** "Matrix indices must be numerical."
- #define **MATRIX_CONSTRUCTOR** "MtrxCns"

- #define **MATRIX_CONSTRUCTOR_MSG** "Matrix() should be called either with dimensions or an array initializer."
- #define **MATRIX_INVLDARRAYINIT** "MtrxInvldInit"
- #define **MATRIX_INVLDARRAYINIT_MSG** "Array initializer passed to Matrix() must be a 1 or 2 dimensional array."
- #define **MATRIX_ARITHARGS** "MtrxInvldArg"
- #define **MATRIX_ARITHARGS_MSG** "Matrix arithmetic methods expect a matrix or number as their argument."
- #define **MATRIX_INCOMPATIBLEMATRICES** "MtrxIncmptbl"
- #define **MATRIX_INCOMPATIBLEMATRICES_MSG** "Matrices have incompatible shape."
- #define **MATRIX_SINGULAR** "MtrxSnglr"
- #define **MATRIX_SINGULAR_MSG** "Matrix is singular."
- #define **MATRIX_NOTSQ** "MtrxNtSq"
- #define **MATRIX_NOTSQ_MSG** "Matrix is not square."
- #define MATRIX_ISSMALL(m) (m->nrows*m->ncols<MORPHO_MAXIMUMSTACKALLOC)

Enumerations

- enum **objectmatrixerror** {
  MATRIX_OK, **MATRIX_INCMPTBLDIM**, **MATRIX_SING**, **MATRIX_INVLD**,
  MATRIX_NSQ, **MATRIX_ALLOC** }

Functions

- bool matrix_getarraydimensions (objectarray *array, unsigned int dim[], unsigned int maxdim, unsigned int *ndim)
- value matrix_getarrayelement (objectarray *array, unsigned int ndim, unsigned int *indx)
- bool matrix_setelement (objectmatrix *matrix, unsigned int row, unsigned int col, double value)

  Sets a matrix element.
- bool matrix_getelement (objectmatrix *matrix, unsigned int row, unsigned int col, double *value)

  Gets a matrix element.
- objectmatrixerror matrix_add (objectmatrix *a, objectmatrix *b, objectmatrix *out)
- objectmatrixerror matrix_sub (objectmatrix *a, objectmatrix *b, objectmatrix *out)
- objectmatrixerror matrix_mul (objectmatrix *a, objectmatrix *b, objectmatrix *out)
- objectmatrixerror matrix_divs (objectmatrix *a, objectmatrix *b, objectmatrix *out)
- objectmatrixerror matrix_divl (objectmatrix *a, objectmatrix *b, objectmatrix *out)
- objectmatrixerror matrix_transpose (objectmatrix *a, objectmatrix *out)
- objectmatrixerror matrix_trace (objectmatrix *a, double *out)
- void **matrix_initialize** (void)

10.10.1 Detailed Description

Veneer class over the objectmatrix type that interfaces with blas and lapack.

Author

T J Atherton

10.10.2 Macro Definition Documentation

10.10.2.1 MATRIX_ISSMALL

```
#define MATRIX_ISSMALL(
             m ) (m->nrows*m->ncols<MORPHO_MAXIMUMSTACKALLOC)
```

Macro to decide if a matrix is 'small' or 'large' and hence static or dynamic allocation should be used.

10.10.2.2 MORPHO_LINALG_USE_LAPACKE

```
#define MORPHO_LINALG_USE_LAPACKE
```

Use Apple's Accelerate library for LAPACK and BLAS Otherwise, use LAPACKE

10.10.3 Function Documentation

10.10.3.1 matrix_add()

```
objectmatrixerror matrix_add (
             objectmatrix * a,
             objectmatrix * b,
             objectmatrix * out )
```

Performs a + b -> out.

10.10.3.2 matrix_divl()

```
objectmatrixerror matrix_divl (
             objectmatrix * a,
             objectmatrix * b,
             objectmatrix * out )
```

Solves the system a.x = b for large matrices (test with MATRIX_ISSMALL)

10.10.3.3 matrix_divs()

```
objectmatrixerror matrix_divs (
             objectmatrix * a,
             objectmatrix * b,
             objectmatrix * out )
```

Solves the system a.x = b for small matrices (test with MATRIX_ISSMALL)

Warning

Uses the C stack for storage, which avoids malloc but can cause stack overflow

10.10.3.4 matrix_getarraydimensions()

```
bool matrix_getarraydimensions (
             objectarray * array,
             unsigned int dim[],
             unsigned int maxdim,
             unsigned int * ndim )
```

Recurses into an objectarray to find the dimensions of the array and all child arrays

Parameters

in	*array*	- to search
out	*dim*	- array of dimensions to be filled out (must be zero'd before initial call)
in	*maxdim*	- maximum number of dimensions
out	*ndim*	- number of dimensions of the array

10.10.3.5 matrix_getarrayelement()

```
value matrix_getarrayelement (
            objectarray * array,
            unsigned int ndim,
            unsigned int * indx )
```

Looks up an array element recursively if necessary

10.10.3.6 matrix_getelement()

```
bool matrix_getelement (
            objectmatrix * matrix,
            unsigned int row,
            unsigned int col,
            double * value )
```

Gets a matrix element.

Returns

true if the element is in the range of the matrix, false otherwise

10.10.3.7 matrix_mul()

```
objectmatrixerror matrix_mul (
            objectmatrix * a,
            objectmatrix * b,
            objectmatrix * out )
```

Performs a * b -> out

10.10.3.8 matrix_setelement()

```
bool matrix_setelement (
            objectmatrix * matrix,
            unsigned int row,
            unsigned int col,
            double value )
```

Sets a matrix element.

Returns

true if the element is in the range of the matrix, false otherwise

10.10.3.9 matrix_sub()

```
objectmatrixerror matrix_sub (
            objectmatrix * a,
            objectmatrix * b,
            objectmatrix * out )
```

Performs a - b -> out

10.10.3.10 matrix_trace()

```
objectmatrixerror matrix_trace (
            objectmatrix * a,
            double * out )
```

Calculate the trace of a matrix

10.10.3.11 matrix_transpose()

```
objectmatrixerror matrix_transpose (
            objectmatrix * a,
            objectmatrix * out )
```

Transpose a matrix

10.11 datastructures/object.c File Reference

Provide functionality for extended and mutable data types.

```
#include <string.h>
#include <stdio.h>
#include "value.h"
#include "object.h"
#include "builtin.h"
#include "memory.h"
#include "error.h"
#include "sparse.h"
#include "common.h"
```

Functions

- void object_init (object *obj, objecttype type)

  Initializes an object to be a certain type.
- void object_free (object *obj)
- object * object_new (size_t size, objecttype type)

  Allocates an object.
- value object_stringfromcstring (const char *in, size_t length)

  Creates a string from an existing character array with given length.
- value object_stringfromvarraychar (varray_char *in)

  Converts a varray_char into a string.
- value **object_clonestring** (value val)
- value object_concatenatestring (value a, value b)

  Concatenates strings together.
- **DEFINE_VARRAY** (upvalue, upvalue)
- **DEFINE_VARRAY** (varray_upvalue, varray_upvalue)
- void object_functioninit (objectfunction *func)

  Initializes a new function.
- void object_functionclear (objectfunction *func)

  Clears a function.
- objectfunction * object_newfunction (indx entry, value name, objectfunction *parent, unsigned int nargs)

  Creates a new function.
- objectfunction * object_getfunctionparent (objectfunction *func)
- value object_getfunctionname (objectfunction *func)
- varray_value * object_functiongetconstanttable (objectfunction *func)
- bool object_functionaddprototype (objectfunction *func, varray_upvalue *v, indx *ix)
- void **object_closureinit** (objectclosure *c)
- objectclosure * object_newclosure (objectfunction *sf, objectfunction *func, indx np)

  Creates a new closure.
- void object_upvalueinit (objectupvalue *c)
- objectupvalue * object_newupvalue (value *reg)
- objectclass * **object_newclass** (value name)
- objectinstance * **object_newinstance** (objectclass *klass)
- bool **objectinstance_setproperty** (objectinstance *obj, value key, value val)
- bool **objectinstance_getproperty** (objectinstance *obj, value key, value *val)
- objectinvocation * object_newinvocation (value receiver, value method)
- objectdictionary * object_newdictionary (void)
- dictionary * object_dictionary (objectdictionary *dict)
- void object_arrayinit (objectarray *array, unsigned int ndim, unsigned int *dim)
- objectarray * object_newarray (unsigned int ndim, unsigned int *dim)

  Creates an array object.
- objectarray * object_arrayfromlist (unsigned int n, value *v)
- objectarray * object_arrayfromvarrayvalue (varray_value *v)
- bool object_arrayvaluestoindices (unsigned int ndim, value *indx, unsigned int *iout)
- objectarray * object_arrayfromvalueindices (unsigned int ndim, value *dim)
- bool object_arrayindicestoelement (objectarray *array, unsigned int ndim, unsigned int *indx, unsigned int *ixout)

  Calculates the correct element from a set of array indices.
- objectarrayerror object_getarrayelement (objectarray *array, unsigned int ndim, value *indx, value *out)
- objectarrayerror object_setarrayelement (objectarray *array, unsigned int ndim, value *indx, value set)
- void object_print (value v)
- void object_printtobuffer (value v, varray_char *buffer)
- size_t object_size (object *obj)

10.11.1 Detailed Description

Provide functionality for extended and mutable data types.

Author

T J Atherton

10.11.2 Function Documentation

10.11.2.1 object_arrayfromlist()

```
objectarray* object_arrayfromlist (
            unsigned int n,
            value * v )
```

Creates a new 1D array from a list of values

10.11.2.2 object_arrayfromvalueindices()

```
objectarray* object_arrayfromvalueindices (
            unsigned int ndim,
            value * dim )
```

Creates a new array object with the indices given as a list of values

10.11.2.3 object_arrayfromvarrayvalue()

```
objectarray* object_arrayfromvarrayvalue (
            varray_value * v )
```

Creates a new 1D array from a list of varray_value

10.11.2.4 object_arrayindicestoelement()

```
bool object_arrayindicestoelement (
            objectarray * array,
            unsigned int ndim,
            unsigned int * indx,
            unsigned int * ixout )
```

Calculates the correct element from a set of array indices.

Parameters

in	*array*	- the array
in	*ndim*	- number of dimensions
in	*indx*	- list of indices
out	*ixout*	- the element number to use

Returns

true on success, false if indices are out of bounds

10.11.2.5 object_arrayinit()

```
void object_arrayinit (
            objectarray * array,
            unsigned int ndim,
            unsigned int * dim )
```

Initializes an array given the size

10.11.2.6 object_arrayvaluestoindices()

```
bool object_arrayvaluestoindices (
            unsigned int ndim,
            value * indx,
            unsigned int * iout )
```

Converts a list of indices into a list of unsigned ints

Parameters

ndim	- number of dimensions
indx	- the indices to evaluate
iout	- the indices as integers

Returns

true on success, or false if an unexpected type was encountered

10.11.2.7 object_concatenatestring()

```
value object_concatenatestring (
            value a,
            value b )
```

Concatenates strings together.

Parameters

a	first string
b	second string

Returns

the object (as a value) which will be MORPHO_NIL on failure

10.11.2.8 object_dictionary()

```
dictionary* object_dictionary (
            objectdictionary * dict )
```

Extracts the dictionary from an objectdictionary.

10.11.2.9 object_free()

```
void object_free (
            object * obj )
```

Frees an object

10.11.2.10 object_functionaddprototype()

```
bool object_functionaddprototype (
            objectfunction * func,
            varray_upvalue * v,
            indx * ix )
```

Adds an upvalue prototype to a function

Parameters

in	*func*	function object to add to
in	*v*	a varray of upvalues that will be copied into the function definition.
out	*ix*	index of the closure created

Returns

true on success

10.11.2.11 object_functionclear()

```
void object_functionclear (
            objectfunction * func )
```

Clears a function.

Clear the upvalue prototypes

10.11.2.12 object_functiongetconstanttable()

```
varray_value* object_functiongetconstanttable (
              objectfunction * func )
```

Gets the constant table associated with a function

10.11.2.13 object_getarrayelement()

```
objectarrayerror object_getarrayelement (
              objectarray * array,
              unsigned int ndim,
              value * indx,
              value * out )
```

Gets an array element

10.11.2.14 object_getfunctionname()

```
value object_getfunctionname (
              objectfunction * func )
```

Gets the name of a function

10.11.2.15 object_getfunctionparent()

```
objectfunction* object_getfunctionparent (
              objectfunction * func )
```

Gets the parent of a function

10.11.2.16 object_init()

```
void object_init (
              object * obj,
              objecttype type )
```

Initializes an object to be a certain type.

Parameters

obj	object to initialize
type	type to initialize with

10.11.2.17 object_new()

```
object* object_new (
            size_t size,
            objecttype type )
```

Allocates an object.

Parameters

size	size of memory to reserve
type	type to initialize with

10.11.2.18 object_newarray()

```
objectarray* object_newarray (
            unsigned int ndim,
            unsigned int * dim )
```

Creates an array object.

Arrays are stored in memory as follows: objectarray structure with flexible array member value value [0..dim-1] the dimensions of the array value [dim..] array elements in column major order, i.e. the matrix [[1, 2], [3, 4]] is stored as: <structure> // the structure 2, 2, // the dimensions 1, 3, 2, 4 // the elements in column major order

10.11.2.19 object_newclosure()

```
objectclosure* object_newclosure (
            objectfunction * sf,
            objectfunction * func,
            indx np )
```

Creates a new closure.

Parameters

sf	the objectfunction of the current environment
func	a function object to enclose
np	the prototype number to use

10.11.2.20 object_newdictionary()

```
objectdictionary* object_newdictionary (
            void  )
```

Creates a new dictionary

10.11.2.21 object_newinvocation()

```
objectinvocation* object_newinvocation (
            value receiver,
            value method )
```

Create a new invocation

10.11.2.22 object_newupvalue()

```
objectupvalue* object_newupvalue (
            value * reg )
```

Creates a new upvalue for the register pointed to by reg.

10.11.2.23 object_print()

```
void object_print (
            value v )
```

Prints an object

10.11.2.24 object_printtobuffer()

```
void object_printtobuffer (
            value v,
            varray_char * buffer )
```

Prints an object to a string buffer

Parameters

in	*v*	Object to convert to a buffer
in	*buffer*	Buffer to output to

10.11.2.25 object_setarrayelement()

```
objectarrayerror object_setarrayelement (
            objectarray * array,
            unsigned int ndim,
            value * indx,
            value set )
```

Sets an array element

10.11.2.26 object_size()

```
size_t object_size (
            object * obj )
```

Gets the total size of an object

10.11.2.27 object_stringfromcstring()

```
value object_stringfromcstring (
            const char * in,
            size_t length )
```

Creates a string from an existing character array with given length.

Parameters

in	the string to copy
length	length of string to copy

Returns

the object (as a value) which will be MORPHO_NIL on failure

10.11.2.28 object_stringfromvarraychar()

```
value object_stringfromvarraychar (
            varray_char * in )
```

Converts a varray_char into a string.

Parameters

in	the varray to convert

Returns

the object (as a value) which will be MORPHO_NIL on failure

10.11.2.29 object_upvalueinit()

```
void object_upvalueinit (
            objectupvalue * c )
```

Initializes a new upvalue object.

10.12 datastructures/object.h File Reference

Provide functionality for extended and mutable data types.

```
#include <stddef.h>
#include "value.h"
#include "dictionary.h"
```

Classes

- struct sobject
- struct objectstring
- struct upvalue
- struct sobjectfunction
- struct sobjectupvalue
- struct objectclosure
- struct sobjectclass
- struct objectinstance
- struct objectinvocation
- struct objectrange
- struct objectdictionary
- struct objectarray
- struct objectmatrix
- struct objectdokkey
- struct sparsedok
- struct sparseccs
- struct objectsparse
- struct objectmesh

Macros

- #define MORPHO_GETOBJECTTYPE(val) (MORPHO_GETOBJECT(val)->type)
- #define MORPHO_GETOBJECTHASH(val) (MORPHO_GETOBJECT(val)->hsh)
- #define MORPHO_SETOBJECTHASH(val, newhash) (MORPHO_GETOBJECT(val)->hsh = newhash)
- #define **MORPHO_GETSTRING**(val) ((objectstring *) MORPHO_GETOBJECT(val))
- #define **MORPHO_GETCSTRING**(val) (((objectstring *) MORPHO_GETOBJECT(val))->string)
- #define **MORPHO_GETSTRINGLENGTH**(val) (((objectstring *) MORPHO_GETOBJECT(val))->length)
- #define MORPHO_ISSTRING(val) object_istype(val, OBJECT_STRING)
- #define MORPHO_STATICSTRING(cstring) { .obj.type=OBJECT_STRING, .obj.status=OBJECT_ISUNM↩ ANAGED, .obj.next=NULL, .string=cstring, .length=strlen(cstring) }
- #define MORPHO_STATICSTRINGWITHLENGTH(cstring, len) { .obj.type=OBJECT_STRING, .obj.↩ status=OBJECT_ISUNMANAGED, .obj.next=NULL, .string=cstring, .length=len }
- #define **OBJECT_STRINGLABEL** "string"
- #define **OBJECT_SYMBOLLABEL** "symbol"
- #define MORPHO_GETFUNCTION(val) ((objectfunction *) MORPHO_GETOBJECT(val))
- #define MORPHO_ISFUNCTION(val) object_istype(val, OBJECT_FUNCTION)
- #define MORPHO_GETUPVALUE(val) ((objectupvalue *) MORPHO_GETOBJECT(val))
- #define MORPHO_ISUPVALUE(val) object_istype(val, OBJECT_UPVALUE)
- #define MORPHO_ISCLOSURE(val) object_istype(val, OBJECT_CLOSURE)
- #define MORPHO_GETCLOSURE(val) ((objectclosure *) MORPHO_GETOBJECT(val))
- #define MORPHO_GETCLOSUREFUNCTION(val) (((objectclosure *) MORPHO_GETOBJECT(val))->func)

- #define MORPHO_ISCLASS(val) object_istype(val, OBJECT_CLASS)
- #define MORPHO_GETCLASS(val) ((objectclass *) MORPHO_GETOBJECT(val))
- #define MORPHO_GETSUPERCLASS(val) (MORPHO_GETCLASS(val)->superclass)
- #define MORPHO_ISINSTANCE(val) object_istype(val, OBJECT_INSTANCE)
- #define MORPHO_GETINSTANCE(val) ((objectinstance *) MORPHO_GETOBJECT(val))
- #define MORPHO_ISINVOCATION(val) object_istype(val, OBJECT_INVOCATION)
- #define MORPHO_GETINVOCATION(val) ((objectinvocation *) MORPHO_GETOBJECT(val))
- #define MORPHO_ISRANGE(val) object_istype(val, OBJECT_RANGE)
- #define MORPHO_GETRANGE(val) ((objectrange *) MORPHO_GETOBJECT(val))
- #define MORPHO_ISDICTIONARY(val) object_istype(val, OBJECT_DICTIONARY)
- #define MORPHO_GETDICTIONARY(val) ((objectdictionary *) MORPHO_GETOBJECT(val))
- #define MORPHO_ISARRAY(val) object_istype(val, OBJECT_ARRAY)
- #define MORPHO_GETARRAY(val) ((objectarray *) MORPHO_GETOBJECT(val))
- #define MORPHO_ISMATRIX(val) object_istype(val, OBJECT_MATRIX)
- #define MORPHO_GETMATRIX(val) ((objectmatrix *) MORPHO_GETOBJECT(val))
- #define MORPHO_STATICMATRIX(darray, nr, nc) { .obj.type=OBJECT_MATRIX, .obj.status=OBJECT_I↩ SUNMANAGED, .obj.next=NULL, .elements=darray, .nrows=nr, .ncols=nc }

  Use to create static matrices on the C stack.
- #define MORPHO_STATICDOKKEY(i, j) { .obj.type=OBJECT_DOKKEY, .obj.status=OBJECT_ISUNMAN↩ AGED, .obj.next=NULL, .row=i, .col=j }
- #define MORPHO_ISDOKKEY(val) object_istype(val, OBJECT_DOKKEY)
- #define MORPHO_GETDOKKEY(val) ((objectdokkey *) MORPHO_GETOBJECT(val))
- #define MORPHO_GETDOKKEYROW(objptr) ((unsigned int) (objptr)->row)
- #define **MORPHO_GETDOKKEYCOL**(objptr) ((unsigned int) (objptr)->col)
- #define **MORPHO_GETDOKROWWVAL**(val) ((unsigned int) (MORPHO_GETDOKKEY(val)->row))
- #define **MORPHO_GETDOKCOLWVAL**(val) ((unsigned int) (MORPHO_GETDOKKEY(val)->col))
- #define MORPHO_ISSPARSE(val) object_istype(val, OBJECT_SPARSE)
- #define MORPHO_GETSPARSE(val) ((objectsparse *) MORPHO_GETOBJECT(val))
- #define MORPHO_ISMESH(val) object_istype(val, OBJECT_MESH)
- #define MORPHO_GETMESH(val) ((objectmesh *) MORPHO_GETOBJECT(val))

Typedefs

- typedef ptrdiff_t **indx**
- typedef struct sobjectfunction objectfunction
- typedef struct sobjectupvalue **objectupvalue**
- typedef struct sobjectclass **objectclass**

Enumerations

- enum objecttype {
  OBJECT_STRING, **OBJECT_FUNCTION**, **OBJECT_BUILTINFUNCTION**, **OBJECT_CLOSURE**, **OBJECT_UPVALUE**, **OBJECT_CLASS**, **OBJECT_INSTANCE**, **OBJECT_INVOCATION**, **OBJECT_RANGE**, **OBJECT_DICTIONARY**, **OBJECT_ARRAY**, **OBJECT_MATRIX**, **OBJECT_SPARSE**, **OBJECT_DOKKEY**, **OBJECT_MESH**, **OBJECT_EXTERN** }
- enum **objectarrayerror** { **ARRAY_OK**, **ARRAY_WRONGDIM**, **ARRAY_NONNUMERICALINDX**, **ARRA↩ Y_OUTOFBOUNDS** }

Functions

- void object_init (object *obj, objecttype type)

  Initializes an object to be a certain type.
- void object_free (object *obj)
- void object_print (value v)
- void object_printtobuffer (value v, varray_char *buffer)
- object * object_new (size_t size, objecttype type)

  Allocates an object.
- size_t object_size (object *obj)
- value object_stringfromcstring (const char *in, size_t length)

  Creates a string from an existing character array with given length.
- value object_stringfromvarraychar (varray_char *in)

  Converts a varray_char into a string.
- value **object_clonestring** (value val)
- value object_concatenatestring (value a, value b)

  Concatenates strings together.
- void object_functioninit (objectfunction *func)

  Initializes a new function.
- void object_functionclear (objectfunction *func)

  Clears a function.
- bool object_functionaddprototype (objectfunction *func, varray_upvalue *v, indx *ix)
- objectfunction * object_getfunctionparent (objectfunction *func)
- value object_getfunctionname (objectfunction *func)
- varray_value * object_functiongetconstanttable (objectfunction *func)
- objectfunction * object_newfunction (indx entry, value name, objectfunction *parent, unsigned int nargs)

  Creates a new function.
- void object_upvalueinit (objectupvalue *c)
- objectupvalue * object_newupvalue (value *reg)
- objectclosure * object_newclosure (objectfunction *sf, objectfunction *func, indx np)

  Creates a new closure.
- objectclass * **object_newclass** (value name)
- objectinstance * **object_newinstance** (objectclass *klass)
- bool **objectinstance_setproperty** (objectinstance *obj, value key, value val)
- bool **objectinstance_getproperty** (objectinstance *obj, value key, value *val)
- objectinvocation * object_newinvocation (value receiver, value method)
- bool **objectinstance_insertpropertybycstring** (objectinstance *obj, char *property, value val)
- bool **objectinstance_getpropertybycstring** (objectinstance *obj, char *property, value *val)
- objectrange * object_newrange (value start, value end, value step)
- objectdictionary * object_newdictionary (void)
- objectarray * object_newarray (unsigned int dimension, unsigned int *dim)

  Creates an array object.
- objectarray * object_arrayfromlist (unsigned int n, value *v)
- objectarray * object_arrayfromvarrayvalue (varray_value *v)
- objectarray * object_arrayfromvalueindices (unsigned int ndim, value *dim)
- bool object_arrayvaluestoindices (unsigned int ndim, value *indx, unsigned int *iout)
- bool object_arrayindicestoelement (objectarray *array, unsigned int ndim, unsigned int *indx, unsigned int *ixout)

  Calculates the correct element from a set of array indices.
- objectarrayerror object_getarrayelement (objectarray *array, unsigned int ndim, value *indx, value *out)
- objectarrayerror object_setarrayelement (objectarray *array, unsigned int ndim, value *indx, value set)
- objectmatrix * object_newmatrix (unsigned int nrows, unsigned int ncols, bool zero)
- objectmatrix * object_matrixfromarray (objectarray *array)
- **DECLARE_VARRAY** (dokkey, objectdokkey)
- objectsparse * object_newsparse (int *nrows, int *ncols)
- objectsparse * **sparse_sparsefromarray** (objectarray *array)
- objectmesh * object_newmesh (unsigned int dim, unsigned int nv, double *v)

10.12.1 Detailed Description

Provide functionality for extended and mutable data types.

Author

T J Atherton

10.12.2 Macro Definition Documentation

10.12.2.1 MORPHO_GETARRAY

```
#define MORPHO_GETARRAY(
              val ) ((objectarray *) MORPHO_GETOBJECT(val))
```

Gets the object as an array

10.12.2.2 MORPHO_GETCLASS

```
#define MORPHO_GETCLASS(
              val ) ((objectclass *) MORPHO_GETOBJECT(val))
```

Gets the object as a class

10.12.2.3 MORPHO_GETCLOSURE

```
#define MORPHO_GETCLOSURE(
              val ) ((objectclosure *) MORPHO_GETOBJECT(val))
```

Gets the object as a closure

10.12.2.4 MORPHO_GETCLOSUREFUNCTION

```
#define MORPHO_GETCLOSUREFUNCTION(
              val ) (((objectclosure *) MORPHO_GETOBJECT(val))->func)
```

Retrieve the function object from a closure

10.12.2.5 MORPHO_GETDICTIONARY

```
#define MORPHO_GETDICTIONARY(
              val ) ((objectdictionary *) MORPHO_GETOBJECT(val))
```

Gets the object as a dictionary

10.12.2.6 MORPHO_GETDOKKEY

```
#define MORPHO_GETDOKKEY(
              val ) ((objectdokkey *) MORPHO_GETOBJECT(val))
```

Gets the object as a dok key

10.12.2.7 MORPHO_GETDOKKEYROW

```
#define MORPHO_GETDOKKEYROW(
              objptr ) ((unsigned int) (objptr)->row)
```

Gets the row and column from a objectdokkey

10.12.2.8 MORPHO_GETFUNCTION

```
#define MORPHO_GETFUNCTION(
              val ) ((objectfunction *) MORPHO_GETOBJECT(val))
```

Gets an objectfunction from a value

10.12.2.9 MORPHO_GETINSTANCE

```
#define MORPHO_GETINSTANCE(
              val ) ((objectinstance *) MORPHO_GETOBJECT(val))
```

Gets the object as a class

10.12.2.10 MORPHO_GETINVOCATION

```
#define MORPHO_GETINVOCATION(
              val ) ((objectinvocation *) MORPHO_GETOBJECT(val))
```

Gets the object as an invocation

10.12.2.11 MORPHO_GETMATRIX

```
#define MORPHO_GETMATRIX(
              val ) ((objectmatrix *) MORPHO_GETOBJECT(val))
```

Gets the object as an matrix

10.12.2.12 MORPHO_GETMESH

```
#define MORPHO_GETMESH(
              val ) ((objectmesh *) MORPHO_GETOBJECT(val))
```

Gets the object as a mesh

10.12.2.13 MORPHO_GETOBJECTHASH

```
#define MORPHO_GETOBJECTHASH(
              val ) (MORPHO_GETOBJECT(val)->hsh)
```

Gets an objects key

10.12.2.14 MORPHO_GETOBJECTTYPE

```
#define MORPHO_GETOBJECTTYPE(
              val ) (MORPHO_GETOBJECT(val)->type)
```

Gets the type of the object associated with a value

10.12.2.15 MORPHO_GETRANGE

```
#define MORPHO_GETRANGE(
              val ) ((objectrange *) MORPHO_GETOBJECT(val))
```

Gets the object as a range

10.12.2.16 MORPHO_GETSPARSE

```
#define MORPHO_GETSPARSE(
              val ) ((objectsparse *) MORPHO_GETOBJECT(val))
```

Gets the object as a sparse matrix

10.12.2.17 MORPHO_GETSUPERCLASS

```
#define MORPHO_GETSUPERCLASS(
              val ) (MORPHO_GETCLASS(val)->superclass)
```

Gets the superclass

10.12.2.18 MORPHO_GETUPVALUE

```
#define MORPHO_GETUPVALUE(
              val ) ((objectupvalue *) MORPHO_GETOBJECT(val))
```

Gets an objectfunction from a value

10.12.2.19 MORPHO_ISARRAY

```
#define MORPHO_ISARRAY(
              val ) object_istype(val, OBJECT_ARRAY)
```

Tests whether an object is an array

10.12.2.20 MORPHO_ISCLASS

```
#define MORPHO_ISCLASS(
              val ) object_istype(val, OBJECT_CLASS)
```

Tests whether an object is a class

10.12.2.21 MORPHO_ISCLOSURE

```
#define MORPHO_ISCLOSURE(
              val ) object_istype(val, OBJECT_CLOSURE)
```

Tests whether an object is a closure

10.12.2.22 MORPHO_ISDICTIONARY

```
#define MORPHO_ISDICTIONARY(
              val ) object_istype(val, OBJECT_DICTIONARY)
```

Tests whether an object is a dictionary

10.12.2.23 MORPHO_ISDOKKEY

```
#define MORPHO_ISDOKKEY(
              val ) object_istype(val, OBJECT_DOKKEY)
```

Tests whether an object is a dok key

10.12.2.24 MORPHO_ISFUNCTION

```
#define MORPHO_ISFUNCTION(
              val ) object_istype(val, OBJECT_FUNCTION)
```

Tests whether an object is a function

10.12.2.25 MORPHO_ISINSTANCE

```
#define MORPHO_ISINSTANCE(
              val ) object_istype(val, OBJECT_INSTANCE)
```

Tests whether an object is a class

10.12.2.26 MORPHO_ISINVOCATION

```
#define MORPHO_ISINVOCATION(
              val ) object_istype(val, OBJECT_INVOCATION)
```

Tests whether an object is an invocation

10.12.2.27 MORPHO_ISMATRIX

```
#define MORPHO_ISMATRIX(
              val ) object_istype(val, OBJECT_MATRIX)
```

Tests whether an object is a matrix

10.12.2.28 MORPHO_ISMESH

```
#define MORPHO_ISMESH(
              val ) object_istype(val, OBJECT_MESH)
```

Tests whether an object is a mesh

10.12.2.29 MORPHO_ISRANGE

```
#define MORPHO_ISRANGE(
              val ) object_istype(val, OBJECT_RANGE)
```

Tests whether an object is a range

10.12.2.30 MORPHO_ISSPARSE

```
#define MORPHO_ISSPARSE(
              val ) object_istype(val, OBJECT_SPARSE)
```

Tests whether an object is a sparse matrix

10.12.2.31 MORPHO_ISSTRING

```
#define MORPHO_ISSTRING(
              val ) object_istype(val, OBJECT_STRING)
```

Tests whether an object is a string

10.12.2.32 MORPHO_ISUPVALUE

```
#define MORPHO_ISUPVALUE(
              val ) object_istype(val, OBJECT_UPVALUE)
```

Tests whether an object is a function

10.12.2.33 MORPHO_SETOBJECTHASH

```
#define MORPHO_SETOBJECTHASH(
              val,
              newhash ) (MORPHO_GETOBJECT(val)->hsh = newhash)
```

Sets an objects key

10.12.2.34 MORPHO_STATICDOKKEY

```
#define MORPHO_STATICDOKKEY(
              i,
              j ) { .obj.type=OBJECT_DOKKEY, .obj.status=OBJECT_ISUNMANAGED, .obj.next=NULL,
.row=i, .col=j }
```

Create

10.12.2.35 MORPHO_STATICMATRIX

```
#define MORPHO_STATICMATRIX(
              darray,
              nr,
              nc ) { .obj.type=OBJECT_MATRIX, .obj.status=OBJECT_ISUNMANAGED, .obj.next=NULL,
.elements=darray, .nrows=nr, .ncols=nc }
```

Use to create static matrices on the C stack.

Intended for small matrices; Caller needs to supply a double array of size nr∗nc.

10.12.2.36 MORPHO_STATICSTRING

```
#define MORPHO_STATICSTRING(
              cstring ) { .obj.type=OBJECT_STRING, .obj.status=OBJECT_ISUNMANAGED, .obj.↩
next=NULL, .string=cstring, .length=strlen(cstring) }
```

Use to create static strings on the C stack

10.12.2.37 MORPHO_STATICSTRINGWITHLENGTH

```
#define MORPHO_STATICSTRINGWITHLENGTH(
              cstring,
              len ) { .obj.type=OBJECT_STRING, .obj.status=OBJECT_ISUNMANAGED, .obj.next=NULL,
.string=cstring, .length=len }
```

Use to create static strings on the C stack

10.12.3 Typedef Documentation

10.12.3.1 objectfunction

```
typedef struct sobjectfunction objectfunction
```

A function object

10.12.4 Enumeration Type Documentation

10.12.4.1 objecttype

```
enum objecttype
```

The type of an object

10.12.5 Function Documentation

10.12.5.1 object_arrayfromlist()

```
objectarray* object_arrayfromlist (
             unsigned int n,
             value * v )
```

Creates a new array from a list of values

Creates a new 1D array from a list of values

10.12.5.2 object_arrayfromvalueindices()

```
objectarray* object_arrayfromvalueindices (
             unsigned int ndim,
             value * dim )
```

Creates a new array object with the dimensions given as a list of values

Creates a new array object with the indices given as a list of values

10.12.5.3 object_arrayfromvarrayvalue()

```
objectarray* object_arrayfromvarrayvalue (
             varray_value * v )
```

Creates a new 1D array from a list of varray_value

10.12.5.4 object_arrayindicestoelement()

```
bool object_arrayindicestoelement (
             objectarray * array,
             unsigned int ndim,
             unsigned int * indx,
             unsigned int * ixout )
```

Calculates the correct element from a set of array indices.

Parameters

in	*array*	- the array
in	*ndim*	- number of dimensions
in	*indx*	- list of indices
out	*ixout*	- the element number to use

Returns

true on success, false if indices are out of bounds

10.12.5.5 object_arrayvaluestoindices()

```
bool object_arrayvaluestoindices (
            unsigned int ndim,
            value * indx,
            unsigned int * iout )
```

Converts a list of indices into a list of unsigned ints

Parameters

ndim	- number of dimensions
indx	- the indices to evaluate
iout	- the indices as integers

Returns

true on success, or false if an unexpected type was encountered

10.12.5.6 object_concatenatestring()

```
value object_concatenatestring (
            value a,
            value b )
```

Concatenates strings together.

Parameters

a	first string
b	second string

Returns

the object (as a value) which will be MORPHO_NIL on failure

10.12.5.7 object_free()

```
void object_free (
            object * obj )
```

Frees an object

10.12.5.8 object_functionaddprototype()

```
bool object_functionaddprototype (
            objectfunction * func,
            varray_upvalue * v,
            indx * ix )
```

Adds an upvalue prototype to a function

Parameters

in	*func*	function object to add to
in	*v*	a varray of upvalues that will be copied into the function definition.
out	*ix*	index of the closure created

Returns

true on success

10.12.5.9 object_functionclear()

```
void object_functionclear (
            objectfunction * func )
```

Clears a function.

Clear the upvalue prototypes

10.12.5.10 object_functiongetconstanttable()

```
varray_value* object_functiongetconstanttable (
            objectfunction * func )
```

Gets the constant table associated with a function

10.12.5.11 object_getarrayelement()

```
objectarrayerror object_getarrayelement (
            objectarray * array,
            unsigned int ndim,
            value * indx,
            value * out )
```

Gets an array element

10.12.5.12 object_getfunctionname()

```
value object_getfunctionname (
            objectfunction * func )
```

Gets the name of a function

10.12.5.13 object_getfunctionparent()

```
objectfunction* object_getfunctionparent (
            objectfunction * func )
```

Gets the parent of a function

10.12.5.14 object_init()

```
void object_init (
            object * obj,
            objecttype type )
```

Initializes an object to be a certain type.

Parameters

obj	object to initialize
type	type to initialize with

10.12.5.15 object_matrixfromarray()

```
objectmatrix* object_matrixfromarray (
            objectarray * array )
```

Creates a new array from a list of values

10.12.5.16 object_new()

```
object* object_new (
            size_t size,
            objecttype type )
```

Allocates an object.

Parameters

size	size of memory to reserve
type	type to initialize with

10.12.5.17 object_newarray()

```
objectarray* object_newarray (
            unsigned int ndim,
            unsigned int * dim )
```

Creates an array object.

Creates an array object

Arrays are stored in memory as follows: objectarray structure with flexible array member value value [0..dim-1] the dimensions of the array value [dim..] array elements in column major order, i.e. the matrix [[1, 2], [3, 4]] is stored as: <structure> // the structure 2, 2, // the dimensions 1, 3, 2, 4 // the elements in column major order

10.12.5.18 object_newclosure()

```
objectclosure* object_newclosure (
            objectfunction * sf,
            objectfunction * func,
            indx np )
```

Creates a new closure.

Parameters

sf	the objectfunction of the current environment
func	a function object to enclose
np	the prototype number to use

10.12.5.19 object_newdictionary()

```
objectdictionary* object_newdictionary (
            void  )
```

Creates a new dictionary

10.12.5.20 object_newinvocation()

```
objectinvocation* object_newinvocation (
            value receiver,
            value method )
```

Create a new invocation

10.12.5.21 object_newmatrix()

```
objectmatrix* object_newmatrix (
            unsigned int nrows,
            unsigned int ncols,
            bool zero )
```

Creates a matrix object

10.12.5.22 object_newmesh()

```
objectmesh* object_newmesh (
            unsigned int dim,
            unsigned int nv,
            double * v )
```

Creates a mesh object

10.12.5.23 object_newrange()

```
objectrange* object_newrange (
            value start,
            value end,
            value step )
```

Create a new range. Step may be set to MORPHO_NIL to use the default value of 1

10.12.5.24 object_newsparse()

```
objectsparse* object_newsparse (
            int * nrows,
            int * ncols )
```

Creates a sparse matrix object

Parameters

in	*nrows*	} Optional number of rows and columns
in	*ncols*	}

10.12.5.25 object_newupvalue()

```
objectupvalue* object_newupvalue (
            value * reg )
```

Creates a new upvalue for the register pointed to by reg.

10.12.5.26 object_print()

```
void object_print (
            value v )
```

Prints an object

10.12.5.27 object_printtobuffer()

```
void object_printtobuffer (
            value v,
            varray_char * buffer )
```

Prints an object to a string buffer

Parameters

in	*v*	Object to convert to a buffer
in	*buffer*	Buffer to output to

10.12.5.28 object_setarrayelement()

```
objectarrayerror object_setarrayelement (
            objectarray * array,
            unsigned int ndim,
            value * indx,
            value set )
```

Sets an array element

10.12.5.29 object_size()

```
size_t object_size (
            object * obj )
```

Gets the total size of an object

10.12.5.30 object_stringfromcstring()

```
value object_stringfromcstring (
            const char * in,
            size_t length )
```

Creates a string from an existing character array with given length.

Parameters

in	the string to copy
length	length of string to copy

Returns

the object (as a value) which will be MORPHO_NIL on failure

10.12.5.31 object_stringfromvarraychar()

```
value object_stringfromvarraychar (
            varray_char * in )
```

Converts a varray_char into a string.

Parameters

in	the varray to convert

Returns

the object (as a value) which will be MORPHO_NIL on failure

10.12.5.32 object_upvalueinit()

```
void object_upvalueinit (
            objectupvalue * c )
```

Initializes a new upvalue object.

10.13 datastructures/sparse.c File Reference

Veneer class over the objectsparse type that provides sparse matrices.

```
#include "build.h"
#include "sparse.h"
#include "morpho.h"
#include "dictionary.h"
#include "common.h"
#include "matrix.h"
#include "builtin.h"
#include <limits.h>
#include <stdlib.h>
```

Functions

- DEFINE_VARRAY (dokkey, objectdokkey)
- void sparsedok_init (sparsedok *dok)
- void sparsedok_clear (sparsedok *dok)
- bool sparsedok_insert (sparsedok *dok, int i, int j, value val)
- bool sparsedok_get (sparsedok *dok, int i, int j, value *val)
- bool sparsedok_remove (sparsedok *dok, int i, int j, value *val)
- bool sparsedok_setdimensions (sparsedok *dok, int nrows, int ncols)
- void sparsedok_print (sparsedok *dok)
- void sparseccs_init (sparseccs *ccs)
- void sparseccs_clear (sparseccs *ccs)
- bool sparseccs_resize (sparseccs *ccs, int nrows, int ncols, unsigned int nentries, bool values)
- bool sparseccs_getrowindices (sparseccs *ccs, int col, int *nentries, int **entries)
- bool sparseccs_set (sparseccs *ccs, int i, int j, double val)
- bool sparseccs_get (sparseccs *ccs, int i, int j, double *val)
- bool sparseccs_doktoccs (sparsedok *in, sparseccs *out, bool copyvals)
- void sparseccs_print (sparseccs *ccs)
- bool sparse_checkformat (objectsparse *sparse, objectsparseformat format, bool force)
- void sparse_removeformat (objectsparse *s, objectsparseformat format)
- void **sparse_test** (void)
- objectsparse * object_newsparse (int *nrows, int *ncols)
- objectsparse * object_sparsefromarray (objectarray *array)
- bool sparse_setelement (objectsparse *s, int row, int col, value val)
- bool sparse_getelement (objectsparse *s, int row, int col, value *val)
- objectsparseerror sparse_add (objectsparse *a, objectsparse *b, double alpha, double beta, objectsparse *out)
- objectsparseerror sparse_mul (objectsparse *a, objectsparse *b, objectsparse *out)
- objectsparseerror sparse_div (objectsparse *a, objectmatrix *b, objectmatrix *out)
- void sparse_clear (objectsparse *a)
- size_t sparse_size (objectsparse *a)
- void **sparse_raiseerror** (vm *v, objectsparseerror err)
- value sparse_constructor (vm *v, int nargs, value *args)
- value Sparse_getindex (vm *v, int nargs, value *args)
- value Sparse_setindex (vm *v, int nargs, value *args)
- value **Sparse_print** (vm *v, int nargs, value *args)
- value **Sparse_add** (vm *v, int nargs, value *args)
- value **Sparse_sub** (vm *v, int nargs, value *args)
- value **Sparse_mul** (vm *v, int nargs, value *args)

- value **Sparse_div** (vm *v, int nargs, value *args)
- value **Sparse_divr** (vm *v, int nargs, value *args)
- **MORPHO_METHOD** (MORPHO_GETINDEX_METHOD, Sparse_getindex, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_SETINDEX_METHOD, Sparse_setindex, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_PRINT_METHOD, Sparse_print, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_ADD_METHOD, Sparse_add, BUILTIN_FLAGSEMPTY)
- **MORPHO_METHOD** (MORPHO_SUB_METHOD, Sparse_sub, BUILTIN_FLAGSEMPTY)
- MORPHO_ENDCLASS void **sparse_initialize** (void)

10.13.1 Detailed Description

Veneer class over the objectsparse type that provides sparse matrices.

Author

T J Atherton

10.13.2 Function Documentation

10.13.2.1 DEFINE_VARRAY()

```
DEFINE_VARRAY (
            dokkey ,
            objectdokkey  )
```

—- CSparse —

10.13.2.2 object_newsparse()

```
objectsparse* object_newsparse (
            int * nrows,
            int * ncols )
```

Creates a sparse matrix object

Parameters

in	*nrows*	} Optional number of rows and columns
in	*ncols*	}

10.13.2.3 object_sparsefromarray()

```
objectsparse* object_sparsefromarray (
            objectarray * array )
```

Create a sparse array from a list of lists

10.13.2.4 sparse_add()

```
objectsparseerror sparse_add (
            objectsparse * a,
            objectsparse * b,
            double alpha,
            double beta,
            objectsparse * out )
```

Add two matrices

Parameters

in	*a*	- sparse matrix
in	*b*	- sparse matrix
in	*alpha*	- scale for a
in	*beta*	- scale for b
out	*out*	- alpha∗a+beta∗b.

10.13.2.5 sparse_checkformat()

```
bool sparse_checkformat (
            objectsparse * sparse,
            objectsparseformat format,
            bool force )
```

Checks whether a format is available.

Parameters

sparse	the matrix to check
format	format to check
force	if format is unavailable, try to make it available

Returns

true if the format is available

10.13.2.6 sparse_clear()

```
void sparse_clear (
            objectsparse * a )
```

Clears any data attached to a sparse matrix

10.13.2.7 sparse_constructor()

```
value sparse_constructor (
            vm * v,
            int nargs,
            value * args )
```

Constructs a Matrix object

10.13.2.8 sparse_div()

```
objectsparseerror sparse_div (
            objectsparse * a,
            objectmatrix * b,
            objectmatrix * out )
```

Multiply two matrices

Parameters

in	*a*	- sparse matrix
in	*b*	- dense rhs (may have more than one column)
out	*out*	- Solution to a.x = b.

10.13.2.9 sparse_getelement()

```
bool sparse_getelement (
            objectsparse * s,
            int row,
            int col,
            value * val )
```

Get an element

Parameters

in	*s*	the sparse object
in	*row*	the row
in	*col*	the column
out	*val*	the value; pass NULL to check if an element exists

10.13.2.10 Sparse_getindex()

```
value Sparse_getindex (
            vm * v,
```

```
            int nargs,
            value * args )
```

Retrieve a matrix element

10.13.2.11 sparse_mul()

```
objectsparseerror sparse_mul (
            objectsparse * a,
            objectsparse * b,
            objectsparse * out )
```

Multiply two matrices

Parameters

in	*a*	- sparse matrix
in	*b*	- sparse matrix
out	*out*	- a∗b.

10.13.2.12 sparse_removeformat()

```
void sparse_removeformat (
            objectsparse * s,
            objectsparseformat format )
```

Removes data structures for a given format

10.13.2.13 sparse_setelement()

```
bool sparse_setelement (
            objectsparse * s,
            int row,
            int col,
            value val )
```

Set an element

10.13.2.14 Sparse_setindex()

```
value Sparse_setindex (
            vm * v,
            int nargs,
            value * args )
```

Set a matrix element

10.13.2.15 sparse_size()

```
size_t sparse_size (
            objectsparse * a )
```

Calculate the size of a sparse matrix structure

10.13.2.16 sparseccs_clear()

```
void sparseccs_clear (
            sparseccs * ccs )
```

Clears all data structures associated with a sparseccs

10.13.2.17 sparseccs_doktoccs()

```
bool sparseccs_doktoccs (
            sparsedok * in,
            sparseccs * out,
            bool copyvals )
```

Converts a DOK matrix to a CCS matrix

10.13.2.18 sparseccs_get()

```
bool sparseccs_get (
            sparseccs * ccs,
            int i,
            int j,
            double * val )
```

Retrieves a matrix element (i,j) from a sparseccs structure

Returns

true on success.

10.13.2.19 sparseccs_getrowindices()

```
bool sparseccs_getrowindices (
            sparseccs * ccs,
            int col,
            int * nentries,
            int ** entries )
```

Retrieves the row indices given a column

Parameters

in	*ccs*	the matrix
in	*col*	column index
out	*nentries*	the number of entries
out	*entries*	the entries themselves

10.13.2.20 sparseccs_init()

```
void sparseccs_init (
            sparseccs * ccs )
```

Initializes an empty sparseccs

10.13.2.21 sparseccs_print()

```
void sparseccs_print (
            sparseccs * ccs )
```

Prints a sparsedok matrix

10.13.2.22 sparseccs_resize()

```
bool sparseccs_resize (
            sparseccs * ccs,
            int nrows,
            int ncols,
            unsigned int nentries,
            bool values )
```

Resizes a sparseccs

10.13.2.23 sparseccs_set()

```
bool sparseccs_set (
            sparseccs * ccs,
            int i,
            int j,
            double val )
```

Sets a matrix element (i,j) to be a specified value

Returns

true if the element exists in the given sparsity structure, false otherwise.

10.13.2.24 sparsedok_clear()

```
void sparsedok_clear (
            sparsedok * dok )
```

Clears a sparsedok structure

10.13.2.25 sparsedok_get()

```
bool sparsedok_get (
            sparsedok * dok,
            int i,
            int j,
            value * val )
```

Retrieves a matrix element (i,j) from a sparsedok structure

Returns

true on success.

10.13.2.26 sparsedok_init()

```
void sparsedok_init (
            sparsedok * dok )
```

Initializes a sparsedok structure

10.13.2.27 sparsedok_insert()

```
bool sparsedok_insert (
            sparsedok * dok,
            int i,
            int j,
            value val )
```

Inserts a matrix element (i,j) -> val into a sparsedok structure

Returns

true on success.

10.13.2.28 sparsedok_print()

```
void sparsedok_print (
            sparsedok * dok )
```

Prints a sparsedok matrix

10.13.2.29 sparsedok_remove()

```
bool sparsedok_remove (
            sparsedok * dok,
            int i,
            int j,
            value * val )
```

Removes a matrix element (i,j) from a sparsedok

Returns

true on success.

Warning

Use sparingly as the deleted key is not recovered.

10.13.2.30 sparsedok_setdimensions()

```
bool sparsedok_setdimensions (
            sparsedok * dok,
            int nrows,
            int ncols )
```

Sets the dimensions of the matrix

Returns

true if successful, or false if the dimensions are incompatible with existing matrix entries This function is intended for use in constructing matrix.

10.14 datastructures/sparse.h File Reference

Veneer class over the objectsparse type that provides sparse matrices.

```
#include <stdio.h>
#include "object.h"
#include "morpho.h"
```

Macros

- #define **SPARSE_CLASSNAME** "Sparse"
- #define **SPARSE_CONSTRUCTOR** "SprsCns"
- #define **SPARSE_CONSTRUCTOR_MSG** "Sparse() should be called either with dimensions or an array initializer."
- #define **SPARSE_SETFAILED** "SprsSt"
- #define **SPARSE_SETFAILED_MSG** "Attempt to set sparse matrix element failed."
- #define **SPARSE_INVLDARRAYINIT** "SprsInvldInit"
- #define **SPARSE_INVLDARRAYINIT_MSG** "Array initializer passed to Sparse() must be a 1 or 2 dimensional array."
- #define **SPARSE_CONVFAILEDERR** "SprsCnvFld"
- #define **SPARSE_CONVFAILEDERR_MSG** "Sparse format conversion failed."
- #define **SPARSE_OPFAILEDERR** "SprsOpFld"
- #define **SPARSE_OPFAILEDERR_MSG** "Sparse matrix operation failed."

Enumerations

- enum **objectsparseformat** { **SPARSE_DOK**, **SPARSE_CCS** }
- enum **objectsparseerror** { **SPARSE_OK**, **SPARSE_INCMPTBLDIM**, **SPARSE_CONVFAILED**, **SPARSE_FAILED** }

Functions

- void sparsedok_init (sparsedok *dok)
- void sparsedok_clear (sparsedok *dok)
- bool sparsedok_insert (sparsedok *dok, int i, int j, value val)
- bool sparsedok_get (sparsedok *dok, int i, int j, value *val)
- bool sparsedok_remove (sparsedok *dok, int i, int j, value *val)
- bool sparsedok_setdimensions (sparsedok *dok, int nrows, int ncols)
- void sparseccs_init (sparseccs *ccs)
- void sparseccs_clear (sparseccs *ccs)
- bool sparseccs_resize (sparseccs *ccs, int nrows, int ncols, unsigned int nentries, bool values)
- bool sparseccs_get (sparseccs *ccs, int i, int j, double *val)
- bool sparseccs_getrowindices (sparseccs *ccs, int col, int *nentries, int **entries)
- bool sparseccs_doktoccs (sparsedok *in, sparseccs *out, bool copyvals)
- bool sparse_setelement (objectsparse *matrix, int row, int col, value value)
- bool sparse_getelement (objectsparse *matrix, int row, int col, value *value)
- objectsparseerror sparse_add (objectsparse *a, objectsparse *b, double alpha, double beta, objectsparse *out)
- objectsparseerror sparse_mul (objectsparse *a, objectsparse *b, objectsparse *out)
- void sparse_clear (objectsparse *a)
- size_t sparse_size (objectsparse *a)
- value **Sparse_divr** (vm *v, int nargs, value *args)
- void **sparse_initialize** (void)

10.14.1 Detailed Description

Veneer class over the objectsparse type that provides sparse matrices.

Author

T J Atherton

10.14.2 Function Documentation

10.14.2.1 sparse_add()

```
objectsparseerror sparse_add (
            objectsparse * a,
            objectsparse * b,
            double alpha,
            double beta,
            objectsparse * out )
```

Add two matrices

Parameters

in	*a*	- sparse matrix
in	*b*	- sparse matrix
in	*alpha*	- scale for a
in	*beta*	- scale for b
out	*out*	- alpha∗a+beta∗b.

10.14.2.2 sparse_clear()

```
void sparse_clear (
            objectsparse * a )
```

Clears any data attached to a sparse matrix

10.14.2.3 sparse_getelement()

```
bool sparse_getelement (
            objectsparse * s,
            int row,
            int col,
            value * val )
```

Get an element

Parameters

in	*s*	the sparse object
in	*row*	the row
in	*col*	the column
out	*val*	the value; pass NULL to check if an element exists

10.14.2.4 sparse_mul()

```
objectsparseerror sparse_mul (
            objectsparse * a,
            objectsparse * b,
            objectsparse * out )
```

Multiply two matrices

Parameters

in	*a*	- sparse matrix
in	*b*	- sparse matrix
out	*out*	- a∗b.

10.14.2.5 sparse_setelement()

```
bool sparse_setelement (
            objectsparse * s,
            int row,
            int col,
            value val )
```

Set an element

10.14.2.6 sparse_size()

```
size_t sparse_size (
            objectsparse * a )
```

Calculate the size of a sparse matrix structure

10.14.2.7 sparseccs_clear()

```
void sparseccs_clear (
            sparseccs * ccs )
```

Clears all data structures associated with a sparseccs

10.14.2.8 sparseccs_doktoccs()

```
bool sparseccs_doktoccs (
            sparsedok * in,
            sparseccs * out,
            bool copyvals )
```

Converts a DOK matrix to a CCS matrix

10.14.2.9 sparseccs_get()

```
bool sparseccs_get (
            sparseccs * ccs,
            int i,
            int j,
            double * val )
```

Retrieves a matrix element (i,j) from a sparseccs structure

Returns

true on success.

10.14.2.10 sparseccs_getrowindices()

```
bool sparseccs_getrowindices (
            sparseccs * ccs,
            int col,
            int * nentries,
            int ** entries )
```

Retrieves the row indices given a column

Parameters

in	*ccs*	the matrix
in	*col*	column index
out	*nentries*	the number of entries
out	*entries*	the entries themselves

10.14.2.11 sparseccs_init()

```
void sparseccs_init (
            sparseccs * ccs )
```

Initializes an empty sparseccs

10.14.2.12 sparseccs_resize()

```
bool sparseccs_resize (
            sparseccs * ccs,
            int nrows,
            int ncols,
            unsigned int nentries,
            bool values )
```

Resizes a sparseccs

10.14.2.13 sparsedok_clear()

```
void sparsedok_clear (
            sparsedok * dok )
```

Clears a sparsedok structure

10.14.2.14 sparsedok_get()

```
bool sparsedok_get (
            sparsedok * dok,
            int i,
            int j,
            value * val )
```

Retrieves a matrix element (i,j) from a sparsedok structure

Returns

true on success.

10.14.2.15 sparsedok_init()

```
void sparsedok_init (
            sparsedok * dok )
```

Initializes a sparsedok structure

10.14.2.16 sparsedok_insert()

```
bool sparsedok_insert (
            sparsedok * dok,
            int i,
            int j,
            value val )
```

Inserts a matrix element (i,j) -> val into a sparsedok structure

Returns

true on success.

10.14.2.17 sparsedok_remove()

```
bool sparsedok_remove (
            sparsedok * dok,
            int i,
            int j,
            value * val )
```

Removes a matrix element (i,j) from a sparsedok

Returns

true on success.

Warning

Use sparingly as the deleted key is not recovered.

10.14.2.18 sparsedok_setdimensions()

```
bool sparsedok_setdimensions (
            sparsedok * dok,
            int nrows,
            int ncols )
```

Sets the dimensions of the matrix

Returns

true if successful, or false if the dimensions are incompatible with existing matrix entries This function is intended for use in constructing matrix.

10.15 datastructures/syntaxtree.c File Reference

Syntax tree data structure for morpho.

```
#include <stdio.h>
#include "syntaxtree.h"
#include "common.h"
```

Functions

- **DEFINE_VARRAY** (syntaxtreenode, syntaxtreenode)
- **DEFINE_VARRAY** (syntaxtreeindx, syntaxtreeindx)
- void syntaxtree_init (syntaxtree *tree)

  Initialize a syntax tree.
- void syntaxtree_clear (syntaxtree *tree)

  Finalize a syntax tree.
- syntaxtreeindx syntaxtree_addnode (syntaxtree *tree, syntaxtreenodetype type, value content, int line, int posn, syntaxtreeindx left, syntaxtreeindx right)

  Adds a node to the syntax tree.
- syntaxtreenode * syntaxtree_nodefromindx (syntaxtree *tree, syntaxtreeindx indx)
- void **syntaxtree_flatten** (syntaxtree *tree, syntaxtreeindx indx, unsigned int ntypes, syntaxtreenodetype *types, varray_syntaxtreeindx *list)

10.15.1 Detailed Description

Syntax tree data structure for morpho.

Author

T J Atherton

10.15.2 Function Documentation

10.15.2.1 syntaxtree_addnode()

```
syntaxtreeindx syntaxtree_addnode (
            syntaxtree * tree,
            syntaxtreenodetype type,
            value content,
            int line,
            int posn,
            syntaxtreeindx left,
            syntaxtreeindx right )
```

Adds a node to the syntax tree.

Parameters

tree	tree to add to.
type	type of node to add
content	a value to add
left	} left ...
right	} ...and right branches of the node.

10.15.2.2 syntaxtree_clear()

```
void syntaxtree_clear (
            syntaxtree * tree )
```

Finalize a syntax tree.

Free attached objects

10.15.2.3 syntaxtree_nodefromindx()

```
syntaxtreenode* syntaxtree_nodefromindx (
            syntaxtree * tree,
            syntaxtreeindx indx )
```

Gets a syntaxtree node from its index

10.16 datastructures/syntaxtree.h File Reference

Syntax tree data structure for morpho.

```
#include <stddef.h>
#include "value.h"
#include "varray.h"
```

Classes

- struct _syntaxtreenode

  A node on the syntax tree is defined by a value and indices of the left and right elements.
- struct syntaxtree

Macros

- #define **SYNTAXTREE_ISLEAF**(x) syntaxtree_istype(x, NODE_BASE, NODE_LEAF)
- #define **SYNTAXTREE_ISUNARY**(x) syntaxtree_istype(x, NODE_LEAF, NODE_UNARY)
- #define **SYNTAXTREE_ISOPERATOR**(x) syntaxtree_istype(x, NODE_UNARY, NODE_OPERATOR)
- #define **SYNTAXTREE_ISSTATEMENT**(x) syntaxtree_istype(x, NODE_OPERATOR, NODE_STATEME↩ NT)
- #define **SYNTAXTREE_UNCONNECTED** -1

Typedefs

- typedef ptrdiff_t **syntaxtreeindx**
- typedef struct _syntaxtreenode syntaxtreenode

  A node on the syntax tree is defined by a value and indices of the left and right elements.

Enumerations

- enum syntaxtreenodetype {
  NODE_BASE, **NODE_NIL**, **NODE_BOOL**, **NODE_FLOAT**,
  NODE_INTEGER, **NODE_STRING**, **NODE_SYMBOL**, **NODE_SELF**,
  NODE_SUPER, **NODE_LEAF**, **NODE_NEGATE**, **NODE_NOT**,
  NODE_UNARY, **NODE_ADD**, **NODE_SUBTRACT**, **NODE_MULTIPLY**,
  NODE_DIVIDE, **NODE_POW**, **NODE_ASSIGN**, **NODE_EQ**,
  NODE_NEQ, **NODE_LT**, **NODE_GT**, **NODE_LTEQ**,
  NODE_GTEQ, **NODE_AND**, **NODE_OR**, **NODE_DOT**,
  NODE_RANGE, **NODE_OPERATOR**, **NODE_PRINT**, **NODE_DECLARATION**,
  NODE_FUNCTION, **NODE_METHOD**, **NODE_CLASS**, **NODE_RETURN**,
  NODE_IF, **NODE_THEN**, **NODE_WHILE**, **NODE_FOR**,
  NODE_IN, **NODE_STATEMENT**, **NODE_GROUPING**, **NODE_SEQUENCE**,
  NODE_INTERPOLATION, **NODE_ARGLIST**, **NODE_SCOPE**, **NODE_CALL**,
  NODE_INDEX, **NODE_LIST**, **NODE_IMPORT** }

  Type of node.

Functions

- **DECLARE_VARRAY** (syntaxtreenode, syntaxtreenode)
- **DECLARE_VARRAY** (syntaxtreeindx, syntaxtreeindx)
- void syntaxtree_init (syntaxtree *tree)

  Initialize a syntax tree.
- void syntaxtree_clear (syntaxtree *tree)

  Finalize a syntax tree.
- void **syntaxtree_print** (syntaxtree *tree)
- syntaxtreeindx syntaxtree_addnode (syntaxtree *tree, syntaxtreenodetype type, value content, int line, int posn, syntaxtreeindx left, syntaxtreeindx right)

  Adds a node to the syntax tree.
- syntaxtreenode * syntaxtree_nodefromindx (syntaxtree *tree, syntaxtreeindx indx)
- void **syntaxtree_flatten** (syntaxtree *tree, syntaxtreeindx indx, unsigned int ntypes, syntaxtreenodetype *types, varray_syntaxtreeindx *list)

10.16.1 Detailed Description

Syntax tree data structure for morpho.

Author

T J Atherton

10.16.2 Function Documentation

10.16.2.1 syntaxtree_addnode()

```
syntaxtreeindx syntaxtree_addnode (
            syntaxtree * tree,
            syntaxtreenodetype type,
            value content,
            int line,
            int posn,
            syntaxtreeindx left,
            syntaxtreeindx right )
```

Adds a node to the syntax tree.

Parameters

tree	tree to add to.
type	type of node to add
content	a value to add
left	} left ...
right	} ...and right branches of the node.

10.16.2.2 syntaxtree_clear()

```
void syntaxtree_clear (
            syntaxtree * tree )
```

Finalize a syntax tree.

Free attached objects

10.16.2.3 syntaxtree_nodefromindx()

```
syntaxtreenode* syntaxtree_nodefromindx (
            syntaxtree * tree,
            syntaxtreeindx indx )
```

Gets a syntaxtree node from its index

10.17 datastructures/value.c File Reference

Fundamental data type for morpho.

```
#include "value.h"
#include "common.h"
```

Functions

- **DEFINE_VARRAY** (value, value)
- bool varray_valuefind (varray_value *varray, value v, unsigned int *out)

  Finds a value in an varray using a loose equality test (MORPHO_ISEQUAL)
- bool varray_valuefindsame (varray_value *varray, value v, unsigned int *out)

  Finds a value in an varray using strict equality test (MORPHO_ISSAME)
- bool value_promotenumberlist (unsigned int nv, value *v)

10.17.1 Detailed Description

Fundamental data type for morpho.

Author

T J Atherton

10.17.2 Function Documentation

10.17.2.1 value_promotenumberlist()

```
bool value_promotenumberlist (
            unsigned int nv,
            value * v )
```

Promotes a list of numbers to floats if any are floating point.

Parameters

in	*nv*	- number of values
in	*v*	- list of values

Returns

true if successful, false if any values are not numbers

10.17.2.2 varray_valuefind()

```
bool varray_valuefind (
            varray_value * varray,
            value v,
            unsigned int * out )
```

Finds a value in an varray using a loose equality test (MORPHO_ISEQUAL)

Parameters

in	*varray*	the array to search
in	*v*	value to find
out	*out*	index of the match

Returns

whether the value was found or not.

10.17.2.3 varray_valuefindsame()

```
bool varray_valuefindsame (
            varray_value * varray,
            value v,
            unsigned int * out )
```

Finds a value in an varray using strict equality test (MORPHO_ISSAME)

Parameters

in	*varray*	the array to search
in	*v*	value to find
out	*out*	index of the match

Returns

whether the value was found or not.

10.18 datastructures/value.h File Reference

Fundamental data type for morpho.

```
#include <stdint.h>
#include <stdbool.h>
#include "build.h"
#include "varray.h"
```

Classes

- struct value

  The unboxed value type.

Macros

- #define MORPHO_GETTYPE(v) ((v).type)
- #define MORPHO_ISNIL(v) ((v).type==VALUE_NIL)
- #define **MORPHO_ISINTEGER**(v) ((v).type==VALUE_INTEGER)
- #define **MORPHO_ISFLOAT**(v) ((v).type==VALUE_DOUBLE)
- #define **MORPHO_ISBOOL**(v) ((v).type==VALUE_BOOL)
- #define **MORPHO_ISOBJECT**(v) ((v).type==VALUE_OBJECT)
- #define MORPHO_NIL ((value) { VALUE_NIL, .as.integer = (int) 0 })
- #define **MORPHO_INTEGER**(x) ((value) { VALUE_INTEGER, .as.integer = (int) (x) })
- #define **MORPHO_FLOAT**(x) ((value) { VALUE_DOUBLE, .as.real = (double) x })
- #define **MORPHO_BOOL**(x) ((value) { VALUE_BOOL, .as.boolean = (bool) x })
- #define **MORPHO_OBJECT**(x) ((value) { VALUE_OBJECT, .as.obj = (object *) x })
- #define **MORPHO_TRUE** MORPHO_BOOL(true)
- #define **MORPHO_FALSE** MORPHO_BOOL(false)
- #define MORPHO_GETINTEGERVALUE(v) ((v).as.integer)
- #define **MORPHO_GETFLOATVALUE**(v) ((v).as.real)
- #define **MORPHO_GETBOOLVALUE**(v) ((v).as.boolean)
- #define **MORPHO_GETOBJECT**(v) ((v).as.obj)
- #define **MORPHO_ISNUMBER**(v) (morpho_isnumber(v))
- #define MORPHO_INTEGERTOFLOAT(x) (MORPHO_FLOAT((double) MORPHO_GETINTEGERVALUE((x))))
- #define **MORPHO_ISFALSE**(x) (morpho_isfalse(x))
- #define **MORPHO_ISTRUE**(x) (!morpho_isfalse(x))

Typedefs

- typedef struct sobject **object**

Enumerations

- enum valuetype {
  VALUE_NIL, **VALUE_INTEGER**, **VALUE_DOUBLE**, **VALUE_BOOL**,
  VALUE_OBJECT }

  A Morpho value.

Functions

- **DECLARE_VARRAY** (value, value)
- bool varray_valuefind (varray_value *varray, value v, unsigned int *out)

  Finds a value in an varray using a loose equality test (MORPHO_ISEQUAL)
- bool varray_valuefindsame (varray_value *varray, value v, unsigned int *out)

  Finds a value in an varray using strict equality test (MORPHO_ISSAME)
- bool value_promotenumberlist (unsigned int nv, value *v)

10.18.1 Detailed Description

Fundamental data type for morpho.

Author

T J Atherton

10.18.2 Macro Definition Documentation

10.18.2.1 MORPHO_GETINTEGERVALUE

```
#define MORPHO_GETINTEGERVALUE(
              v ) ((v).as.integer)
```

Get a value

10.18.2.2 MORPHO_GETTYPE

```
#define MORPHO_GETTYPE(
              v ) ((v).type)
```

This macro gets the type of the value.

Warning

Not intended for broad use.

10.18.2.3 MORPHO_INTEGERTOFLOAT

```
#define MORPHO_INTEGERTOFLOAT(
              x ) (MORPHO_FLOAT((double) MORPHO_GETINTEGERVALUE((x))))
```

Conversion

10.18.2.4 MORPHO_ISNIL

```
#define MORPHO_ISNIL(
              v ) ((v).type==VALUE_NIL)
```

Test for the type of a value

10.18.2.5 MORPHO_NIL

```
#define MORPHO_NIL ((value) { VALUE_NIL, .as.integer = (int) 0 })
```

Create a literal

10.18.3 Enumeration Type Documentation

10.18.3.1 valuetype

```
enum valuetype
```

A Morpho value.

A enumerated type defining the different types available in Morpho.

10.18.4 Function Documentation

10.18.4.1 value_promotenumberlist()

```
bool value_promotenumberlist (
            unsigned int nv,
            value * v )
```

Promotes a list of numbers to floats if any are floating point.

Parameters

in	*nv*	- number of values
in	*v*	- list of values

Returns

true if successful, false if any values are not numbers

10.18.4.2 varray_valuefind()

```
bool varray_valuefind (
            varray_value * varray,
            value v,
            unsigned int * out )
```

Finds a value in an varray using a loose equality test (MORPHO_ISEQUAL)

Parameters

in	*varray*	the array to search
in	*v*	value to find
out	*out*	index of the match

Returns

whether the value was found or not.

10.18.4.3 varray_valuefindsame()

```
bool varray_valuefindsame (
            varray_value * varray,
            value v,
            unsigned int * out )
```

Finds a value in an varray using strict equality test (MORPHO_ISSAME)

Parameters

in	*varray*	the array to search
in	*v*	value to find
out	*out*	index of the match

Returns

whether the value was found or not.

10.19 datastructures/varray.c File Reference

Dynamically resizing array (varray) data structure.

```
#include "varray.h"
```

Functions

- **DEFINE_VARRAY** (char, char)
- **DEFINE_VARRAY** (double, double)
- unsigned int varray_powerof2ceiling (unsigned int n)

  Computes the nearest power of 2 above an integer.

10.19.1 Detailed Description

Dynamically resizing array (varray) data structure.

Author

T J Atherton

10.19.2 Function Documentation

10.19.2.1 varray_powerof2ceiling()

```
unsigned int varray_powerof2ceiling (
            unsigned int n )
```

Computes the nearest power of 2 above an integer.

Parameters

n	An integer

Returns

Nearest power of 2 above n See: http://graphics.stanford.edu/~seander/bithacks.html#RoundUpPowerOf2Float

10.20 datastructures/varray.h File Reference

Dynamically resizing array (varray) data structure.

```
#include <stdlib.h>
#include <stdbool.h>
#include "memory.h"
```

Macros

- #define DECLARE_VARRAY(name, type)

  Creates a generic varray containing a specified type.
- #define **DEFINE_VARRAY**(name, type)

Functions

- **DECLARE_VARRAY** (char, char)
- **DECLARE_VARRAY** (double, double)
- unsigned int varray_powerof2ceiling (unsigned int n)

  Computes the nearest power of 2 above an integer.

10.20.1 Detailed Description

Dynamically resizing array (varray) data structure.

Author

T J Atherton

10.20.2 Macro Definition Documentation

10.20.2.1 DECLARE_VARRAY

```
#define DECLARE_VARRAY(
                name,
                type )
```

Value:

```
    typedef struct { \
        unsigned int count; \
        unsigned int capacity; \
        type *data; \
    } varray_##name; \
    \
    void varray_##name##init(varray_##name *v); \
    bool varray_##name##add(varray_##name *v, type *data, int count); \
    bool varray_##name##resize(varray_##name *v, int count); \
    int varray_##name##write(varray_##name *v, type data); \
    void varray_##name##clear(varray_##name *v);
```

Creates a generic varray containing a specified type.

Variable array macros

Varrays only differ by their contents, and so we use macros to conveniently define types and functions. To use these: First, call DECLARE_VARRAY(NAME,TYPE) in your .h file with a selected name for your varray and the type of thing you want to store in it. This will define:

1. A type called varray_NAME (where NAME is the name you gave).
2. Functions varray_NAME_init(v) - Initializes the varray varray_NAME_add(v, data[ ], count) - Adds elements to the varray varray_NAME_write(v, data) - Writes a single element to the varray, returning the index varray_N↩ AME_clear(v) - Clears the varray, freeing memory Then, call DEFINE_VARRAY(NAME,TYPE) in your .c file to define the appropriate functions

10.20.3 Function Documentation

10.20.3.1 varray_powerof2ceiling()

```
unsigned int varray_powerof2ceiling (
            unsigned int n )
```

Computes the nearest power of 2 above an integer.

Parameters

n	An integer

Returns

Nearest power of 2 above n See: http://graphics.stanford.edu/~seander/bithacks.html#RoundUpPowerOf2Float

10.21 geometry/mesh.c File Reference

Mesh class and associated functionality.

```
#include "object.h"
#include "builtin.h"
#include "mesh.h"
#include "file.h"
#include "varray.h"
#include "parse.h"
#include "sparse.h"
```

Functions

- objectmesh * object_newmesh (unsigned int dim, unsigned int nv, double *v)
- bool mesh_checkconnectivity (objectmesh *mesh)
- objectsparse * mesh_getconnectivityelement (objectmesh *mesh, unsigned int row, unsigned int col, bool create)
- bool mesh_addelementwithvertices (objectmesh *mesh, grade g, vertexid *v)
- objectmesh * mesh_load (char *file)
- value mesh_constructor (vm *v, int nargs, value *args)
- value **Mesh_print** (vm *v, int nargs, value *args)
- MORPHO_ENDCLASS void **mesh_initialize** (void)

10.21.1 Detailed Description

Mesh class and associated functionality.

Author

T J Atherton

10.21.2 Function Documentation

10.21.2.1 mesh_addelementwithvertices()

```
bool mesh_addelementwithvertices (
            objectmesh * mesh,
            grade g,
            vertexid * v )
```

Adds an element to a mesh

Parameters

in	*mesh*	the mesh
in	*g*	grade of the element
in	*v*	vertexids

10.21.2.2 mesh_checkconnectivity()

```
bool mesh_checkconnectivity (
            objectmesh * mesh )
```

Ensures a mesh has a valid connectivity matrix

10.21.2.3 mesh_constructor()

```
value mesh_constructor (
            vm * v,
            int nargs,
            value * args )
```

Constructs a Matrix object

10.21.2.4 mesh_getconnectivityelement()

```
objectsparse* mesh_getconnectivityelement (
            objectmesh * mesh,
            unsigned int row,
            unsigned int col,
            bool create )
```

Gets the connectivity matrix corresponding to (row, col); creates one if create is set

10.21.2.5 mesh_load()

```
objectmesh* mesh_load (
            char * file )
```

Loads a .mesh file.

10.21.2.6 object_newmesh()

```
objectmesh* object_newmesh (
            unsigned int dim,
            unsigned int nv,
            double * v )
```

Creates a mesh object

10.22 geometry/mesh.h File Reference

Mesh class and associated functionality.

Macros

- #define **MESH_CLASSNAME** "Mesh"
- #define **MESH_VERTSECTION** "vertices"
- #define **MESH_EDGESECTION** "edges"
- #define **MESH_FACESECTION** "faces"
- #define **MESH_VOLSECTION** "volumes"

Typedefs

- typedef int **grade**
- typedef int **vertexid**

Functions

- void **mesh_initialize** (void)

10.22.1 Detailed Description

Mesh class and associated functionality.

Author

T J Atherton

10.23 interface/cli.c File Reference

Command line interface.

```
#include <time.h>
#include "cli.h"
#include "parse.h"
```

Macros

- #define **CLI_BUFFERSIZE** 1024

Functions

- void cli_reporterror (error *err, vm *v)
- bool cli_lex (char *in, void **ref, linedit_token *out)
- bool cli_complete (char *in, linedit_stringlist *c)
- void cli_help (lineditor *edit, char *query, error *err, bool avail)
- void cli (clioptions opt)

  Provide a command line interface.
- void cli_disassemblewithsrc (program *p, char *src)
- void cli_run (const char *in, clioptions opt)

Variables

- linedit_color cli_tokencolors [ ]

10.23.1 Detailed Description

Command line interface.

Author

T J Atherton

10.23.2 Function Documentation

10.23.2.1 cli_complete()

```
bool cli_complete (
            char * in,
            linedit_stringlist * c )
```

Autocomplete function

10.23.2.2 cli_disassemblewithsrc()

```
void cli_disassemblewithsrc (
            program * p,
            char * src )
```

Disassembles the program showing syntax colored lines of source

10.23.2.3 cli_help()

```
void cli_help (
            lineditor * edit,
            char * query,
            error * err,
            bool avail )
```

Interactive help

10.23.2.4 cli_lex()

```
bool cli_lex (
            char * in,
            void ** ref,
            linedit_token * out )
```

A tokenizer for syntax coloring that leverages the parser's lexer

10.23.2.5 cli_reporterror()

```
void cli_reporterror (
            error * err,
            vm * v )
```

Report an error if one has occurred.

10.23.2.6 cli_run()

```
void cli_run (
            const char * in,
            clioptions opt )
```

Loads and runs a file.

10.23.3 Variable Documentation

10.23.3.1 cli_tokencolors

```
linedit_color cli_tokencolors[]
```

Define colors for different token types

10.24 interface/cli.h File Reference

Command line interface.

```
#include "morpho.h"
#include "varray.h"
#include "error.h"
#include "linedit.h"
#include "help.h"
```

Macros

- #define **CLI_PROMPT** ">"
- #define **CLI_CONTINUATIONPROMPT** "~"
- #define **CLI_QUIT** "quit"
- #define **CLI_HELP** "help"
- #define **CLI_SHORT_HELP** "?"
- #define **CLI_NORMALCODE** "\033[0m"
- #define **CLI_REDCODE** "\033[0;31m"
- #define **CLI_BLUECODE** "\033[0;34m"
- #define **CLI_ERRORCOLOR** ""
- #define **CLI_NORMALTEXT** ""
- #define **CLI_RUN** 0x1
- #define **CLI_DISASSEMBLE** 0x2
- #define **CLI_DISASSEMBLESHOWSRC** 0x4

Typedefs

- typedef unsigned int **clioptions**

Functions

- void cli_run (const char *in, clioptions opt)
- void cli (clioptions opt)

  Provide a command line interface.

10.24.1 Detailed Description

Command line interface.

Author

T J Atherton

10.24.2 Function Documentation

10.24.2.1 cli_run()

```
void cli_run (
            const char * in,
            clioptions opt )
```

Loads and runs a file.

10.25 interface/help.c File Reference

Interactive help system.

```
#include <string.h>
#include <ctype.h>
#include <dirent.h>
#include "help.h"
#include "dictionary.h"
#include "parse.h"
#include "common.h"
```

Macros

- #define **HELP_LINELENGTH** 2048
- #define **HELP_MAXLEVEL** 6

Functions

- objecthelptopic * help_newtopic (char *topic, char *file, long int location, objecthelptopic *parent)
- void help_cleartopic (objecthelptopic *topic)
- size_t help_querylength (char *query, char **s)
- objecthelptopic * help_search (char *query)
- void help_display (lineditor *edit, objecthelptopic *topic)
- value help_parsetopicname (char *line)
- value help_parsetag (char *line)
- int help_parsetopiclevel (char *line)
- bool help_load (char *file)
- bool help_searchpath (char *path)
- bool help_initialize (void)
- void help_finalize (void)

10.25.1 Detailed Description

Interactive help system.

Author

T J Atherton

10.25.2 Function Documentation

10.25.2.1 help_cleartopic()

```
void help_cleartopic (
            objecthelptopic * topic )
```

Free attached data from a help topic

10.25.2.2 help_display()

```
void help_display (
            lineditor * edit,
            objecthelptopic * topic )
```

Displays a help topic

10.25.2.3 help_finalize()

```
void help_finalize (
            void  )
```

Finalizes the help system

10.25.2.4 help_initialize()

```
bool help_initialize (
            void  )
```

Initializes the help system

Returns

true if help is available

10.25.2.5 help_load()

```
bool help_load (
            char * file )
```

Loads a help file

Parameters

file	file to load

Returns

true if any help entries were successfully loaded

10.25.2.6 help_newtopic()

```
objecthelptopic* help_newtopic (
            char * topic,
            char * file,
            long int location,
            objecthelptopic * parent )
```

Create a new help topic

10.25.2.7 help_parsetag()

```
value help_parsetag (
            char * line )
```

Parses a tag, returning it as a Morpho string converted to lower case.

Warning

the input line is modified in the process

10.25.2.8 help_parsetopiclevel()

```
int help_parsetopiclevel (
            char * line )
```

Determines the level of topic from the markdown header level

10.25.2.9 help_parsetopicname()

```
value help_parsetopicname (
            char * line )
```

Parses a topic name, returning it as a Morpho string converted to lower case.

Warning

the input line is modified in the process

10.25.2.10 help_querylength()

```
size_t help_querylength (
            char * query,
            char ** s )
```

Determine starting point and length of a query

Parameters

in	*query*	- the query to examine
out	*s*	- starting character of the query (optional)

Returns

length of the query (0 indicates no query present)

10.25.2.11 help_search()

```
objecthelptopic* help_search (
            char * query )
```

Searches for a given query in the help system

10.25.2.12 help_searchpath()

```
bool help_searchpath (
            char * path )
```

Searches for help files

Parameters

path	directory to search (recursively)

Returns

true if any help files were successfully processed.

10.26 interface/help.h File Reference

Interactive help system.

```
#include <stdio.h>
#include "morpho.h"
#include "object.h"
#include "linedit.h"
```

Classes

- struct sobjecthelptopic

Macros

- #define **HELP_INDEXPAGE** "help"

Typedefs

- typedef struct sobjecthelptopic **objecthelptopic**

Functions

- size_t help_querylength (char *query, char **s)
- objecthelptopic * help_search (char *query)
- void help_display (lineditor *edit, objecthelptopic *topic)
- bool help_initialize (void)
- void help_finalize (void)

10.26.1 Detailed Description

Interactive help system.

Author

T J Atherton

10.26.2 Function Documentation

10.26.2.1 help_display()

```
void help_display (
            lineditor * edit,
            objecthelptopic * topic )
```

Displays a help topic

10.26.2.2 help_finalize()

```
void help_finalize (
            void  )
```

Finalizes the help system

10.26.2.3 help_initialize()

```
bool help_initialize (
            void  )
```

Initializes the help system

Returns

true if help is available

10.26.2.4 help_querylength()

```
size_t help_querylength (
            char * query,
            char ** s )
```

Determine starting point and length of a query

Parameters

in	*query*	- the query to examine
out	*s*	- starting character of the query (optional)

Returns

length of the query (0 indicates no query present)

10.26.2.5 help_search()

```
objecthelptopic* help_search (
            char * query )
```

Searches for a given query in the help system

10.27 interface/linedit.c File Reference

A line editor with history, autocomplete and syntax highlighting.

```
#include "linedit.h"
```

Classes

- struct keypress

Macros

- #define **linedit_MINIMUMSTRINGSIZE** 8
- #define LINEDIT_CODESTRINGSIZE 24
- #define LINEDIT_DEBUGKEYPRESS
- #define LINEDIT_UNSUPPORTEDBUFFER 4096

Enumerations

- enum keytype {
  UNKNOWN, **CHARACTER**, **RETURN**, **TAB**,
  DELETE, **UP**, **DOWN**, **LEFT**,
  RIGHT, **HOME**, **END**, **SHIFT_LEFT**,
  SHIFT_RIGHT, **CTRL** }
- enum **linedit_terminaltype** { **LINEDIT_NOTTTY**, **LINEDIT_UNSUPPORTED**, **LINEDIT_SUPPORTED** }
- enum keycodes { **TAB_CODE** = 9, **RETURN_CODE** = 13, **ESC_CODE** = 27, **DELETE_CODE** = 127 }

Functions

- void linedit_stringinit (linedit_string *string)
- void linedit_stringclear (linedit_string *string)
- bool linedit_stringresize (linedit_string *string, size_t size)
- void linedit_stringaddcharacter (linedit_string *string, char *c, size_t n)
- void linedit_stringinsert (linedit_string *string, size_t posn, char *c, size_t n)

  Inserts characters at a given position.
- void linedit_stringdelete (linedit_string *string, size_t posn, size_t n)

  Deletes characters at a given position.
- void linedit_stringaddcstring (linedit_string *string, char *s)
- char * linedit_cstring (linedit_string *string)
- linedit_string * linedit_newstring (char *string)
- void linedit_stringlistadd (linedit_stringlist *list, char *string)
- void linedit_stringlistinit (linedit_stringlist *list)
- void linedit_stringlistclear (linedit_stringlist *list)
- void linedit_stringlistremove (linedit_stringlist *list, linedit_string *string)
- linedit_string * linedit_stringlistselect (linedit_stringlist *list, unsigned int n, unsigned int *m)
- void linedit_historyadd (lineditor *edit, char *string)
- void linedit_historyclear (lineditor *edit)
- unsigned int linedit_historyselect (lineditor *edit, unsigned int n)
- void linedit_historyadvance (lineditor *edit, unsigned int n)
- bool lineedit_atendofline (lineditor *edit)
- void linedit_generatesuggestions (lineditor *edit)
- bool linedit_aresuggestionsavailable (lineditor *edit)
- char * linedit_currentsuggestion (lineditor *edit)
- void linedit_advancesuggestions (lineditor *edit, unsigned int n)
- void linedit_enablerawmode (void)

  Enables 'raw' mode in the terminal.
- void linedit_disablerawmode (void)

Restore terminal state to normal.

- int linedit_cstrcasecmp (char *str1, char *str2)

  Compares two c strings independently of case.
- linedit_terminaltype linedit_checksupport (void)
- void linedit_write (char *string)

  Writes a string to the terminal.
- void linedit_writechar (char c)

  Writes a character to the terminal.
- bool linedit_readkey (lineditor *edit, keypress *out)

  Read and decode a single keypress from the terminal.
- void linedit_move (linedit_string *out, int posn)

  Writes a control sequence to move to a given position.
- void linedit_setdefaulttext (linedit_string *out)

  Writes a control sequence to reset default text.
- void linedit_setcolor (linedit_string *out, linedit_color col)

  Writes a control sequence to set a given color.
- void linedit_setemphasis (linedit_string *out, linedit_emphasis emph)

  Writes a control sequence to set a given emphasis.
- void linedit_erasetoendofline (linedit_string *out)

  Writes a control sequence to erase the rest of the line.
- void linedit_eraseline (linedit_string *out)

  Writes a control sequence to erase the whole line.
- void **linedit_addcstringwithselection** (lineditor *edit, char *in, size_t offset, size_t length, linedit_color *col, linedit_string *out)
- void linedit_syntaxcolorstring (lineditor *edit, linedit_string *in, linedit_string *out)
- void linedit_showstring (lineditor *edit, linedit_string *in, linedit_string *out)
- void linedit_refreshline (lineditor *edit)
- void linedit_setmode (lineditor *edit, lineditormode mode)
- lineditormode linedit_getmode (lineditor *edit)
- void linedit_setposition (lineditor *edit, int posn)
- void linedit_advanceposition (lineditor *edit, int delta)

  Advances the position by delta.
- bool linedit_processkeypress (lineditor *edit)
- void linedit_noterminal (lineditor *edit)
- void **linedit_unsupported** (lineditor *edit)
- void linedit_supported (lineditor *edit)
- char * linedit (lineditor *edit)
- void linedit_syntaxcolor (lineditor *edit, linedit_tokenizer tokenizer, linedit_color *cols, unsigned int ncols)

  Configures syntax coloring.
- void linedit_autocomplete (lineditor *edit, linedit_completer completer)

  Configures autocomplete.
- void linedit_addsuggestion (linedit_stringlist *completion, char *string)

  Adds a completion suggestion.
- void linedit_setprompt (lineditor *edit, char *prompt)

  Sets the prompt.
- void linedit_displaywithstyle (lineditor *edit, char *string, linedit_color col, linedit_emphasis emph)

  Displays a string with a given color and emphasis.
- void linedit_displaywithsyntaxcoloring (lineditor *edit, char *string)

  Displays a string with syntax coloring.
- void linedit_init (lineditor *edit)
- void linedit_clear (lineditor *edit)

Variables

- struct termios terminit
- bool **termexitregistered** =false

10.27.1 Detailed Description

A line editor with history, autocomplete and syntax highlighting.

Author

T J Atherton

10.27.2 Macro Definition Documentation

10.27.2.1 LINEDIT_CODESTRINGSIZE

```
#define LINEDIT_CODESTRINGSIZE 24
```

Maximum escape code size

10.27.2.2 LINEDIT_DEBUGKEYPRESS

```
#define LINEDIT_DEBUGKEYPRESS
```

Enable this macro to get reports on unhandled keypresses

10.27.2.3 LINEDIT_UNSUPPORTEDBUFFER

```
#define LINEDIT_UNSUPPORTEDBUFFER 4096
```

If the terminal is unsupported, default to fgets with a fixed buffer

10.27.3 Enumeration Type Documentation

10.27.3.1 keycodes

```
enum keycodes
```

Raw codes produced by the terminal

10.27.3.2 keytype

```
enum keytype
```

Identifies the type of keypress

10.27.4 Function Documentation

10.27.4.1 linedit()

```
char* linedit (
            lineditor * edit )
```

Public interface to the line editor.

Parameters

edit	- a line editor that has been initialized with linedit_init.

Returns

the string input by the user, or NULL if nothing entered.

Ensure we are not passed a NULL pointer

10.27.4.2 linedit_addsuggestion()

```
void linedit_addsuggestion (
            linedit_stringlist * completion,
            char * string )
```

Adds a completion suggestion.

Parameters

completion	completion data structure
string	string to add

10.27.4.3 linedit_advanceposition()

```
void linedit_advanceposition (
            lineditor * edit,
            int delta )
```

Advances the position by delta.

We ensure that the current position also lies within the string.

10.27.4.4 linedit_advancesuggestions()

```
void linedit_advancesuggestions (
            lineditor * edit,
            unsigned int n )
```

Advance through the suggestions

10.27.4.5 linedit_aresuggestionsavailable()

```
bool linedit_aresuggestionsavailable (
            lineditor * edit )
```

Check whether any suggestions are available

10.27.4.6 linedit_autocomplete()

```
void linedit_autocomplete (
            lineditor * edit,
            linedit_completer completer )
```

Configures autocomplete.

Parameters

edit	Line editor to configure
completer	a function

10.27.4.7 linedit_checksupport()

```
linedit_terminaltype linedit_checksupport (
            void  )
```

Checks whether the terminal is supported

10.27.4.8 linedit_clear()

```
void linedit_clear (
            lineditor * edit )
```

Finalize a line editor

10.27.4.9 linedit_cstrcasecmp()

```
int linedit_cstrcasecmp (
            char * str1,
            char * str2 )
```

Compares two c strings independently of case.

Parameters

in	*str1*	- } strings to compare
in	*str2*	- }

Returns

0 if the strings are identical, otherwise a positive or negative number indicating their lexographic order

10.27.4.10 linedit_cstring()

```
char* linedit_cstring (
            linedit_string * string )
```

Returns a C string from a string

10.27.4.11 linedit_currentsuggestion()

```
char* linedit_currentsuggestion (
            lineditor * edit )
```

Get the current suggestion

10.27.4.12 linedit_disablerawmode()

```
void linedit_disablerawmode (
            void  )
```

Restore terminal state to normal.

Print a carriage return to ensure we're back on the left hand side

10.27.4.13 linedit_displaywithstyle()

```
void linedit_displaywithstyle (
            lineditor * edit,
            char * string,
            linedit_color col,
            linedit_emphasis emph )
```

Displays a string with a given color and emphasis.

Parameters

edit	Line editor in use
string	String to display

10.27.4.14 linedit_displaywithsyntaxcoloring()

```
void linedit_displaywithsyntaxcoloring (
            lineditor * edit,
            char * string )
```

Displays a string with syntax coloring.

Parameters

edit	Line editor in use
string	String to display

10.27.4.15 linedit_enablerawmode()

```
void linedit_enablerawmode (
            void  )
```

Enables 'raw' mode in the terminal.

In raw mode key presses are passed directly to us rather than being buffered. Get the original state

10.27.4.16 linedit_generatesuggestions()

```
void linedit_generatesuggestions (
            lineditor * edit )
```

Regenerates the list of autocomplete suggestions

10.27.4.17 linedit_getmode()

```
lineditormode linedit_getmode (
            lineditor * edit )
```

Gets the current mode

10.27.4.18 linedit_historyadd()

```
void linedit_historyadd (
            lineditor * edit,
            char * string )
```

Adds an entry to the history list

10.27.4.19 linedit_historyadvance()

```
void linedit_historyadvance (
            lineditor * edit,
            unsigned int n )
```

Advances the history list

10.27.4.20 linedit_historyclear()

```
void linedit_historyclear (
            lineditor * edit )
```

Frees the history list

10.27.4.21 linedit_historyselect()

```
unsigned int linedit_historyselect (
            lineditor * edit,
            unsigned int n )
```

Makes a particular history entry current

10.27.4.22 linedit_init()

```
void linedit_init (
            lineditor * edit )
```

Initialize a line editor

10.27.4.23 linedit_newstring()

```
linedit_string* linedit_newstring (
            char * string )
```

Creates a new string from a C string

10.27.4.24 linedit_noterminal()

```
void linedit_noterminal (
            lineditor * edit )
```

If we're not attached to a terminal, e.g. a pipe, simply read the file in.

10.27.4.25 linedit_processkeypress()

```
bool linedit_processkeypress (
            lineditor * edit )
```

Obtain and process a single keypress

10.27.4.26 linedit_readkey()

```
bool linedit_readkey (
            lineditor * edit,
            keypress * out )
```

Read and decode a single keypress from the terminal.

Decode the escape sequence

10.27.4.27 linedit_refreshline()

```
void linedit_refreshline (
            lineditor * edit )
```

Refreshes a single line

10.27.4.28 linedit_setmode()

```
void linedit_setmode (
            lineditor * edit,
            lineditormode mode )
```

Sets the current mode, setting/clearing any state dependent data

10.27.4.29 linedit_setposition()

```
void linedit_setposition (
            lineditor * edit,
            int posn )
```

Sets the current position

Parameters

edit	- the editor
posn	- position to set, or negative to move to end

10.27.4.30 linedit_setprompt()

```
void linedit_setprompt (
            lineditor * edit,
            char * prompt )
```

Sets the prompt.

Parameters

edit	Line editor to configure
prompt	prompt string to use

10.27.4.31 linedit_showstring()

```
void linedit_showstring (
            lineditor * edit,
            linedit_string * in,
            linedit_string * out )
```

Print a string without syntax coloring

10.27.4.32 linedit_stringaddcharacter()

```
void linedit_stringaddcharacter (
            linedit_string * string,
            char * c,
            size_t n )
```

Adds a character to a string

10.27.4.33 linedit_stringaddcstring()

```
void linedit_stringaddcstring (
            linedit_string * string,
            char * s )
```

Adds a c string to a string

10.27.4.34 linedit_stringclear()

```
void linedit_stringclear (
            linedit_string * string )
```

Clears a string, deallocating memory if necessary

10.27.4.35 linedit_stringinit()

```
void linedit_stringinit (
            linedit_string * string )
```

Initializes a string, clearing all fields

10.27.4.36 linedit_stringinsert()

```
void linedit_stringinsert (
            linedit_string * string,
            size_t posn,
            char * c,
            size_t n )
```

Inserts characters at a given position.

If the position is after the length of the string the new characters are instead appended.

10.27.4.37 linedit_stringlistadd()

```
void linedit_stringlistadd (
            linedit_stringlist * list,
            char * string )
```

Adds an entry to a string list

10.27.4.38 linedit_stringlistclear()

```
void linedit_stringlistclear (
            linedit_stringlist * list )
```

Frees the contents of a string list

10.27.4.39 linedit_stringlistinit()

```
void linedit_stringlistinit (
            linedit_stringlist * list )
```

Initializes a string list

10.27.4.40 linedit_stringlistremove()

```
void linedit_stringlistremove (
            linedit_stringlist * list,
            linedit_string * string )
```

Removes a string from a list

10.27.4.41 linedit_stringlistselect()

```
linedit_string* linedit_stringlistselect (
            linedit_stringlist * list,
            unsigned int n,
            unsigned int * m )
```

Chooses an element of a stringlist

Parameters

in	*list*	the list to select from
in	*n*	entry number to select @parma[out] ∗m entry number actually selected

Returns

the selected element

10.27.4.42 linedit_stringresize()

```
bool linedit_stringresize (
            linedit_string * string,
            size_t size )
```

Resizes a string

Parameters

string	- the string to grow
size	- requested size

Returns

true on success, false on failure

10.27.4.43 linedit_supported()

```
void linedit_supported (
            lineditor * edit )
```

Normal interface used if terminal is present

10.27.4.44 linedit_syntaxcolor()

```
void linedit_syntaxcolor (
            lineditor * edit,
            linedit_tokenizer tokenizer,
            linedit_color * cols,
            unsigned int ncols )
```

Configures syntax coloring.

Parameters

edit	Line editor to configure
tokenizer	A function to be called that will find the next token from a string
cols	An array of colors, one entry for each token type
ncols	Number of entries in the color array

10.27.4.45 linedit_syntaxcolorstring()

```
void linedit_syntaxcolorstring (
            lineditor * edit,
            linedit_string * in,
            linedit_string * out )
```

Print a string with syntax coloring

10.27.4.46 lineedit_atendofline()

```
bool lineedit_atendofline (
            lineditor * edit )
```

Checks if we're at the end of the line

10.27.5 Variable Documentation

10.27.5.1 terminit

```
struct termios terminit
```

Holds the original terminal state

10.28 interface/linedit.h File Reference

A simple line editor with history, prediction and syntax highlighting.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <ctype.h>
#include <string.h>
#include <termios.h>
#include <unistd.h>
```

Classes

- struct slinedit_string
- struct linedit_stringlist
- struct linedit_token
- struct linedit_syntaxcolordata
- struct lineditor

Macros

- #define **LINEDIT_DEFAULTPROMPT** ">"

Typedefs

- typedef struct slinedit_string linedit_string
- typedef bool(∗ linedit_tokenizer) (char ∗in, void ∗∗ref, linedit_token ∗tok)

  Tokenizer callback.
- typedef bool(∗ linedit_completer) (char ∗in, linedit_stringlist ∗completion)

  Autocompletion callback @params in - a string @params completion - autocompletion structure.

Enumerations

- enum linedit_color { **LINEDIT_BLACK**, **LINEDIT_RED**, **LINEDIT_GREEN**, **LINEDIT_YELLOW**, **LINEDIT_BLUE**, **LINEDIT_MAGENTA**, **LINEDIT_CYAN**, **LINEDIT_WHITE**, **LINEDIT_DEFAULTCOLOR** }
- enum **linedit_emphasis** { **LINEDIT_BOLD**, **LINEDIT_UNDERLINE**, **LINEDIT_REVERSE**, **LINEDIT_NONE** }
- enum lineditormode { **LINEDIT_DEFAULTMODE**, **LINEDIT_SELECTIONMODE**, **LINEDIT_HISTORYMODE** }

Functions

- char * linedit (lineditor *edit)
- void linedit_syntaxcolor (lineditor *edit, linedit_tokenizer tokenizer, linedit_color *cols, unsigned int ncols)

  Configures syntax coloring.
- void linedit_autocomplete (lineditor *edit, linedit_completer completer)

  Configures autocomplete.
- void linedit_addsuggestion (linedit_stringlist *completion, char *string)

  Adds a completion suggestion.
- void linedit_setprompt (lineditor *edit, char *prompt)

  Sets the prompt.
- void linedit_displaywithstyle (lineditor *edit, char *string, linedit_color col, linedit_emphasis emph)

  Displays a string with a given color and emphasis.
- void linedit_displaywithsyntaxcoloring (lineditor *edit, char *string)

  Displays a string with syntax coloring.
- void linedit_init (lineditor *edit)
- void linedit_clear (lineditor *edit)

10.28.1 Detailed Description

A simple line editor with history, prediction and syntax highlighting.

Author

T J Atherton

10.28.2 Typedef Documentation

10.28.2.1 linedit_completer

```
typedef bool(* linedit_completer) (char *in, linedit_stringlist *completion)
```

Autocompletion callback @params in - a string @params completion - autocompletion structure.

This user function is called when linedit requests autocompletion of a string. The function should identify any possible suggestions and call linedit_addcompletion to add them one by one.

Only *remaining* characters from the suggestion should be added, e.g. for "hello" if the user has typed "he" the function should add "llo" as a suggestion.

The function should return true if autocompletion was successfully processed or false otherwise.

10.28.2.2 linedit_string

```
typedef struct slinedit_string linedit_string
```

lineditor strings

10.28.2.3 linedit_tokenizer

```
typedef bool(* linedit_tokenizer) (char *in, void **ref, linedit_token *tok)
```

Tokenizer callback.

Parameters

in	- a string
ref	- System for storing persistent data between calls to the lexer.
tok	- pointer to a token structure that the caller should fill out. This user function is called when linedit needs to tokenize a string. The function should identify the next token in the string and fill out the following fields: tok->type - should contain the token type. This is used e.g. an index to the color array. tok->start - should point to the first significant character in the token tok->length - should contain the length of the token, in bytes The function should return true if a token was successfully processed or false otherwise.

Storing persistent data over a sequance of calls to implement non-CFG: On the first call, ∗ref is NULL. You should malloc your structure, initialize it, set ∗ref to point to it and return the first token. After that, ∗ref will point to your structure. Once linedit is done tokenizing, it will call free on your pointer.

Warning

: You must not malloc child structures as these will not be freed.

10.28.3 Enumeration Type Documentation

10.28.3.1 linedit_color

```
enum linedit_color
```

Colors

10.28.3.2 lineditormode

```
enum lineditormode
```

Keep track of what the line editor is doing

10.28.4 Function Documentation

10.28.4.1 linedit()

```
char* linedit (
            lineditor * edit )
```

Public interface to the line editor.

Parameters

edit	- a line editor that has been initialized with linedit_init.

Returns

the string input by the user, or NULL if nothing entered.

Ensure we are not passed a NULL pointer

10.28.4.2 linedit_addsuggestion()

```
void linedit_addsuggestion (
            linedit_stringlist * completion,
            char * string )
```

Adds a completion suggestion.

Parameters

completion	completion data structure
string	string to add

10.28.4.3 linedit_autocomplete()

```
void linedit_autocomplete (
            lineditor * edit,
            linedit_completer completer )
```

Configures autocomplete.

Parameters

edit	Line editor to configure
completer	a function

10.28.4.4 linedit_clear()

```
void linedit_clear (
            lineditor * edit )
```

Finalize a line editor

10.28.4.5 linedit_displaywithstyle()

```
void linedit_displaywithstyle (
            lineditor * edit,
            char * string,
            linedit_color col,
            linedit_emphasis emph )
```

Displays a string with a given color and emphasis.

Parameters

edit	Line editor in use
string	String to display
col	Color
emph	Emphasis
edit	Line editor in use
string	String to display

10.28.4.6 linedit_displaywithsyntaxcoloring()

```
void linedit_displaywithsyntaxcoloring (
            lineditor * edit,
            char * string )
```

Displays a string with syntax coloring.

Parameters

edit	Line editor in use
string	String to display

10.28.4.7 linedit_init()

```
void linedit_init (
            lineditor * edit )
```

Initialize a line editor

10.28.4.8 linedit_setprompt()

```
void linedit_setprompt (
            lineditor * edit,
            char * prompt )
```

Sets the prompt.

Parameters

edit	Line editor to configure
prompt	prompt string to use

10.28.4.9 linedit_syntaxcolor()

```
void linedit_syntaxcolor (
            lineditor * edit,
            linedit_tokenizer tokenizer,
            linedit_color * cols,
            unsigned int ncols )
```

Configures syntax coloring.

Parameters

edit	Line editor to configure
tokenizer	A function to be called that will find the next token from a string
cols	An array of colors, one entry for each token type
ncols	Number of entries in the color array

10.29 main.c File Reference

Main entry point.

```
#include <stdio.h>
#include <stdarg.h>
#include "cli.h"
#include "value.h"
#include "object.h"
#include "random.h"
#include "sparse.h"
```

Functions

- int **main** (int argc, const char *argv[ ])

10.29.1 Detailed Description

Main entry point.

Author

T J Atherton

10.30 morpho.h File Reference

Define public interface to Morpho.

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "build.h"
#include "value.h"
#include "error.h"
```

Macros

- #define **MORPHO_INITIALIZER_METHOD** "init"
- #define **MORPHO_GETINDEX_METHOD** "index"
- #define **MORPHO_SETINDEX_METHOD** "setindex"
- #define **MORPHO_ADD_METHOD** "add"
- #define **MORPHO_SUB_METHOD** "sub"
- #define **MORPHO_MUL_METHOD** "mul"
- #define **MORPHO_DIV_METHOD** "div"
- #define **MORPHO_ENUMERATE_METHOD** "enumerate"
- #define **MORPHO_PRINT_METHOD** "prnt"

Typedefs

- typedef void **vm**
- typedef void **program**
- typedef void **compiler**

Functions

- void morpho_writeerrorwithid (error *err, errorid id, int line, int position,...)

  Writes an error message to an error structure.
- void morpho_defineerror (errorid id, errorcategory cat, char *message)
- errorid morpho_geterrorid (error *err)
- program * morpho_newprogram (void)

  Creates and initializes a new program.
- void morpho_freeprogram (program *p)

  Frees a program.
- vm * morpho_newvm (void)
- void morpho_freevm (vm *v)
- void morpho_runtimeerror (vm *v, errorid id,...)

  Public interface to raise a runtime error.
- compiler * morpho_newcompiler (program *out)
- void morpho_freecompiler (compiler *c)
- bool morpho_compile (char *in, compiler *c, error *err)
- const char * **morpho_compilerrestartpoint** (compiler *c)
- void **morpho_resetentry** (program *p)
- bool morpho_interpret (vm *v, program *code, error *err)

  Executes a sequence of code.
- void morpho_disassemble (program *code, unsigned int *matchline)
- void morpho_stacktrace (vm *v)
- void morpho_initialize (void)
- void morpho_finalize (void)

Variables

- value **initselector**
- value **indexselector**
- value **setindexselector**
- value **addselector**
- value **subselector**
- value **mulselector**
- value **divselector**
- value **printselector**
- value **enumerateselector**

10.30.1 Detailed Description

Define public interface to Morpho.

Author

T J Atherton

10.30.2 Function Documentation

10.30.2.1 morpho_compile()

```
bool morpho_compile (
            char * in,
            compiler * c,
            error * err )
```

Interface to the compiler

Parameters

in	*in*	A string to compile
in	*c*	The compiler
out	*err*	Pointer to error block on failure

Returns

A bool indicating success or failure

10.30.2.2 morpho_defineerror()

```
void morpho_defineerror (
            errorid id,
```

```
            errorcategory cat,
            char * message )
```

Defines an error

Parameters

id	Error struct to fill out
cat	The category of error
message	The message string

10.30.2.3 morpho_disassemble()

```
void morpho_disassemble (
            program * code,
            unsigned int * matchline )
```

Disassembles a program

Parameters

code	- program to disassemble
matchline	- optional line number to match

10.30.2.4 morpho_finalize()

```
void morpho_finalize (
            void  )
```

Finalizes morpho

10.30.2.5 morpho_freecompiler()

```
void morpho_freecompiler (
            compiler * c )
```

Frees a compiler

10.30.2.6 morpho_freevm()

```
void morpho_freevm (
            vm * v )
```

Frees a virtual machine

10.30.2.7 morpho_geterrorid()

```
errorid morpho_geterrorid (
            error * err )
```

Gets the id of an error

10.30.2.8 morpho_initialize()

```
void morpho_initialize (
            void  )
```

Initializes morpho

10.30.2.9 morpho_interpret()

```
bool morpho_interpret (
            vm * v,
            program * code,
            error * err )
```

Executes a sequence of code.

Parameters

v	The virtual machine to use
code	The program to execute

Returns

A morpho error

10.30.2.10 morpho_newcompiler()

```
compiler* morpho_newcompiler (
            program * out )
```

Creates a new compiler

10.30.2.11 morpho_newvm()

```
vm* morpho_newvm (
            void  )
```

Creates a new virtual machine

10.30.2.12 morpho_runtimeerror()

```
void morpho_runtimeerror (
             vm * v,
             errorid id,
              ...  )
```

Public interface to raise a runtime error.

Parameters

v	the virtual machine
id	error id
...	additional data for sprintf.

10.30.2.13 morpho_stacktrace()

```
void morpho_stacktrace (
             vm * v )
```

Prints a stacktrace

10.30.2.14 morpho_writeerrorwithid()

```
void morpho_writeerrorwithid (
             error * err,
             errorid id,
             int line,
             int posn,
              ...  )
```

Writes an error message to an error structure.

Parameters

err	The error structure
id	The error id.
line	The line at which the error occurred, if identifiable.
posn	The position in the line at which the error occurred, if identifiable.
...	Additional parameters (the data for the printf commands in the message)

10.31 utils/common.c File Reference

Common types, data structures and functions for the Morpho VM.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <sys/stat.h>
#include "common.h"
#include "object.h"
```

Macros

- #define MORPHO_TOSTRINGTMPBUFFERSIZE 64

  Concatenates a sequence of values as a string.

Functions

- void morpho_printvalue (value v)

  Prints a value.
- value **morpho_concatenatestringvalues** (int nval, value *v)
- char * morpho_strdup (char *string)

  Duplicates a string.
- unsigned int morpho_powerof2ceiling (unsigned int n)

  Computes the nearest power of 2 above an integer.
- bool **morpho_isdirectory** (const char *path)

10.31.1 Detailed Description

Common types, data structures and functions for the Morpho VM.

Author

T J Atherton

10.31.2 Function Documentation

10.31.2.1 morpho_powerof2ceiling()

```
unsigned int morpho_powerof2ceiling (
            unsigned int n )
```

Computes the nearest power of 2 above an integer.

Parameters

n	An integer

Returns

Nearest power of 2 above n See: http://graphics.stanford.edu/~seander/bithacks.html#RoundUpPowerOf2Float

10.31.2.2 morpho_printvalue()

```
void morpho_printvalue (
            value v )
```

Prints a value.

Parameters

v	The value to print

10.31.2.3 morpho_strdup()

```
char* morpho_strdup (
            char * string )
```

Duplicates a string.

Parameters

string	String to duplicate

Warning

Caller should call MALLOC_FREE on the allocated string

10.32 utils/common.h File Reference

Morpho virtual machine.

```
#include <stddef.h>
#include <stdarg.h>
#include <math.h>
#include <float.h>
#include <string.h>
#include "value.h"
#include "object.h"
#include "error.h"
```

Macros

- #define **COMMON_NILSTRING** "nil"
- #define **COMMON_TRUESTRING** "true"
- #define **COMMON_FALSESTRING** "false"
- #define EQUAL 0

  Compares two values.
- #define **NOTEQUAL** 1
- #define **BIGGER** 1
- #define **SMALLER** -1
- #define MORPHO_ISEQUAL(a, b) (!morpho_comparevalue(a,b))
- #define MORPHO_ISSAME(a, b) (morpho_comparevaluesame(a,b))

Functions

- void morpho_printvalue (value v)

  Prints a value.
- value **morpho_concatenatestringvalues** (int nval, value *v)
- char * morpho_strdup (char *string)

  Duplicates a string.
- unsigned int morpho_powerof2ceiling (unsigned int n)

  Computes the nearest power of 2 above an integer.
- bool **morpho_isdirectory** (const char *path)

10.32.1 Detailed Description

Morpho virtual machine.

Author

T J Atherton

10.32.2 Macro Definition Documentation

10.32.2.1 EQUAL

```
#define EQUAL 0
```

Compares two values.

Parameters

a	value to compare
b	value to compare

Returns

0 if a and b are equal, a positive number if b>a and a negative number if a<b

10.32.2.2 MORPHO_ISEQUAL

```
#define MORPHO_ISEQUAL(
              a,
              b ) (!morpho_comparevalue(a,b))
```

Macros to compare values
Use this one to carefully compare the values in each object

10.32.2.3 MORPHO_ISSAME

```
#define MORPHO_ISSAME(
              a,
              b ) (morpho_comparevaluesame(a,b))
```

Use this one where we want to check the values refer to the same object

10.32.3 Function Documentation

10.32.3.1 morpho_powerof2ceiling()

```
unsigned int morpho_powerof2ceiling (
             unsigned int n )
```

Computes the nearest power of 2 above an integer.

Parameters

n	An integer

Returns

Nearest power of 2 above n See: http://graphics.stanford.edu/~seander/bithacks.html#RoundUpPowerOf2Float

10.32.3.2 morpho_printvalue()

```
void morpho_printvalue (
             value v )
```

Prints a value.

Parameters

v	The value to print

10.32.3.3 morpho_strdup()

```
char* morpho_strdup (
            char * string )
```

Duplicates a string.

Parameters

string	String to duplicate

Warning

Caller should call MALLOC_FREE on the allocated string

10.33 utils/error.c File Reference

Morpho error handling.

```
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include "error.h"
#include "common.h"
#include "dictionary.h"
```

Functions

- void error_clear (error *err)
- void error_init (error *err)
- bool morpho_getdefinitionfromid (errorid id, errordefinition **def)
- void morpho_writeerrorwithidvalist (error *err, errorid id, int line, int posn, va_list args)

  Writes an error message to an error structure.
- void morpho_writeerrorwithid (error *err, errorid id, int line, int posn,...)

  Writes an error message to an error structure.
- void morpho_defineerror (errorid id, errorcategory cat, char *message)
- errorid morpho_geterrorid (error *err)
- void error_initialize (void)
- void error_finalize (void)

10.33.1 Detailed Description

Morpho error handling.

Morpho memory management.

Author

T J Atherton

10.33.2 Function Documentation

10.33.2.1 error_clear()

```
void error_clear (
            error * err )
```

Clears an error structure

Parameters

err	Error struct to fill out

10.33.2.2 error_finalize()

```
void error_finalize (
            void  )
```

Finalizes the error handling system

10.33.2.3 error_init()

```
void error_init (
            error * err )
```

Clears an error structure

Parameters

err	Error struct to fill out

10.33.2.4 error_initialize()

```
void error_initialize (
            void  )
```

Initializes the error handling system

10.33.2.5 morpho_defineerror()

```
void morpho_defineerror (
            errorid id,
            errorcategory cat,
            char * message )
```

Defines an error

Parameters

id	Error struct to fill out
cat	The category of error
message	The message string

10.33.2.6 morpho_getdefinitionfromid()

```
bool morpho_getdefinitionfromid (
            errorid id,
            errordefinition ** def )
```

Gets an error definition given an errorid

Parameters

in	*id*	Error to retrieve
out	*def*	The error definition

Returns

true on success

10.33.2.7 morpho_geterrorid()

```
errorid morpho_geterrorid (
            error * err )
```

Gets the id of an error

10.33.2.8 morpho_writeerrorwithid()

```
void morpho_writeerrorwithid (
             error * err,
             errorid id,
             int line,
             int posn,
              ...  )
```

Writes an error message to an error structure.

Parameters

err	The error structure
id	The error id.
line	The line at which the error occurred, if identifiable.
posn	The position in the line at which the error occurred, if identifiable.
...	Additional parameters (the data for the printf commands in the message)

10.33.2.9 morpho_writeerrorwithidvalist()

```
void morpho_writeerrorwithidvalist (
             error * err,
             errorid id,
             int line,
             int posn,
             va_list args )
```

Writes an error message to an error structure.

Parameters

err	The error structure
id	The error id.
line	The line at which the error occurred, if identifiable.
posn	The position in the line at which the error occurred, if identifiable.
args	Additional parameters (the data for the printf commands in the message)

10.34 utils/error.h File Reference

Morpho error handling.

```
#include <stdarg.h>
#include "build.h"
#include "varray.h"
```

Classes

- struct error

  A static container for error messages.
- struct errordefinition

  Definition of an error message.

Macros

- #define ERROR_SHOULDCONTINUE(cat) (cat < ERROR_HALT)
- #define ERROR_SUCCEEDED(err) ((err).cat == ERROR_NONE)
- #define ERROR_ISRUNTIMEERROR(err) ((err).cat <= ERROR_EXIT)
- #define ERROR_POSNUNIDENTIFIABLE -1
- #define UNREACHABLE(x)
- #define BSD_EX_SOFTWARE 70
- #define **ERROR_ALLOCATIONFAILED** "Alloc"
- #define **ERROR_ALLOCATIONFAILED_MSG** "%s allocation failed."
- #define **ERROR_INTERNALERROR** "Intrnl"
- #define **ERROR_INTERNALERROR_MSG** "Internal error (contact developer)."
- #define **COMPILE_UNTERMINATEDCOMMENT** "UntrmComm"
- #define **COMPILE_UNTERMINATEDCOMMENT_MSG** "Unterminated multiline comment '/∗'."
- #define **COMPILE_UNTERMINATEDSTRING** "UntrmStrng"
- #define **COMPILE_UNTERMINATEDSTRING_MSG** "Unterminated string."
- #define **COMPILE_INCOMPLETEEXPRESSION** "IncExp"
- #define **COMPILE_INCOMPLETEEXPRESSION_MSG** "Incomplete expression."
- #define **COMPILE_MISSINGPARENTHESIS** "MssngParen"
- #define **COMPILE_MISSINGPARENTHESIS_MSG** "Expect ')' after expression."
- #define **COMPILE_EXPECTEXPRESSION** "ExpExpr"
- #define **COMPILE_EXPECTEXPRESSION_MSG** "Expected expression."
- #define **COMPILE_MISSINGSEMICOLON** "MssngSemiVal"
- #define **COMPILE_MISSINGSEMICOLON_MSG** "Expect ; after value."
- #define **COMPILE_MISSINGSEMICOLONEXP** "MssngExpTerm"
- #define **COMPILE_MISSINGSEMICOLONEXP_MSG** "Expect expression terminator (; or newline) after expression."
- #define **COMPILE_MISSINGSEMICOLONVAR** "MssngSemiVar"
- #define **COMPILE_MISSINGSEMICOLONVAR_MSG** "Expect ; after variable declaration."
- #define **COMPILE_VAREXPECTED** "VarExpct"
- #define **COMPILE_VAREXPECTED_MSG** "Variable name expected after var."
- #define **COMPILE_BLOCKTERMINATOREXP** "MssngBrc"
- #define **COMPILE_BLOCKTERMINATOREXP_MSG** "Expected '}' to finish block."
- #define **COMPILE_IFLFTPARENMISSING** "IfMssngLftPrn"
- #define **COMPILE_IFLFTPARENMISSING_MSG** "Expected '(' after if."
- #define **COMPILE_IFRGHTPARENMISSING** "IfMssngRgtPrn"
- #define **COMPILE_IFRGHTPARENMISSING_MSG** "Expected ')' after condition."
- #define **COMPILE_WHILELFTPARENMISSING** "WhlMssngLftPrn"
- #define **COMPILE_WHILELFTPARENMISSING_MSG** "Expected '(' after while."
- #define **COMPILE_FORLFTPARENMISSING** "ForMssngLftPrn"
- #define **COMPILE_FORLFTPARENMISSING_MSG** "Expected '(' after for."
- #define **COMPILE_FORSEMICOLONMISSING** "ForMssngSemi"
- #define **COMPILE_FORSEMICOLONMISSING_MSG** "Expected ';'."
- #define **COMPILE_FORRGHTPARENMISSING** "ForMssngRgtPrn"
- #define **COMPILE_FORRGHTPARENMISSING_MSG** "Expected ')' after for clauses."
- #define **COMPILE_FNNAMEMISSING** "FnNoName"

- #define **COMPILE_FNNAMEMISSING_MSG** "Expected function or method name."
- #define **COMPILE_FNLEFTPARENMISSING** "FnMssngLftPrn"
- #define **COMPILE_FNLEFTPARENMISSING_MSG** "Expect '(' after name."
- #define **COMPILE_FNRGHTPARENMISSING** "FnMssngRgtPrn"
- #define **COMPILE_FNRGHTPARENMISSING_MSG** "Expect ')' after parameters."
- #define **COMPILE_FNLEFTCURLYMISSING** "FnMssngLftBrc"
- #define **COMPILE_FNLEFTCURLYMISSING_MSG** "Expect '{' before body."
- #define **COMPILE_CALLRGHTPARENMISSING** "CllMssngRgtPrn"
- #define **COMPILE_CALLRGHTPARENMISSING_MSG** "Expect ')' after arguments."
- #define **COMPILE_EXPECTCLASSNAME** "ClsNmMssng"
- #define **COMPILE_EXPECTCLASSNAME_MSG** "Expect class name."
- #define **COMPILE_CLASSLEFTCURLYMISSING** "ClsMssngLftBrc"
- #define **COMPILE_CLASSLEFTCURLYMISSING_MSG** "Expect '{' before class body."
- #define **COMPILE_CLASSRGHTCURLYMISSING** "ClsMssngRgtBrc"
- #define **COMPILE_CLASSRGHTCURLYMISSING_MSG** "Expect '}' after class body."
- #define **COMPILE_EXPECTDOTAFTERSUPER** "ExpctDtSpr"
- #define **COMPILE_EXPECTDOTAFTERSUPER_MSG** "Expect '.' after 'super'"
- #define **COMPILE_INCOMPLETESTRINGINT** "IntrpIncmp"
- #define **COMPILE_INCOMPLETESTRINGINT_MSG** "Incomplete string after interpolation."
- #define **COMPILE_VARBLANKINDEX** "EmptyIndx"
- #define **COMPILE_VARBLANKINDEX_MSG** "Empty capacity in variable declaration."
- #define **COMPILE_IMPORTMISSINGNAME** "ImprtMssngNm"
- #define **COMPILE_IMPORTMISSINGNAME_MSG** "Import expects a module or file name."
- #define **COMPILE_IMPORTUNEXPCTDTOK** "ImprtExpctFrAs"
- #define **COMPILE_IMPORTUNEXPCTDTOK_MSG** "Import only expects for or as after module or file name."
- #define **COMPILE_IMPORTASSYMBL** "ExpctSymblAftrAs"
- #define **COMPILE_IMPORTASSYMBL_MSG** "Expect symbol after as in import."
- #define **COMPILE_IMPORTFORSYMBL** "ExpctSymblAftrFr"
- #define **COMPILE_IMPORTFORSYMBL_MSG** "Expect symbol(s) after for in import."
- #define **PARSE_UNRECGNZEDTOK** "UnrcgnzdTok"
- #define **PARSE_UNRECGNZEDTOK_MSG** "Encountered an unrecognized token."
- #define **COMPILE_SYMBOLNOTDEFINED** "SymblUndf"
- #define **COMPILE_SYMBOLNOTDEFINED_MSG** "Symbol '%s' not defined."
- #define **COMPILE_TOOMANYCONSTANTS** "TooMnyCnst"
- #define **COMPILE_TOOMANYCONSTANTS_MSG** "Too many constants."
- #define **COMPILE_ARGSNOTSYMBOLS** "FnPrmSymb"
- #define **COMPILE_ARGSNOTSYMBOLS_MSG** "Function parameters must be symbols."
- #define **COMPILE_PROPERTYNAMERQD** "PptyNmRqd"
- #define **COMPILE_PROPERTYNAMERQD_MSG** "Property name required."
- #define **COMPILE_SELFOUTSIDECLASS** "SlfOtsdClss"
- #define **COMPILE_SELFOUTSIDECLASS_MSG** "Cannot use 'self' outside of a class."
- #define **COMPILE_RETURNININITIALIZER** "InitRtn"
- #define **COMPILE_RETURNININITIALIZER_MSG** "Cannot return a value in an initializer."
- #define **COMPILE_EXPECTSUPER** "SprNmMssng"
- #define **COMPILE_EXPECTSUPER_MSG** "Expect superclass name."
- #define **COMPILE_SUPERCLASSNOTFOUND** "SptNtFnd"
- #define **COMPILE_SUPERCLASSNOTFOUND_MSG** "Superclass '%s' not found."
- #define **COMPILE_SUPEROUTSIDECLASS** "SptOtsdClss"
- #define **COMPILE_SUPEROUTSIDECLASS_MSG** "Cannot use 'super' outside of a class."
- #define **COMPILE_NOSUPER** "SprSelMthd"
- #define **COMPILE_NOSUPER_MSG** "Can only use 'super' to select a method."
- #define **COMPILE_INVALIDASSIGNMENT** "InvldAssgn"
- #define **COMPILE_INVALIDASSIGNMENT_MSG** "Invalid assignment target."
- #define **COMPILE_CLASSINHERITSELF** "ClssCrcRf"
- #define **COMPILE_CLASSINHERITSELF_MSG** "A class cannot inherit from itself."

- #define **COMPILE_TOOMANYARGS** "TooMnyArg"
- #define **COMPILE_TOOMANYARGS_MSG** "Too many arguments."
- #define **COMPILE_TOOMANYPARAMS** "TooMnyPrm"
- #define **COMPILE_TOOMANYPARAMS_MSG** "Too many parameters."
- #define **COMPILE_ISOLATEDSUPER** "IsoSpr"
- #define **COMPILE_ISOLATEDSUPER_MSG** "Expect '.' after 'super'."
- #define **COMPILE_VARALREADYDECLARED** "VblDcl"
- #define **COMPILE_VARALREADYDECLARED_MSG** "Variable with this name already declared in this scope."
- #define **VM_NONNUMERICALOPERAND** "NonNumOp"
- #define **VM_NONNUMERICALOPERAND_MSG** "Operands must be numbers."
- #define **VM_CONCATENATIONFAILED** "Cnct"
- #define **VM_CONCATENATIONFAILED_MSG** "Concatenation failed."
- #define **VM_UNCALLABLE** "Uncallable"
- #define **VM_UNCALLABLE_MSG** "Can only call a function or method."
- #define **VM_GLOBALRETURN** "GlblRtrn"
- #define **VM_GLOBALRETURN_MSG** "Return encountered outside a function or method."
- #define **VM_INSTANTIATEFAILED** "InstFail"
- #define **VM_INSTANTIATEFAILED_MSG** "Could not instantiate object."
- #define **VM_NOTANOBJECT** "NotAnObj"
- #define **VM_NOTANOBJECT_MSG** "Not an object."
- #define **VM_OBJECTLACKSPROPERTY** "ObjLcksPrp"
- #define **VM_OBJECTLACKSPROPERTY_MSG** "Object lacks property or method '%s'."
- #define **VM_NOINITIALIZER** "NoInit"
- #define **VM_NOINITIALIZER_MSG** "Cannot instantiate with arguments because class '%s' does not provide an initializer."
- #define **VM_NOTANINSTANCE** "NotAnInst"
- #define **VM_NOTANINSTANCE_MSG** "Can only invoke methods on objects."
- #define **VM_CLASSLACKSPROPERTY** "ClssLcksMthd"
- #define **VM_CLASSLACKSPROPERTY_MSG** "Class lacks method '%s'."
- #define **VM_INVALIDARGS** "InvldArgs"
- #define **VM_INVALIDARGS_MSG** "Expected %u arguments but got %u."
- #define **VM_INVALIDOPERANDS** "InvldOp"
- #define **VM_INVALIDOPERANDS_MSG** "Cannot add these operands."
- #define **VM_NOTINDEXABLE** "NotIndxbl"
- #define **VM_NOTINDEXABLE_MSG** "Value or object not indexable."
- #define **VM_OUTOFBOUNDS** "ArrayBnds"
- #define **VM_OUTOFBOUNDS_MSG** "Array index out of bounds."
- #define **VM_NONNUMINDX** "NonNmIndx"
- #define **VM_NONNUMINDX_MSG** "Non-numerical array index."
- #define **VM_ARRAYWRONGDIM** "ArrayDim"
- #define **VM_ARRAYWRONGDIM_MSG** "Incorrect number of dimensions for array."

Typedefs

- typedef char * errorid

  Identifier for errors.
- typedef errorcategory morphoerror

  A type used by public-facing morpho functions.

Enumerations

- enum errorcategory {
  ERROR_NONE, ERROR_INFO, ERROR_WARNING, ERROR_HALT,
  ERROR_EXIT, ERROR_LEX, ERROR_PARSE, ERROR_COMPILE }

Functions

- void error_init (error *err)
- void error_clear (error *err)
- void morpho_writeerrorwithid (error *err, errorid id, int line, int posn,...)

  Writes an error message to an error structure.
- void morpho_writeerrorwithidvalist (error *err, errorid id, int line, int posn, va_list args)

  Writes an error message to an error structure.
- void morpho_defineerror (errorid id, errorcategory cat, char *message)
- errorid morpho_geterrorid (error *err)
- void error_initialize (void)
- void error_finalize (void)

10.34.1 Detailed Description

Morpho error handling.

Author

T J Atherton

10.34.2 Macro Definition Documentation

10.34.2.1 BSD_EX_SOFTWARE

```
#define BSD_EX_SOFTWARE 70
```

Exit codes from BSD standard

10.34.2.2 ERROR_ISRUNTIMEERROR

```
#define ERROR_ISRUNTIMEERROR(
              err ) ((err).cat <= ERROR_EXIT)
```

Is this a runtime error?

10.34.2.3 ERROR_POSNUNIDENTIFIABLE

```
#define ERROR_POSNUNIDENTIFIABLE -1
```

Set line and posn to this value if they can't be determined

10.34.2.4 ERROR_SHOULDCONTINUE

```
#define ERROR_SHOULDCONTINUE(
              cat ) (cat < ERROR_HALT)
```

Checks if execution should continue after the error

10.34.2.5 ERROR_SUCCEEDED

```
#define ERROR_SUCCEEDED(
              err ) ((err).cat == ERROR_NONE)
```

Did an operation succeed without errors?

10.34.2.6 UNREACHABLE

```
#define UNREACHABLE(
              x )
```

A varray of errordefinitions Macro to place in code that should be unreachable

10.34.3 Enumeration Type Documentation

10.34.3.1 errorcategory

```
enum errorcategory
```

Identifies the category of error that has occurred

Enumerator

ERROR_INFO	No error.
ERROR_WARNING	Informational messages generated.
ERROR_HALT	Warnings generated.
ERROR_EXIT	or has occured. Should return to the user as fast as possible.
ERROR_LEX	Unrecoverable error has occured; Morpho will exit quickly.
ERROR_PARSE	or generated by the lexer
ERROR_COMPILE	or generated by the parser or generated by the compiler

10.34.4 Function Documentation

10.34.4.1 error_clear()

```
void error_clear (
            error * err )
```

Clears an error structure

Parameters

err	Error struct to fill out

10.34.4.2 error_finalize()

```
void error_finalize (
            void  )
```

Finalizes the error handling system

10.34.4.3 error_init()

```
void error_init (
            error * err )
```

Clears an error structure

Parameters

err	Error struct to fill out

10.34.4.4 error_initialize()

```
void error_initialize (
            void  )
```

Initializes the error handling system

10.34.4.5 morpho_defineerror()

```
void morpho_defineerror (
            errorid id,
            errorcategory cat,
            char * message )
```

Defines an error

Parameters

id	Error struct to fill out
cat	The category of error
message	The message string

10.34.4.6 morpho_geterrorid()

```
errorid morpho_geterrorid (
            error * err )
```

Gets the id of an error

10.34.4.7 morpho_writeerrorwithid()

```
void morpho_writeerrorwithid (
            error * err,
            errorid id,
            int line,
            int posn,
             ...  )
```

Writes an error message to an error structure.

Parameters

err	The error structure
id	The error id.
line	The line at which the error occurred, if identifiable.
posn	The position in the line at which the error occurred, if identifiable.
...	Additional parameters (the data for the printf commands in the message)

10.34.4.8 morpho_writeerrorwithidvalist()

```
void morpho_writeerrorwithidvalist (
            error * err,
            errorid id,
            int line,
            int posn,
            va_list args )
```

Writes an error message to an error structure.

Parameters

err	The error structure
id	The error id.
line	The line at which the error occurred, if identifiable.
posn	The position in the line at which the error occurred, if identifiable.
args	Additional parameters (the data for the printf commands in the message)

10.35 utils/memory.h File Reference

Morpho memory management.

```
#include <stdlib.h>
#include "build.h"
```

Macros

- #define MORPHO_MALLOC(size) morpho_allocate(NULL, 0, size)
- #define MORPHO_FREE(x) morpho_allocate(x, 0, 0)
- #define MORPHO_REALLOC(x, size) morpho_allocate(x, 0, size)

Functions

- void ∗ morpho_allocate (void ∗old, size_t oldsize, size_t newsize)

  Generic allocator function.

10.35.1 Detailed Description

Morpho memory management.

Author

T J Atherton

10.35.2 Macro Definition Documentation

10.35.2.1 MORPHO_FREE

```
#define MORPHO_FREE(
              x ) morpho_allocate(x, 0, 0)
```

Macro to redirect free through our memory management

10.35.2.2 MORPHO_MALLOC

```
#define MORPHO_MALLOC(
              size ) morpho_allocate(NULL, 0, size)
```

Macro to redirect malloc through our memory management

10.35.2.3 MORPHO_REALLOC

```
#define MORPHO_REALLOC(
              x,
              size ) morpho_allocate(x, 0, size)
```

Macro to redirect realloc through our memory management

10.35.3 Function Documentation

10.35.3.1 morpho_allocate()

```
void* morpho_allocate (
             void * old,
             size_t oldsize,
             size_t newsize )
```

Generic allocator function.

Parameters

old	A previously allocated pointer, or NULL to allocate new memory
oldsize	The previously allocated size
newsize	New size to allocate

Returns

A pointer to allocated memory, or NULL on failure.

10.36 utils/parse.c File Reference

Lexer and parser.

```
#include <string.h>
#include "parse.h"
#include "object.h"
#include "common.h"
```

Macros

- #define UNUSED { NULL, NULL, PREC_NONE }

  Parse table. Each line in the table defines the parserule(s) for a specific token type.
- #define **PREFIX**(fn) { fn, NULL, PREC_NONE }
- #define **INFIX**(fn, prec) { NULL, fn, prec }
- #define **MIXFIX**(unaryfn, infixfn, prec) { unaryfn, infixfn, prec }

Functions

- void lex_init (lexer ∗l, const char ∗start, int line)

  Initializes a lexer with a given starting point.
- void lex_recordtoken (lexer ∗l, tokentype type, token ∗tok)

  Records a token.
- tokentype lex_symboltype (lexer ∗l)

  Determines if a token matches any of the reserved words.
- bool lex (lexer ∗l, token ∗tok, error ∗err)

  Identifies the next token.
- void parse_init (parser ∗p, lexer ∗lex, error ∗err, syntaxtree ∗tree)

  Initialize a parser.
- void parse_synchronize (parser ∗p)
- syntaxtreeindx syntaxtree_addnode (syntaxtree ∗tree, syntaxtreenodetype type, value content, int line, int posn, syntaxtreeindx left, syntaxtreeindx right)

  Adds a node to the syntax tree.
- bool parse (parser ∗p)
- bool parse_stringtovaluearray (char ∗string, unsigned int nmax, value ∗v, unsigned int ∗n, error ∗err)

Variables

- parserule **rules** [ ]

10.36.1 Detailed Description

Lexer and parser.

Author

T J Atherton

10.36.2 Macro Definition Documentation

10.36.2.1 UNUSED

```
#define UNUSED { NULL, NULL, PREC_NONE }
```

Parse table. Each line in the table defines the parserule(s) for a specific token type.

Warning

It is imperative that this table be in the same order as the tokentype enum

10.36.3 Function Documentation

10.36.3.1 lex()

```
bool lex (
            lexer * l,
            token * tok,
            error * err )
```

Identifies the next token.

Parameters

in	*l*	The lexer in use
out	*err*	An error block to fill out on an error
out	*tok*	Token structure to fill out

Returns

true on success or false on failure

10.36.3.2 lex_init()

```
void lex_init (
            lexer * l,
            const char * start,
            int line )
```

Initializes a lexer with a given starting point.

Parameters

l	The lexer to initialize
start	Starting point to lex from
line	The current line number

10.36.3.3 lex_recordtoken()

```
void lex_recordtoken (
            lexer * l,
```

```
            tokentype type,
            token * tok )
```

Records a token.

Parameters

in	*l*	The lexer in use
in	*type*	Type of token to record
out	*tok*	Token structure to fill out

10.36.3.4 parse()

```
bool parse (
            parser * p )
```

Entry point into the parser

10.36.3.5 parse_init()

```
void parse_init (
            parser * p,
            lexer * lex,
            error * err,
            syntaxtree * tree )
```

Initialize a parser.

Parameters

p	the parser to initialize
lex	lexer to use
err	an error structure to fill out if necessary
tree	syntaxtree to fill out

10.36.3.6 parse_stringtovaluearray()

```
bool parse_stringtovaluearray (
            char * string,
            unsigned int nmax,
            value * v,
            unsigned int * n,
            error * err )
```

Convenience function to parse a string into an array of values

Parameters

in	*string*	- string to parse
in	*nmax*	- maximum number of values to read

Parameters

in	*v*	- value array, filled out on return
out	*n*	- number of values read
out	*err*	- error structure filled out if an error occurs

Returns

true if successful, false otherwise.

10.36.3.7 parse_synchronize()

```
void parse_synchronize (
            parser * p )
```

Keep parsing til the end of a statement boundary. Align

10.36.3.8 syntaxtree_addnode()

```
syntaxtreeindx syntaxtree_addnode (
            syntaxtree * tree,
            syntaxtreenodetype type,
            value content,
            int line,
            int posn,
            syntaxtreeindx left,
            syntaxtreeindx right )
```

Adds a node to the syntax tree.

Parameters

tree	tree to add to.
type	type of node to add
content	a value to add
left	} left ...
right	} ...and right branches of the node.

10.37 utils/parse.h File Reference

Lexer and parser.

```
#include <stdio.h>
#include "error.h"
#include "syntaxtree.h"
```

Classes

- struct token
- struct lexer

  Store the current configuration of a lexer.
- struct parser

  A structure that defines the state of a parser.
- struct parserule

  A parse rule will be defined for each token, providing functions to parse the token if it is encountered in the prefix or infix positions. The parse rule also defines the precedence.

Macros

- #define TOKEN_BLANK ((token) {.type=TOKEN_NONE, .start=NULL, .length=0, .line=0, .posn=0})

Typedefs

- typedef syntaxtreeindx(∗ parsefunction) (parser ∗c)

  Definition of a parse function.

Enumerations

- enum tokentype {
  TOKEN_NONE, **TOKEN_NEWLINE**, **TOKEN_HELP**, **TOKEN_STRING**,
  TOKEN_INTERPOLATION, **TOKEN_INTEGER**, **TOKEN_NUMBER**, **TOKEN_SYMBOL**,
  TOKEN_TRUE, **TOKEN_FALSE**, **TOKEN_NIL**, **TOKEN_SELF**,
  TOKEN_SUPER, **TOKEN_LEFTPAREN**, **TOKEN_RIGHTPAREN**, **TOKEN_LEFTSQBRACKET**,
  TOKEN_RIGHTSQBRACKET, **TOKEN_LEFTCURLYBRACKET**, **TOKEN_RIGHTCURLYBRACKET**, **TO↩KEN_COLON**,
  TOKEN_SEMICOLON, **TOKEN_COMMA**, **TOKEN_PLUS**, **TOKEN_MINUS**,
  TOKEN_STAR, **TOKEN_SLASH**, **TOKEN_CIRCUMFLEX**, **TOKEN_PLUSPLUS**,
  TOKEN_MINUSMINUS, **TOKEN_PLUSEQ**, **TOKEN_MINUSEQ**, **TOKEN_STAREQ**,
  TOKEN_SLASHEQ, **TOKEN_DOT**, **TOKEN_DOTDOT**, **TOKEN_EXCLAMATION**,
  TOKEN_AMP, **TOKEN_VBAR**, **TOKEN_DBLAMP**, **TOKEN_DBLVBAR**,
  TOKEN_EQUAL, **TOKEN_EQ**, **TOKEN_NEQ**, **TOKEN_LT**,
  TOKEN_GT, **TOKEN_LTEQ**, **TOKEN_GTEQ**, **TOKEN_PRINT**,
  TOKEN_VAR, **TOKEN_IF**, **TOKEN_ELSE**, **TOKEN_IN**,
  TOKEN_WHILE, **TOKEN_FOR**, **TOKEN_FUNCTION**, **TOKEN_RETURN**,
  TOKEN_CLASS, **TOKEN_IMPORT**, **TOKEN_AS**, **TOKEN_INCOMPLETE**,
  TOKEN_ERROR, **TOKEN_EOF** }
- enum precedence {
  PREC_NONE, **PREC_LOWEST**, **PREC_ASSIGN**, **PREC_OR**,
  PREC_AND, **PREC_EQUALITY**, **PREC_COMPARISON**, **PREC_RANGE**,
  PREC_TERM, **PREC_FACTOR**, **PREC_POW**, **PREC_UNARY**,
  PREC_CALL, **PREC_HIGHEST** }

  an enumerated type that defines precedence order in Morpho.

Functions

- void lex_init (lexer ∗l, const char ∗start, int line)

  Initializes a lexer with a given starting point.
- bool lex (lexer ∗l, token ∗tok, error ∗err)

  Identifies the next token.
- void parse_init (parser ∗p, lexer ∗lex, error ∗err, syntaxtree ∗tree)

  Initialize a parser.
- bool parse (parser ∗p)
- bool parse_stringtovaluearray (char ∗string, unsigned int nmax, value ∗v, unsigned int ∗n, error ∗err)

10.37.1 Detailed Description

Lexer and parser.

Author

T J Atherton and others (see below)

10.37.2 Macro Definition Documentation

10.37.2.1 TOKEN_BLANK

```
#define TOKEN_BLANK ((token) {.type=TOKEN_NONE, .start=NULL, .length=0, .line=0, .posn=0} )
```

Literal for a blank token

10.37.3 Enumeration Type Documentation

10.37.3.1 tokentype

```
enum tokentype
```

The type of token. N.B. Each token will have a parserule in the parser

10.37.4 Function Documentation

10.37.4.1 lex()

```
bool lex (
            lexer * l,
            token * tok,
            error * err )
```

Identifies the next token.

Parameters

in	*l*	The lexer in use
out	*err*	An error block to fill out on an error
out	*tok*	Token structure to fill out

Returns

true on success or false on failure

10.37.4.2 lex_init()

```
void lex_init (
            lexer * l,
            const char * start,
            int line )
```

Initializes a lexer with a given starting point.

Parameters

l	The lexer to initialize
start	Starting point to lex from
line	The current line number

10.37.4.3 parse()

```
bool parse (
            parser * p )
```

Entry point into the parser

10.37.4.4 parse_init()

```
void parse_init (
            parser * p,
            lexer * lex,
            error * err,
            syntaxtree * tree )
```

Initialize a parser.

Parameters

p	the parser to initialize
lex	lexer to use
err	an error structure to fill out if necessary
tree	syntaxtree to fill out

10.37.4.5 parse_stringtovaluearray()

```
bool parse_stringtovaluearray (
            char * string,
            unsigned int nmax,
            value * v,
            unsigned int * n,
            error * err )
```

Convenience function to parse a string into an array of values

Parameters

in	*string*	- string to parse
in	*nmax*	- maximum number of values to read
in	*v*	- value array, filled out on return
out	*n*	- number of values read
out	*err*	- error structure filled out if an error occurs

Returns

true if successful, false otherwise.

10.38 utils/random.c File Reference

Random number generation.

```
#include "random.h"
```

Functions

- void splitmix64_seed (uint64_t seed)
- void **xoshiro256pp_jump** (void)
- void **xoshiro256pp_longjump** (void)
- void **xoshiro256p_jump** (void)
- void **xoshiro256p_longjump** (void)
- double random_double (void)
- unsigned int random_int (void)
- void random_initialize (void)

Variables

- uint64_t **splitmix64_state**

10.38.1 Detailed Description

Random number generation.

Author

T J Atherton and others (see below)

10.38.2 Function Documentation

10.38.2.1 random_double()

```
double random_double (
            void  )
```

Generate a random double on the interval [0.0,1.0]

10.38.2.2 random_initialize()

```
void random_initialize (
            void  )
```

Initialize the random number generator

10.38.2.3 random_int()

```
unsigned int random_int (
            void  )
```

Generate a random 32 bit unsigned int

10.38.2.4 splitmix64_seed()

```
void splitmix64_seed (
            uint64_t seed )
```

Set the seed for splitmix64

10.39 vm/compile.c File Reference

Compiles raw input to Morpho instructions.

```
#include <stdarg.h>
#include <string.h>
#include "compile.h"
#include "error.h"
#include "vm.h"
#include "morpho.h"
#include "file.h"
```

Macros

- #define **NODE_NORULE** NULL
- #define **NODE_UNDEFINED** { NODE_NORULE }

Functions

- void compiler_adddebuginfo (compiler *c, syntaxtreenode *node)

  Adds source info to the program Debugging info is stored in a simple list with run length encoding, i.e. Each entry corresponds to ninstr instructions.
- **DEFINE_VARRAY** (registeralloc, registeralloc)
- void compiler_fstackinit (compiler *c)
- void compiler_fstackclear (compiler *c)
- void compiler_beginscope (compiler *c)
- void compiler_endscope (compiler *c)
- unsigned int compiler_currentscope (compiler *c)
- registerindx compiler_addupvalue (functionstate *f, bool islocal, indx ix)
- indx compiler_closure (compiler *c, syntaxtreenode *node, registerindx reg)
- codeinfo **compiler_movefromregister** (compiler *c, syntaxtreenode *node, codeinfo dest, registerindx reg)
- bool **compiler_listentries** (compiler *c, syntaxtreenode *node)
- objectclass * compiler_findclass (objectfunction *f, value name)
- void compiler_stripend (compiler *c)
- void compiler_copyglobals (compiler *src, compiler *dest)
- void compiler_init (const char *source, program *out, compiler *c)

  Initialize a compiler.
- void compiler_clear (compiler *c)

  Clear attached data structures from a compiler.
- bool morpho_compile (char *in, compiler *c, error *err)
- compiler * morpho_newcompiler (program *out)
- void morpho_freecompiler (compiler *c)
- void compile_initialize (void)
- void compile_finalize (void)

Variables

- compilenoderule noderules [ ]

10.39.1 Detailed Description

Compiles raw input to Morpho instructions.

Author

T J Atherton

10.39.2 Function Documentation

10.39.2.1 compile_finalize()

```
void compile_finalize (
             void  )
```

Finalizes the compiler

10.39.2.2 compile_initialize()

```
void compile_initialize (
             void  )
```

Initializes the error handling system

10.39.2.3 compiler_addupvalue()

```
registerindx compiler_addupvalue (
             functionstate * f,
             bool islocal,
             indx ix )
```

Adds an upvalue to a functionstate

10.39.2.4 compiler_beginscope()

```
void compiler_beginscope (
             compiler * c )
```

Increments the scope counter in the current functionstate

10.39.2.5 compiler_clear()

```
void compiler_clear (
             compiler * c )
```

Clear attached data structures from a compiler.

Parameters

in	*c*	compiler to clear

10.39.2.6 compiler_closure()

```
indx compiler_closure (
            compiler * c,
            syntaxtreenode * node,
            registerindx reg )
```

Creates code to generate a closure for the current environment.

10.39.2.7 compiler_copyglobals()

```
void compiler_copyglobals (
            compiler * src,
            compiler * dest )
```

Copies the globals across from one compiler to another

10.39.2.8 compiler_currentscope()

```
unsigned int compiler_currentscope (
            compiler * c )
```

Decrements the scope counter in the current functionstate

10.39.2.9 compiler_endscope()

```
void compiler_endscope (
            compiler * c )
```

Decrements the scope counter in the current functionstate

10.39.2.10 compiler_findclass()

```
objectclass* compiler_findclass (
            objectfunction * f,
            value name )
```

Finds a class in the constant table of a function

10.39.2.11 compiler_fstackclear()

```
void compiler_fstackclear (
            compiler * c )
```

Clears the function stack

10.39.2.12 compiler_fstackinit()

```
void compiler_fstackinit (
            compiler * c )
```

Initializes the function stack

10.39.2.13 compiler_init()

```
void compiler_init (
            const char * source,
            program * out,
            compiler * c )
```

Initialize a compiler.

Parameters

in	*source*	source code to compile
in	*out*	destination program to compile to
out	*c*	compiler structure is filled out

10.39.2.14 compiler_stripend()

```
void compiler_stripend (
            compiler * c )
```

Strip an 'end' instruction at the end of the program

10.39.2.15 morpho_compile()

```
bool morpho_compile (
            char * in,
            compiler * c,
            error * err )
```

Interface to the compiler

Parameters

in	*in*	A string to compile
in	*c*	The compiler
out	*err*	Pointer to error block on failure

Returns

A bool indicating success or failure

10.39.2.16 morpho_freecompiler()

```
void morpho_freecompiler (
            compiler * c )
```

Frees a compiler

10.39.2.17 morpho_newcompiler()

```
compiler* morpho_newcompiler (
            program * out )
```

Creates a new compiler

10.39.3 Variable Documentation

10.39.3.1 noderules

```
compilenoderule noderules[]
```

Compiler definition table. This specifies a compiler function that handles each node type

10.40 vm/compile.h File Reference

Compiles raw input to Morpho instructions.

```
#include "core.h"
#include "syntaxtree.h"
#include "parse.h"
```

Classes

- struct registeralloc
- struct functionstate
- struct scompilerlist
- struct scompiler

  A structure that stores the state of a compiler.
- struct codeinfo
- struct compilenoderule

  A compilenoderule rule will be defined for each syntax tree node type, providing a function to compile the node.

Macros

- #define **MORPHO_CORE**
- #define GLOBAL_UNALLOCATED -1

  Value to indicate a global has not been allocated.
- #define REGISTER_UNALLOCATED -1

  Value to indicate a register has not been allocated.
- #define **FUNCTIONTYPE_ISMETHOD**(f) (f==FUNCTION ? false : true)
- #define **FUNCTIONTYPE_ISINITIALIZER**(f) (f==INITIALIZER)
- #define **CODEINFO_ISREGISTER**(info) (info.returntype==REGISTER)
- #define **CODEINFO_ISCONSTANT**(info) (info.returntype==CONSTANT)
- #define **CODEINFO_ISUPVALUE**(info) (info.returntype==UPVALUE)
- #define **CODEINFO_ISGLOBAL**(info) (info.returntype==GLOBAL)
- #define **CODEINFO**(c, d, n) ((codeinfo) { .returntype=(c), .dest=(d), .ninstructions=(n)})
- #define **CODEINFO_EMPTY** CODEINFO(REGISTER, REGISTER_UNALLOCATED, 0)

Typedefs

- typedef int globalindx

  Index of a global.
- typedef int registerindx

  Index of a register.
- typedef struct scompilerlist compilerlist
- typedef struct scompiler compiler

  A structure that stores the state of a compiler.
- typedef codeinfo(∗ compiler_nodefn) (compiler ∗c, syntaxtreenode ∗node, registerindx reg)

Enumerations

- enum functiontype { **FUNCTION**, **METHOD**, **INITIALIZER** }

Functions

- void compiler_init (const char ∗source, program ∗out, compiler ∗c)

  Initialize a compiler.
- void compiler_clear (compiler ∗c)

  Clear attached data structures from a compiler.

10.40.1 Detailed Description

Compiles raw input to Morpho instructions.

Author

T J Atherton

10.40.2 Typedef Documentation

10.40.2.1 compiler_nodefn

```
typedef codeinfo(* compiler_nodefn) (compiler *c, syntaxtreenode *node, registerindx reg)
```

A compiler_nodefn takes a syntax tree node and compiles it to bytecode

10.40.2.2 compilerlist

```
typedef struct scompilerlist compilerlist
```

This structure holds a list as it is being created

10.40.3 Enumeration Type Documentation

10.40.3.1 functiontype

```
enum functiontype
```

The type of the current function

10.40.4 Function Documentation

10.40.4.1 compiler_clear()

```
void compiler_clear (
            compiler * c )
```

Clear attached data structures from a compiler.

Parameters

in	*c*	compiler to clear

10.40.4.2 compiler_init()

```
void compiler_init (
            const char * source,
            program * out,
            compiler * c )
```

Initialize a compiler.

Parameters

in	*source*	source code to compile
in	*out*	destination program to compile to
out	*c*	compiler structure is filled out

10.41 vm/core.h File Reference

Data types for core Morpho components.

```
#include <stdio.h>
#include <stddef.h>
#include "error.h"
#include "random.h"
#include "varray.h"
#include "value.h"
#include "object.h"
#include "common.h"
#include "dictionary.h"
#include "builtin.h"
#include "opcodes.h"
```

Classes

- struct callframe
- struct debuginfo
- struct sprogram

  Morpho code program and associated data.
- struct graylist
- struct svm

  A Morpho virtual machine and its current state.

Macros

- #define ENCODE(op, A, B, C) (((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18) | ((B & 0xff) << 9) | (C & 0xff))
- #define ENCODEC(op, A, Bc, B, Cc, C) (((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18) | ((Bc & 0x1) << 17) | ((B & 0xff) << 9) | ((Cc & 0x1) << 8) | (C & 0xff))
- #define ENCODE_LONG(op, A, Bx) (((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18) | (Bx & 0xffff))
- #define ENCODE_LONGFLAGS(op, A, f, g, Bx) (((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18) | ((f & 0x1) << 17) | ((g & 0x1) << 16) | (Bx & 0xffff))
- #define ENCODE_SINGLE(op, A) (((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18))
- #define ENCODE_DOUBLE(op, A, Bc, B) (((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18) | ((Bc & 0x1) << 17) | ((B & 0xff) << 9))
- #define ENCODE_BYTE(op) ((((unsigned) op) & 0xff) << 26)
- #define ENCODE_EMPTYOPERAND 0
- #define DECODE_OP(x) (x >> 26)
- #define DECODE_A(x) ((x>>18) & (0xff))
- #define DECODE_B(x) ((x>>9) & (0xff))
- #define DECODE_C(x) (x & 0xff)
- #define DECODE_ISBCONSTANT(x) ((x>>17) & (0x1))
- #define DECODE_ISCCONSTANT(x) ((x>>8) & (0x1))
- #define DECODE_Bx(x) (x & 0xffff)
- #define DECODE_sBx(x) ((short) (x & 0xffff))
- #define DECODE_F(x) ((bool) ((x>>17) & (0x1)))
- #define **DECODE_G**(x) ((bool) ((x>>16) & (0x1)))
- #define **OPCODE**(name) OP_##name,
- #define MORPHO_MAXREGISTERS 255

  Maximum number of registers per call frame.
- #define VM_MAXIMUMREGISTERNUMBER 255

  Highest register addressable in a window.

Typedefs

- typedef struct sprogram **program**
- typedef struct svm **vm**
- typedef unsigned int instruction

  A Morpho instruction.
- typedef indx instructionindx

  Index into instructions.

Enumerations

- enum **opcode**

Functions

- **DECLARE_VARRAY** (instruction, instruction)
- void compile_initialize (void)
- void compile_finalize (void)

10.41.1 Detailed Description

Data types for core Morpho components.

Author

T J Atherton

10.41.2 Macro Definition Documentation

10.41.2.1 DECODE_A

```
#define DECODE_A(
              x ) ((x>>18) & (0xff))
```

Decode operand A

10.41.2.2 DECODE_B

```
#define DECODE_B(
              x ) ((x>>9) & (0xff))
```

Decode operand B

10.41.2.3 DECODE_Bx

```
#define DECODE_Bx(
              x ) (x & 0xffff)
```

Decode long operand Bx

10.41.2.4 DECODE_C

```
#define DECODE_C(
              x ) (x & 0xff)
```

Decode operand C

10.41.2.5 DECODE_F

```
#define DECODE_F(
              x ) ((bool) ((x>>17) & (0x1)))
```

Decode flags F and G

10.41.2.6 DECODE_ISBCONSTANT

```
#define DECODE_ISBCONSTANT(
              x ) ((x>>17) & (0x1))
```

Is Operand B a constant or a register?

10.41.2.7 DECODE_ISCCONSTANT

```
#define DECODE_ISCCONSTANT(
              x ) ((x>>8) & (0x1))
```

Is Operand C a constant or a register?

10.41.2.8 DECODE_OP

```
#define DECODE_OP(
              x ) (x >> 26)
```

Decode the opcode

10.41.2.9 DECODE_sBx

```
#define DECODE_sBx(
              x ) ((short) (x & 0xffff))
```

Decode signed long operand Bx

10.41.2.10 ENCODE

```
#define ENCODE(
              op,
              A,
              B,
              C ) ( ((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18) | ((B & 0xff) << 9)
| (C & 0xff) )
```

Encodes an instruction with operands A, B and C.

10.41.2.11 ENCODE_BYTE

```
#define ENCODE_BYTE(
              op ) ((((unsigned) op) & 0xff) << 26)
```

Encodes an instruction with no operands

10.41.2.12 ENCODE_DOUBLE

```
#define ENCODE_DOUBLE(
              op,
              A,
              Bc,
              B ) ( ((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18) | ((Bc & 0x1) <<
17) | ((B & 0xff) << 9) )
```

Encodes an instruction with operands A and B

10.41.2.13 ENCODE_EMPTYOPERAND

```
#define ENCODE_EMPTYOPERAND 0
```

Encodes an empty operand

10.41.2.14 ENCODE_LONG

```
#define ENCODE_LONG(
              op,
              A,
              Bx ) ( ((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18) | (Bx & 0xffff) )
```

Encodes an instruction with operand A and long operand Bx

10.41.2.15 ENCODE_LONGFLAGS

```
#define ENCODE_LONGFLAGS(
              op,
              A,
              f,
              g,
              Bx ) ( ((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18) | ((f & 0x1) <<
17) | ((g & 0x1) << 16) | (Bx & 0xffff) )
```

Encodes an instruction with operand A and long operand Bx

10.41.2.16 ENCODE_SINGLE

```
#define ENCODE_SINGLE(
              op,
              A ) ( ((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18))
```

Encodes an instruction with operand A

10.41.2.17 ENCODEC

```
#define ENCODEC(
              op,
              A,
              Bc,
              B,
              Cc,
              C ) ( ((((unsigned) op) & 0xff) << 26) | ((A & 0xff) << 18) | ((Bc & 0x1) <<
17) | ((B & 0xff) << 9) | ((Cc & 0x1) << 8) | (C & 0xff) )
```

Encodes an instruction with operands A, B and C also setting the constant flags Bc and Cc

10.41.3 Typedef Documentation

10.41.3.1 instruction

```
typedef unsigned int instruction
```

A Morpho instruction.

Each instruction fits into a 32 bit unsigned int with the following arrangement

```
        24      16      8       0
 .......|.......|.......|.......|
 **op**                          Opcode (gives 64 instructions)
       ***A****                  Operand A
               *       *         Select Constant (set) or Register for operands B & C
                ***B**** ***C**** } Operands B & C
               fg**Bx************ } Operand Bx with two boolean flags f & g
               fg*sBx************ } Signed operand Bx with two boolean flags f & g
```

Morpho instructions

10.41.4 Function Documentation

10.41.4.1 compile_finalize()

```
void compile_finalize (
            void  )
```

Finalizes the compiler

10.41.4.2 compile_initialize()

```
void compile_initialize (
            void  )
```

Initializes the error handling system

10.42 vm/opcodes.h File Reference

Morpho opcodes.

10.42.1 Detailed Description

Morpho opcodes.

Author

T J Atherton

10.43 vm/vm.c File Reference

Morpho virtual machine.

```
#include <stdarg.h>
#include <time.h>
#include "vm.h"
#include "compile.h"
#include "morpho.h"
#include "object.h"
```

Macros

- #define DISASSEMBLE_SHOWA 1

  Disassembles a single ABC format instruction.
- #define **DISASSEMBLE_SHOWB** 2
- #define **DISASSEMBLE_SHOWC** 4
- #define **DISASSEMBLE_LITC** 8
- #define DISASSEMBLE_SHOWA 1

  Disassembles a single ABC format instruction.
- #define **DISASSEMBLE_SHOWBx** 2
- #define **DISASSEMBLE_SHOWsBx** 4
- #define **DISASSEMBLE_SHOWF** 8
- #define **DISASSEMBLE_SHOWG** 16
- #define **DISASSEMBLE_OP**(op) printf("%s", op)
- #define **DISASSEMBLE_OPA**(op, i) disassemble_ABx(op, i, DISASSEMBLE_SHOWA, reg)
- #define **DISASSEMBLE_OPAB**(op, i)
- #define **DISASSEMBLE_OPAcB**(op, i)

- #define **DISASSEMBLE_OPABC**(op, i)
- #define **DISASSEMBLE_OPB**(op, i)
- #define **DISASSEMBLE_OPABx**(op, i) disassemble_ABx(op, i, DISASSEMBLE_SHOWA | DISASSEMB↩ LE_SHOWBx, reg)
- #define **DISASSEMBLE_OPBx**(op, i) disassemble_ABx(op, i, DISASSEMBLE_SHOWBx, reg)
- #define **DISASSEMBLE_OPAsBx**(op, i) disassemble_ABx(op, i, DISASSEMBLE_SHOWA | DISASSEM↩ BLE_SHOWBx, reg)
- #define **DISASSEMBLE_OPsBx**(op, i) disassemble_ABx(op, i, DISASSEMBLE_SHOWsBx, reg)
- #define **MORPHO_DISASSEMBLE_INSRUCTION**(bc, pc, k, r) ;
- #define **OPCODECNT**(p)
- #define **OPOPCODECNT**(p, bc)
- #define **INTERPRET_LOOP**
- #define **CASE_CODE**(name) case OP_##name
- #define **DISPATCH**() goto loop;
- #define **ERROR**(id) { vm_runtimeerror(v, pc-instructions, id); goto vm_error; }
- #define **VERROR**(id, ...) { vm_runtimeerror(v, pc-instructions, id, __VA_ARGS__); goto vm_error; }
- #define **ERRORCHK**() if (v->err.cat!=ERROR_NONE) goto vm_error;
- #define **CHECKCMPTYPE**(l, r)

Functions

- **DEFINE_VARRAY** (instruction, instruction)
- **DEFINE_VARRAY** (debuginfo, debuginfo)
- program ∗ morpho_newprogram (void)

  Creates and initializes a new program.
- void morpho_freeprogram (program ∗p)

  Frees a program.
- void program_setentry (program ∗p, instructionindx entry)
- instructionindx program_getentry (program ∗p)
- debuginfo ∗ morpho_getdebugfromindx (program ∗p, instructionindx indx)
- void program_bindobject (program ∗p, object ∗obj)

  Binds an object to a program.
- value program_internsymbol (program ∗p, value symbol)

  Interns a symbol into the programs symbol table.
- void **vm_graylistinit** (graylist ∗g)
- void **vm_graylistclear** (graylist ∗g)
- void **vm_graylistadd** (graylist ∗g, object ∗obj)
- vm ∗ morpho_newvm (void)
- void morpho_freevm (vm ∗v)
- void vm_freeobjects (vm ∗v)
- void vm_collectgarbage (vm ∗v)
- void vm_bindobject (vm ∗v, value obj)

  Binds an object to a Virtual Machine.
- void vm_disassembleinstruction (instruction instruction, ptrdiff_t indx, value ∗konst, value ∗reg)

  Disassembles a single instruction, writing the output to the console.
- void morpho_disassemble (program ∗code, unsigned int ∗matchline)
- void morpho_stacktrace (vm ∗v)
- void vm_gcmarkobject (vm ∗v, object ∗obj)
- void vm_gcmarkvalue (vm ∗v, value val)
- void vm_gcmarkdictionary (vm ∗v, dictionary ∗dict)
- void vm_gcmarkarray (vm ∗v, varray_value ∗array)
- void vm_gcmarkroots (vm ∗v)
- void **vm_gcmarkretainobject** (vm ∗v, object ∗obj)

- void vm_gctrace (vm ∗v)
- void vm_gcsweep (vm ∗v)
- void vm_runtimeerror (vm ∗v, ptrdiff_t iindx, errorid id,...)

  Raises a runtime error.
- void morpho_runtimeerror (vm ∗v, errorid id,...)

  Public interface to raise a runtime error.
- bool morpho_interpret (vm ∗v, program ∗code, error ∗err)

  Executes a sequence of code.
- void morpho_initialize (void)
- void morpho_finalize (void)

Variables

- value **initselector** = MORPHO_NIL
- value **indexselector** = MORPHO_NIL
- value **setindexselector** = MORPHO_NIL
- value **addselector** = MORPHO_NIL
- value **subselector** = MORPHO_NIL
- value **mulselector** = MORPHO_NIL
- value **divselector** = MORPHO_NIL
- value **printselector** = MORPHO_NIL
- value **enumerateselector** = MORPHO_NIL
- vm ∗ **globalvm** =NULL

10.43.1 Detailed Description

Morpho virtual machine.

Author

T J Atherton

10.43.2 Macro Definition Documentation

10.43.2.1 CHECKCMPTYPE

```
#define CHECKCMPTYPE(
                l,
                r )
```

Value:

```
            if (!morpho_ofsametype(l, r)) { \
                if (MORPHO_ISINTEGER(l) && MORPHO_ISFLOAT(r)) { \
                    l = MORPHO_INTEGERTOFLOAT(l); \
                } else if (MORPHO_ISFLOAT(l) && MORPHO_ISINTEGER(r)) { \
                    r = MORPHO_INTEGERTOFLOAT(right); \
                } \
            }
```

10.43.2.2 DISASSEMBLE_OPAB

```
#define DISASSEMBLE_OPAB(
              op,
              i )
```

Value:

```
disassemble_ABC(op, i, DISASSEMBLE_SHOWA | \
DISASSEMBLE_SHOWB, konst, reg)
```

10.43.2.3 DISASSEMBLE_OPABC

```
#define DISASSEMBLE_OPABC(
              op,
              i )
```

Value:

```
disassemble_ABC(op, i, DISASSEMBLE_SHOWA | \
DISASSEMBLE_SHOWB | DISASSEMBLE_SHOWC, \
konst, reg)
```

10.43.2.4 DISASSEMBLE_OPAcB

```
#define DISASSEMBLE_OPAcB(
              op,
              i )
```

Value:

```
disassemble_ABC(op, i, DISASSEMBLE_SHOWA | \
DISASSEMBLE_SHOWB | DISASSEMBLE_SHOWAc, konst, reg)
```

10.43.2.5 DISASSEMBLE_OPB

```
#define DISASSEMBLE_OPB(
              op,
              i )
```

Value:

```
disassemble_ABC(op, i, DISASSEMBLE_SHOWB, \
konst, reg)
```

10.43.2.6 DISASSEMBLE_SHOWA [1/2]

```
#define DISASSEMBLE_SHOWA 1
```

Disassembles a single ABC format instruction.

Disassembles a single ABx format instruction.

Parameters

op	string to print to describe the op
i	the instruction
show	which operands to show
konst	current constant table
reg	current registers
op	string to print to describe the op
i	the instruction
show	which operands to show
reg	current registers

10.43.2.7 DISASSEMBLE_SHOWA [2/2]

```
#define DISASSEMBLE_SHOWA 1
```

Disassembles a single ABC format instruction.

Disassembles a single ABx format instruction.

Parameters

op	string to print to describe the op
i	the instruction
show	which operands to show
konst	current constant table
reg	current registers
op	string to print to describe the op
i	the instruction
show	which operands to show
reg	current registers

10.43.2.8 INTERPRET_LOOP

```
#define INTERPRET_LOOP
```

Value:

```
        loop:                                                                   \
        bc=*pc++;                                                               \
        OPOPCODECNT(pp, bc)                                                     \
        op=DECODE_OP(bc);                                                       \
        OPCODECNT(op)                                                           \
        MORPHO_DISASSEMBLE_INSRUCTION(bc,pc-instructions,konst, reg)            \
        switch (op)
```

10.43.3 Function Documentation

10.43.3.1 morpho_disassemble()

```
void morpho_disassemble (
            program * code,
            unsigned int * matchline )
```

Disassembles a program

Parameters

code	- program to disassemble
matchline	- optional line number to match

10.43.3.2 morpho_finalize()

```
void morpho_finalize (
            void  )
```

Finalizes morpho

10.43.3.3 morpho_freevm()

```
void morpho_freevm (
            vm * v )
```

Frees a virtual machine

10.43.3.4 morpho_getdebugfromindx()

```
debuginfo* morpho_getdebugfromindx (
            program * p,
            instructionindx indx )
```

Finds the debugging info asssociated with instruction at indx

10.43.3.5 morpho_initialize()

```
void morpho_initialize (
            void  )
```

Initializes morpho

10.43.3.6 morpho_interpret()

```
bool morpho_interpret (
            vm * v,
            program * code,
            error * err )
```

Executes a sequence of code.

Parameters

v	The virtual machine to use
code	The program to execute

Returns

A morpho error

10.43.3.7 morpho_newvm()

```
vm* morpho_newvm (
             void  )
```

Creates a new virtual machine

10.43.3.8 morpho_runtimeerror()

```
void morpho_runtimeerror (
             vm * v,
             errorid id,
              ...  )
```

Public interface to raise a runtime error.

Parameters

v	the virtual machine
id	error id
...	additional data for sprintf.

10.43.3.9 morpho_stacktrace()

```
void morpho_stacktrace (
             vm * v )
```

Prints a stacktrace

10.43.3.10 program_bindobject()

```
void program_bindobject (
             program * p,
             object * obj )
```

Binds an object to a program.

Objects bound to the program are freed with the program; use for static data (e.g. held in constant tables)

10.43.3.11 program_getentry()

```
instructionindx program_getentry (
            program * p )
```

Gets the entry point of a program

10.43.3.12 program_internsymbol()

```
value program_internsymbol (
            program * p,
            value symbol )
```

Interns a symbol into the programs symbol table.

Note that the string is cloned if it does not exist already. Interning is used to accelerate dynamic lookups as the same string for a symbol will be used universally

10.43.3.13 program_setentry()

```
void program_setentry (
            program * p,
            instructionindx entry )
```

Sets the entry point of a program

10.43.3.14 vm_bindobject()

```
void vm_bindobject (
            vm * v,
            value obj )
```

Binds an object to a Virtual Machine.

Any object created during execution should be bound to a VM; this object is then managed by the garbage collector.

Parameters

v	the virtual machine
obj	object to bind

10.43.3.15 vm_collectgarbage()

```
void vm_collectgarbage (
            vm * v )
```

Collects garbage

10.43.3.16 vm_disassembleinstruction()

```
void vm_disassembleinstruction (
             instruction instruction,
             ptrdiff_t indx,
             value * konst,
             value * reg )
```

Disassembles a single instruction, writing the output to the console.

Parameters

instruction	The instruction to disassemble
indx	Instruction index to display
konst	current constant table
reg	current registers

10.43.3.17 vm_freeobjects()

```
void vm_freeobjects (
             vm * v )
```

Frees all objects bound to a virtual machine

10.43.3.18 vm_gcmarkarray()

```
void vm_gcmarkarray (
             vm * v,
             varray_value * array )
```

Marks all entries in an array

10.43.3.19 vm_gcmarkdictionary()

```
void vm_gcmarkdictionary (
             vm * v,
             dictionary * dict )
```

Marks all entries in a dictionary

10.43.3.20 vm_gcmarkobject()

```
void vm_gcmarkobject (
             vm * v,
             object * obj )
```

Marks an object as reachable

10.43.3.21 vm_gcmarkroots()

```
void vm_gcmarkroots (
            vm * v )
```

Searches a vm for all reachable objects Mark anything on the stack

Mark closure objects in use

10.43.3.22 vm_gcmarkvalue()

```
void vm_gcmarkvalue (
            vm * v,
            value val )
```

Marks a value as reachable

10.43.3.23 vm_gcsweep()

```
void vm_gcsweep (
            vm * v )
```

Go through the VM's object list and free all unmarked objects

10.43.3.24 vm_gctrace()

```
void vm_gctrace (
            vm * v )
```

Trace all objects on the graylist

10.43.3.25 vm_runtimeerror()

```
void vm_runtimeerror (
            vm * v,
            ptrdiff_t iindx,
            errorid id,
             ... )
```

Raises a runtime error.

Parameters

v	the virtual machine
id	error id
...	additional data for sprintf.

10.44 vm/vm.h File Reference

The Morpho virtual machine.

```
#include "core.h"
```

Macros

- #define **MORPHO_CORE**
- #define **MORPHO_PROGRAMSTART** 0

Functions

- void program_setentry (program *p, instructionindx entry)
- instructionindx program_getentry (program *p)
- varray_value * **program_getconstanttable** (program *p)
- void program_bindobject (program *p, object *obj)

  Binds an object to a program.
- value program_internsymbol (program *p, value symbol)

  Interns a symbol into the programs symbol table.
- void vm_disassembleinstruction (instruction bc, ptrdiff_t pc, value *konst, value *reg)

  Disassembles a single instruction, writing the output to the console.
- void **morpho_bindobject** (vm *v, value obj)
- void vm_freeobjects (vm *v)
- void **morpho_collectgarbage** (void)
- void morpho_initialize (void)
- void morpho_finalize (void)

10.44.1 Detailed Description

The Morpho virtual machine.

Author

T J Atherton

10.44.2 Function Documentation

10.44.2.1 morpho_finalize()

```
void morpho_finalize (
            void  )
```

Finalizes morpho

10.44.2.2 morpho_initialize()

```
void morpho_initialize (
             void  )
```

Initializes morpho

10.44.2.3 program_bindobject()

```
void program_bindobject (
             program * p,
             object * obj )
```

Binds an object to a program.

Objects bound to the program are freed with the program; use for static data (e.g. held in constant tables)

10.44.2.4 program_getentry()

```
instructionindx program_getentry (
             program * p )
```

Gets the entry point of a program

10.44.2.5 program_internsymbol()

```
value program_internsymbol (
             program * p,
             value symbol )
```

Interns a symbol into the programs symbol table.

Note that the string is cloned if it does not exist already. Interning is used to accelerate dynamic lookups as the same string for a symbol will be used universally

10.44.2.6 program_setentry()

```
void program_setentry (
             program * p,
             instructionindx entry )
```

Sets the entry point of a program

10.44.2.7 vm_disassembleinstruction()

```
void vm_disassembleinstruction (
             instruction instruction,
             ptrdiff_t indx,
             value * konst,
             value * reg )
```

Disassembles a single instruction, writing the output to the console.

Parameters

instruction	The instruction to disassemble
indx	Instruction index to display
konst	current constant table
reg	current registers

10.44.2.8 vm_freeobjects()

```
void vm_freeobjects (
             vm * v )
```

Frees all objects bound to a virtual machine

Index