



Primitives Reference



Revision 1.0 Dec. 29, 2005 Andescotia and the Marten logo are trademarks of Andescotia LLC.

Marten is a trademark of Andescotia LLC, registered in the U.S.

Codewarrior is a trademark of Metrowerks Corporation, registered in the U.S. and other countries.

Xcode and MacOS are trademarks of Apple Computer, Inc., registered in the U.S. and other countries.

© Copyright 2005. Andescotia LLC. ALL RIGHTS RESERVED.

It should be understood that Andescotia LLC reserves the right to make changes at any time, without further notice, to the Marten Integrated Development Environment (IDE) suite of software, documentation, example code, and related materials in order to improve them. This document is a member of that suite of products.

In addition, Andescotia LLC does not assume any liability arising from the application or use of any product in the Marten IDE product suite.

The products of the Marten IDE suite are not authorized for use in any capacity to develop software applications where the failure, malfunction, or any inaccuracy of the developed application carries a risk of death, bodily injury, or damage to tangible property. Examples of (but not limited to) such proscribed uses are control systems, medical devices, nuclear facilities, banking and other financial software, and emergency systems.

Documentation that is supplied in electronic form may be printed for use by the purchaser under their rights to "fair use". Except for "fair use" purposes, no portion of this document or any of the Marten IDE product suite may be reproduced or transmitted in any form or by any means, including electronic or mechanical, without prior written permission from Andescotia LLC.

ALL SOFTWARE, DOCUMENTATION, AND RELATED MATERIALS OF THE MARTEN IDE PRODUCT SUITE ARE SUBJECT TO THE MARTEN IDE ANDESCOTIA END USER LICENSE AGREEMENT.

Andescotia LLC Contact Information

Office:	Andescotia LLC
	524 Fieldstone Dr.
	Bozeman, MT 59715
Website:	www.andescotia.com
Technical Support:	techsupport@andescotia.com

Contents

	Contents	3
Chapter 1:	Working with libraries	5
	Library contents	5
	Loading libraries	5
	Viewing the libraries in a project	7
Chapter 2:	Marten Primitives	9
	Primitive documentation conventions	9
	Syntax description	10
	Categorized primitive information	12
	Primitives by category	12
	<u>Bit</u>	13
	Callbacks	16
	Data	20
	File_	22
	Graphics	26
	Input/Output	29
	Interpreter control	31
	<u>List</u>	31
	Logical/Relational	40
	Math	45
	Memory	53
	String	58
	System	63
	Type	66

I	nd	ex	7	1



Working with libraries

<u>Library contents</u>
 <u>Loading libraries</u>
 <u>Viewing the libraries in a project</u>

Every computer language has operators which generate new data from old. For example, the C language has the operators +, & &, and <<. These operators are part of the definition of the C language. The Marten IDE is somewhat different in that there are very few defined operators (one example is the "Get" operator). Instead, operators are added to a project, as needed. This allows different projects to use different sets of operators. These operators are called primitives and are contained in libraries which are placed into a project using the Add To Project command under the File menu.

Library contents

The following table shows the categories of primitives contained in the libraries packaged with Marten:

Library	contains the following primitive categories
Standard	Bit, Callbacks, Data, Interpreter control, List, Logical/Relational, Math, Memory, String, System, Type,
Carbon	File, Graphics, Input/Output

Loading libraries

In order to make use of standard primitives, custom primitives, or external definitions, you must first load the library containing those resources into your project.

For information on contents of the standard libraries delivered with Marten, see "Library contents" on page 1.

Contents Index Working with libraries

> To load a library into your project:

1. From the File menu, choose Add to Project.

A **Choose Object** navigation dialog opens.

 Use the Choose Object dialog to locate and select the library or libraries you want to add to your project. Libraries are usually installed in the /Library/Frameworks directory or in the user-relative directory, ~/Library/Frameworks.



3. Click Choose.

Once you save a project to which you have added a library, that library is subsequently added to the project automatically whenever you load the project into Marten.

Viewing the libraries in a project

You can view the libraries loaded into a project and inspect their contents.

- > To view the libraries currently loaded into a project:
 - 1. Double-click the primitives icon of a project item in a Projects window.



A Primitives of Project Name window opens.



Double-click a library icon to display its primitives.
 A Primitives of *Library Name* window opens.





Marten Primitives

Primitive documentation conventions Primitives by category

This chapter tells you everything you have to know about the built-in Marten primitives. It provides details on how primitives are documented in this chapter, provides a categorized listing of all available primitives, and includes a detailed description of each primitive.

Primitive documentation conventions

Primitive descriptions in this chapter have two components:

- <u>Syntax description</u> a graphic depiction of the primitive operation that provides a syntax diagram
- <u>Categorized primitive information</u> provide details on the function of the primitive, inputs and outputs, and other information required to use the primitive.

The following provides an example of the typical information required to use a primitive:

round-down	Number [Precision]
Description	Returns the nearest number less than or equal to the provided number according to the provided precision. Positive and negative values for the precision parameter dictate the number of decimal places to the right and left of the decimal point, respectively.
Inputs	Number <number>:</number>
Default(s)	Precision <integer>: Precision = 0 (return an integer).</integer>
Outputs See also	Result <number>: trunc, round, round-up</number>

Contents I Index Marten Primitives

Syntax description

The graphic shown with each primitive description acts as a syntax description for that primitive; it shows the primitive's default arity (numbers of inputs and outputs), as well as any optional terminals or roots. Each terminal or root node is labelled with a meaningful name.

Simple variable arity primitives

Some primitives have optional inputs or outputs. These are called variable arity primitives. For this type of primitive, names of optional parameters are enclosed in square brackets [].

For example, the **div** primitive has an optional **Dividend** input.



If two or more optional inputs are paired within square brackets, all must be provided.

Primitives with a variable number of inputs

In some cases, a primitive accepts a variable number of similar inputs. The **"join"** primitive, for example, concatenates two or more strings, making the first two inputs mandatory and additional inputs optional. The syntax diagram for this type of primitive looks like the following:



Conditional inputs

Some optional input parameters are actually conditional. For example, the **make-list** primitive has only one required input: **Length**. Optionally, you can provide a **Start** input and IF AND ONLY IF you provide a **Start** input, you can also provide a **Step** input. The syntax diagram for such a primitive looks like the following:



Variable type parameters

Some primitives accept an input or produce an output of more than one type. For example, a primitive that searches a list for an item with a particular value, might accept

Contents Index Marten Primitives

both integers and strings as the value to be found. In the detailed descriptions for such a parameter, all valid datatypes are listed.

Some primitives have parameters that accept virtually any type of Marten data. For example, a primitive that creates a list from a set of values will allow you to create list of values of any valid Marten datatype. The description for such a parameter has the <any> type designation. Read the entire description for such a primitive as there may be exceptions.

Primitives that return boolean results

Certain primitives perform tests or manipulate boolean input values to obtain a boolean result. Every such boolean primitive can either have one root, in which case it returns TRUE or FALSE, or no roots, in which case it succeeds or fails. Use an output root on a boolean primitive if you need to do further calculations with the boolean results such as "AND" the result with the result of another boolean primitive.

If a boolean primitive is used without a root, it should have an associated control so that possible failure of the operation does not result in an execution error:



The boolean primitives are those listed under in the Logical/Relational category, as well as any primitive with a question mark (?) at the end of the name.

Notes on lists and indices as inputs

Certain primitives have a list and one or more integers as input, where the integers are indices into the list or the list of lists. An example of this is the primitive get-nth:

	List N1 [N2]
	get-nth
get-nth	Element
Description	Given a list, returns an element specified by index. This primitive can also return elements from nested lists by passing it additional index numbers.
Inputs	List list>: the list to be searched.
	N1 <integer>: for a simple list, the index of the element to be returned; for a list of lists, the index of the nested list.</integer>
	N2 <integer>: the index of the element within a nested list or the index of another nested list. The final terminal must be the index of the element to be returned in the deepest nested list.</integer>
Outputs	Element <any>:</any>
See also	insert-nth, set-nth, set-nth!, split-nth

Contents Index Marten Primitives

If such a primitive receives an integer less than 1 or greater than the length of the list it accesses, an **Out of range** execution error occurs.

Categorized primitive information

Documentation for each primitive includes all details you need to work with the primitive. This information is categorized to make lookup easy. The most common categories are:

Description	Provides a straightforward description of the purpose of the primitive.
Inputs	Provides the names and types of inputs and, as required, further details or elaboration.
Outputs	Provides the names and types of outputs and, as required, further details or elaboration.
See also	Provides a listing of related primitives.
Other categories include:	
Compiler	Notes any differences in behavior of the primitive in compiled versus interpreted execution.
Default(s)	Provides any default values for parameters.
Equivalent	If another primitive provides identical function, its name is provided.
Example	Provides an example of the primitive.
Note	Provides any important or obscure details about usage of this primitive.
Side effects	Describes any additional or non-obvious processing, such as whether input values are altered.

Primitives by category

The primitives delivered as part of the standard Marten package, fall into the following categories:

- <u>Bit</u>
- Callbacks
- Data
- **File**
- Graphics
- Input/Output
- Interpreter control
- List
- Logical/Relational
- Math
- Memory
- String

- System
- Type

Bit

The Bit primitives allow you to perform bit arithmetic on Marten Integer data types. Currently Marten Integers store their values as 32-bit integers. Each description of a bit operation provides an example of the operation result.

The following primitives are provided:

bit-and	The output is the bitwise AND of two integers.
bit-not	The output is the bitwise COMPLEMENT an integer.
<u>bit-or</u>	The output is the bitwise OR of two integers.
<u>bit-shift-l</u>	The output is a bitwise shift left by one or more places on an integer. The rightmost bits of the result are set to zero (0) .
<u>bit-shift-r</u>	The output is a bitwise shift right with sign extension by one or more places on an integer.
<u>bit-xor</u>	The output is the bitwise EXCLUSIVE-OR of two integers.
test-all?	The output is TRUE if all of the set bits of an integer correspond to the set bits of a provided mask.
test-bit?	The output is TRUE if the bit in a specified position of an integer is set.
test-one?	The output is TRUE if ONE of the set bits of an integer corresponds to one of the set bits of a provided mask.



bit-and

Description	The bitwise AND of two integers is output. For example the result for 5 (b101) and 3 (b11) is 1 (b1).
Inputs	Returns the bitwise AND of two integers.
Inputs	InBits1 <integer>:</integer>
	InBits2 <integer>:</integer>
Outputs	OutBits <integer>:</integer>
See also	bit-not, bit-or, bit-shift-1, bit-shift-r, bit-xor



bit-not

Description	The bitwise COMPLEMENT of an integer is output.
Inputs	InBits <integer>:</integer>
Outputs	OutBits <integer>:</integer>
See also	bit-and, bit-or, bit-shift-l, bit-shift-r, bit-xor



bit-or

π

bit-shift-l	InBits N bit-shift-l OutBits	
Description	A bitwise shift left by one or more places for an integer is output. The N rightmost bits of the result are set to zero (0). For example the result for 5 (b101) shifted left by 2 is 20 (b10100).	
Inputs	InBits <integer>:</integer>	
	N <integer>: the number of places to the left that each bit is to be</integer>	

	shifted.
Outputs	OutBits <integer>:</integer>
See also	bit-and, bit-not, bit-or, bit-shift-r, bit-xor

Outputs See also

boolean

test-all?, test-one?

	bit-shift-r	
bit-shift-r	OutBits	
Description	A bitwise shift right with sign extension by one or more places for an integer is output. For example the result for 13 (b1101) shifted right by 2 is 3 (b11) and the result for -13 (b11110011) shifted right by 2 is -4 (b11111100).	
Inputs	InBits <integer>:</integer>	
	N < integer>: the number of places to the right that each bit is to be shifted.	
Outputs	OutBits <integer>:</integer>	
See also	bit-and, bit-not, bit-or, bit-shift-l, bit-xor	
hit-yor	InBits1 InBits2	
Description Inputs	The bitwise EXCLUSIVE-OR of two integers is output. InBits1 <integer>:</integer>	
.	InBits2 <integer>:</integer>	
Outputs See also	OutBits <integer>:</integer>	
1	Value Mask	
test-all?		
Description	Returns TRUE if all of the set bits of an integer correspond to the set bits of a provided mask.	
Inputs	Value <integer>:</integer>	
Outrasta	Mask <integer>:</integer>	
Outputs See also	boolean test-hit? test-one?	
	Value BitPosition	
test_hit?	📃 test-bit? 📃 🗙	
Description	Returns TRUE if the bit in a specified position of an integer is set.	
mputa	BitPosition <integer></integer>	
	Live option since of a	

Contents Index Marten Primitiv	es 8
test-one?	Value Mask
Description	Returns TRUE if ONE of the set bits of an integer corresponds to one of the set bits of a provided mask. For example the result for 5 (b101) and 3 (b11) is TRUE and the result for 5 (b101) and 2 (b10) is FALSE.
Inputs	Value <integer>: Mask <integer>:</integer></integer>
Outputs See also	boolean test-all?, test-bit?

Callbacks

Many languages such as C allow function arguments to be references to other functions. In the vernacular of C, these references are known as function pointers or callbacks. A callback is a reference to a function by its address, letting one function call another, without knowing its name.

The address of the callback is passed as an argument to the function, which in turn, calls the callback by address to perform specialized tasks.

For example, the following C function, **VPL_TestCallback**, requires the use of a callback:

```
long VPL_TestCallback(long input, void *functionPtr);
long VPL_TestCallback(long input, void *functionPtr)
{
    long result = 0;
    long (*addOne)(long);
    addOne = functionPtr;
    result = addSome(input);
    return result;
}
```

The function **VPL_TestCallback** accepts two parameters, the first is a long integer and the second is a callback. The declaration of **addOne** indicates that the callback is a function that takes a long integer as argument and returns a long integer. An example of such a callback is:

```
long MyCallback(long input);
long MyCallback(long input)
{
    long result = 0;
    result = input++;
    return result;
}
```

Marten callbacks let you write a Marten method that would stand in for **MyCallback**. The Marten code for MyCallback would be:

Contents Index Marten Primitives



In order for this method to be used as a callback in Marten, the C function **VPL_TestCallback** must be created and a definition of it must be made available to the MacVPL engine. In addition, a definition of the structure of the callback function must also be present. An example is presented below:

```
VPL_Parameter _TestCallbackProc_R = { kLongType,4,NULL,0,0,NULL};
VPL_Parameter _TestCallbackProc_1 = { kLongType,4,NULL,0,0,NULL};
VPL_ExtProcedure _TestCallbackProc_F = { "TestCallback-
Proc",NULL,&_TestCallbackProc_1,&_TestCallbackProc_R};
VPL_Parameter _TestCallback_R = { kShortType,2,NULL,0,0,NULL};
VPL_Parameter _TestCallback_2 = { kPointerType,0,"void",1,1,NULL};
VPL_Parameter _TestCallback_1 = { kLong-
Type,4,NULL,0,0,&_TestCallback_2};
VPL_ExtProcedure _TestCallback_F = { "TestCall-
back",VPL_TestCallback,&_TestCallback_1,&_TestCallback_R};
VPL_DictionaryNode VPX_MacVPL_Procedures[] ={ { "TestCallbackProc",&_TestCallbackProc_F},
    { { "TestCallback", &_TestCallback_F} } ;
```

Once a Marten extension has been created with those definitions, then the external procedure TestCallback can be used in a method and the method **My Callback** can be supplied as a callback. The following code illustrates this:



Callbacks are an advanced feature of Marten which require some knowledge of C or Pascal. A Marten method cannot be called directly by address from an external procedure. In order to provide the external procedure with an address to executable

Contents Index Marten Primitives

code, the Marten implementation of the callback primitive allocates a small amount of memory and writes a set of instructions into it that can be called directly by C. A reference to this memory is output from the callback primitive as an allocated block. This block is what is passed to the external procedure of interest.

Since memory is allocated, it should be freed up by using the dispose-callback primitive when the callback is no longer needed.

The first input to the callback primitive is the name of the Marten universal method to be used as the callback (for instance, "My Callback"). The required second input of the callback primitive is the name of the definition of the of the callback (in this case, "TestCallbackProc").

The optional third input of the callback primitive is a Marten object that will be passed in as the first input into the called method. This object is most usefully an instance of a class, which then allows the callback method to be data-determined (for example, "/ Open"). Since this object will now be "owned" by the system, all Marten references to the object may disappear. Normally when this happens, the object would be garbagecollected. To prevent this from happening, any object passed to the callback primitive has its "system-used" flag set to TRUE. When the object is truly available for disposal, then the system-used primitive should be used to reset the flag to FALSE to allow the object to be managed correctly.

Finally, the optional fourth input of the callback primitive is a Boolean. The default value is FALSE, which means that the arguments to the callback method will be passed into the method as an equal (or plus one, if the third input is non-NULL) number of roots of the input operation. If the value is TRUE, then the arguments will be passed into the method as a list. An example of using the callback primitive in this manner is:



And the callback class method to be called is:



Note the use of the system-used primitive to allow the instance of Sub Class to be garbage-collected.

Marten provides the following primitives for working with callbacks:

- <u>callback</u>
- dispose-callback

	MethodName CallbackDef [Object][ListInput?]		
	callback		
callback	MethodNamePointer		
Description	Returns a pointer to universal method suitable for use as a callback to an external procedure.		
Note	For background on the use of callbacks in Marten, see <u>"Callbacks"</u> on page 8.		
Inputs	MethodName <string>: the name of the universal method to be used as the callback. CallbackDef <string>: the name of the definition of the of the callback</string></string>		
Object < <any>>: the object that will be passed as the fir into the called method. ListInput? <boolean>:</boolean></any>			
			Outputs Default(s) See also



Description	Disposes the memory in the pointer used for the callback. This	
	primitive must be called after the routine to which the	
	MethodPointer was passed completes execution and returns.	
Inputs	MethodPointer <ablock@></ablock@>	
See also	<u>callback</u>	

Data

The data primitives provide miscellaneous functionality for working with Marten data. The following primitives are provided:

Creates a copy of a Marten data element.	
Returns the class name and a list of instance attribute values of a provided instance.	
Creates and returns an instance of a specified class with a set of provided instance attribute values.	
Creates a shallow copy of a Marten data element.	



сору

Description	Makes a copy of a Marten data element. If the item is a complex data type (instance or list), the referenced complex objects in the attributes or list slots are recursively copied to arbitrary depth. Referenced simple objects (integers, strings, and so on) contained in the complex object are not copied, but their use counts are incremented.
Inputs	Item <any>:</any>
Outputs See also	ItemCopy <any>: shallow copy</any>

	inst-to-list	
inst-to-list	Class List	
Description	Returns the class name and a list of instance attribute values of a provided instance.	
Inputs	Instance << <any>></any>	
Outputs	Class <string>: the name of the class from which the instance was spawned.</string>	
Note	List list>: a list of the values of all instance attributes. Attribute names are not provided. Attribute values are listed in the same order in which attributes are displayed in the Attribute window for the owning class.	
See also	list-to-inst	
	Class List	
	list-to-inst	
list-to-inst	Instance	
Description	Creates and returns an instance of a specified class with a set of provided instance attribute values.	
Inputs	Class <string>: the name of the class for which an instance is to be spawned.</string>	
	List list>: a list of values for the class' instance attributes. The list should be ordered identically to the order displayed in an Attribute window for the owning class.	
Outputs	Instance < <any>></any>	
Note	The specified class must already exist, and the number of its instance attributes must equal the length of the provided list of instance attribute values.	
See also	<u>inst-to-list</u>	
	Item	
	shanow-copy	
shallow copy	ItemCopy	
Description	Makes a shallow copy of a Marten data element. If the item is a complex object (instance or list), the referenced objects (both complex and simple) in the attributes or list slots are not copied, but their use counts are incremented.	

InputsItem <any>:OutputsItemCopy <any>:See alsocopy

13

Instance

File

The Marten File primitives provide functionality at a slightly higher level than that of the Macintosh File System. They are intended to provide enough functionality to cover all common file operations.

The following primitives are provided:

<u>close-file</u>	Closes a specified file.	
create-object-file	Creates a new file of a specified type.	
create-text-file	Creates a new file of type 'TEXT'.	
delete-file	Deletes a specified file.	
get-file	Opens an Open File dialog that lets a user locate, select, and open a file.	
open-file	Opens a file with a specified access level.	
put-file	Opens a Save File dialog that lets a user save a file in a specified location.	
read-buffer	Reads in the contents of the file and places them into an external block	
read-object	Reads in the contents of a Marten object file and returns the object.	
read-text	Reads in the contents of a text file and returns a String.	
write-object	Writes the provided object to the specified file.	
write-text	Writes a provided string to a specified text file.	



close-file

Description	Closes the file identified by the file reference number created by the primitive open-file.
Inputs	Reference number <integer>: The MacOS X file reference number.</integer>
Note See also	The primitive can FAIL if the file cannot be closed. <u>create-object-file</u> , <u>create-text-file</u>

FSSpec	Script code	File type
	reate-objec	:t-file

create-object-file

Description	Creates a new file on disk. If a file already exists, it is deleted and then the new one created.
Inputs	FSSpec <external block="">: A MacOS X FSSpec structure.</external>
	Script code <integer>: A MacOS X script code returned by the <u>put-file</u> primitive.</integer>
	File type <integer>: A MacOS X file type (for example, 'TEXT').</integer>

Contents	Index	Marten Primitives
----------	-------	-------------------

Note See also	The primitive can FAIL if the file cannot be created. create-text-file, close-file, open-file
	FSSpec Script code
create-text-file	create-text-file
Description	Creates a new file on disk of file type 'TEXT'. If a file already exists, it is deleted and then the new one created.
Inputs	FSSpec <external block="">: A MacOS X FSSpec structure.</external>
	Script code <integer>: A MacOS X script code returned by the <u>put-file</u> primitive.</integer>
Note	The primitive can FAIL if the file cannot be created.
See also	create-object-file, close-file, open-file
	FSSpec
delete-file	📃 delete-file 📃
Description	Deletes the specified file
Inputs	FSSpec <external block="">: A MacOS X FSSpec structure.</external>
Note	The primitive can FAIL if the file cannot be created.
	get-file
get-file	File type FSSpec
Description	Opens a MacOS X Open File Navigation Dialog. If the user selects a file, the type and the FSSpec are returned. If user cancels, then get-file FAILs.
Outputs	File type <integer>: A MacOS X file type (for example, 'TEXT').</integer>
Note	This primitive fails if the user cancels the dialog.
	FSSpec <external block="">: A MacOS X FSSpec structure.</external>
Example	





Reference number Object

write-object

write-object

Description	Writes the provided object to the specified file.
Inputs	Reference number <integer>: A MacOS X file reference number.</integer>
	Object <any>: Any Marten object.</any>
Note	This primitive can FAIL if the object cannot be written to file.
See also	read-object, read-text, read-buffer, write-text

write-text	write-text
Description	Writes the string to the text file identified by the file reference number.
Inputs	Reference number <integer>: A MacOS X file reference number.</integer>
Note See also	Text <string>: The text to be written to the file. This primitive can FAIL if the text cannot be written to file. <u>read-object</u>, <u>read-text</u>, <u>read-buffer</u>, <u>write-object</u></string>

Graphics

The Graphics primitives perform manipulations and calculations on the following types:

- Point
- RGB
- Rect.

These are represented textually in Marten as two, three, or four integers, respectively, separated by spaces, inside braces. For example, $\{58\}$ is a Point, with a vertical coordinate of 5 and a horizontal coordinate of 8. $\{10\ 15\ 20\}$ is an RGBType, with a red component of 10, a green component of 15, and a blue component of 20. Finally, $\{0\ 0\ 100\ 100\}$ is a Rect, with top and left coordinates of 0, and bottom and right coordinates of 100.

Note: For more information on these datatypes, refer to the *Marten Users Guide*.

The following primitives are provided:

list-to-Point	Creates a Mac OS X Point from supplied co- ordinates.
list-to-Rect	Creates a Mac OS X Rect from supplied co-ordinates.
list-to-RGB	Creates a Mac OS X RGB specifier from supplied colour values.
Point-to-list	Returns the co-ordinates specified by a Mac OS X Point.
Rect-to-list	Returns the co-ordinates specified by a Mac OS X Rect.
RGB-to-list	Returns the individual colour values specified by a Mac OS X RGB.



list-to-Point

Description	Returns the point two specified by co-ordinates.
Inputs	Co-ords-list <list>: the two integers specifying the points co-ordinates.</list>
Outputs	Point <external block="">: a MacOS X Point data structure.</external>
See also	Point-to-list



list-to-Rect

Description	Returns the Rect specified by a set of co-ordinates.
Inputs	Co-ords-list <list>: the four integers specifying the Rect co-ordinates.</list>
Outputs Example	Point <external block="">: a MacOS X Rect structure.</external>



See also





list-to-RGB

Description Inputs Outputs See also Returns the specified RGB structure. **Specifiers-list** <list>: the three integers specifying the RGB. **RGB** <External Block>: a MacOS X RGB structure. <u>RGB-to-list</u>



Point-to-list

Description	Returns the list of two coordinates specified by a Point.
Inputs	Point <external block="">: a MacOS X Point structure.</external>
Outputs	Specifiers-list <list>: the two integers specifying the co-ordinates of the Point.</list>
See also	list-to-Point



Rect-to-list

Description	Returns the list of four coordinates specified by a Rect.
Inputs	Rect <external block="">: a MacOS X Rect structure.</external>
Outputs	Co-ords-list <list>: the four integers specifying the co-ordinates of the Rect</list>



RGB-to-list

Description	Returns the list of three colour specifiers of a Rect.
Inputs	RGB <external block="">: a MacOS X RGB structure.</external>
Outputs	Specifiers-list <list>: the three integers specifying the individual colors of the RGB.</list>
See also	list-to-RGB

Input/Output

	Prompt Button1 [Button2 [Button3]]
	answer
answer	ButtonClicked
Description	Displays a dialog with one to three buttons labelled with textual representations of provided Marten values and returns the value corresponding to the button that the user clicks. The modal dialog has a specified textual prompt and 1, 2, or 3 horizontally arranged buttons. Marten uses textual representations of Button1 to Button3 if they are not already strings.
Inputs	Prompt <string>: a textual prompt to aid the user in making a choice.</string>
	Button1 <any>: any valid Marten data item.</any>
	Button2 <any>: any valid Marten data item.</any>
	Button3 <any>: any valid Marten data item.</any>
Outputs	ButtonClicked <any>: the value of the input parameter corresponding to the user's selection.</any>
See also	answer-v, ask, select
	Prompt Button1 [Button2 [Button3]]
answer-v	ButtonClicked
Description	Displays a dialog with one to three buttons labelled with textual representations of provided Marten values and returns the value corresponding to the button that the user clicks. The modal dialog has a specified textual prompt and 1, 2, or 3 vertically arranged buttons. Marten uses textual representations of Button1 to Button3 if they are not already strings.
Inputs	Prompt <string>: a textual prompt to aid the user in making a decision.</string>
	Button1 <any>: any valid Marten data item.</any>
	Button2 <any>: any valid Marten data item.</any>
	Button3 <any>: any valid Marten data item.</any>
Outputs	ButtonClicked <any>: the value of the input corresponding to the user's selection.</any>
See also	answer, ask, select

	[Prompt [DefaultValue]]
	ask
ask	Value Canceled?
Description	Opens a modal dialog prompting a user for input. The dialog has two buttons (Cancel and OK), an editable area, a textual prompt, and a default value in the editable area.
Inputs	Prompt <string>: a textual prompt to aid the user in providing a value.</string>
Noto	DefaultValue <any>: an initial, default value to be used as input if the user presses OK without typing a value in the editable area.</any>
Note	or an instance of a class
Default(s)	Prompt = 'Enter value'; DefaultValue = ''
Outputs	Value <any>: Contains the last value entered and displayed. Canceled? <boolean>: True if the user pressed the Cancel button, false if they pressed the OK button.</boolean></any>
See also	answer, answer-v, select
	Strings [Prompt]
select	SelectedString
Description	Opens a modal dialog prompting the user to make a selection from a list of provided alternatives. The modal dialog has a scrolling list, two buttons (Select and Cancel), and, optionally, a textual prompt.
Inputs	Strings <list>: the items that are to be displayed in the scroll list. Prompt <string>: a textual prompt displayed in the dialog</string></list>
Outputs	SelectedString <string null="" ="">: if the dialog is dismissed with the Select button and a string was selected, then this parameter contains the selected string; otherwise it has a value of NULL.</string>
See also	answer, answer-v, ask
	Item1 [Item2]
show	snow
Description	Displays output in a modal dialog. The dialog contains a string obtained by concatenating textual representations of the inputs.
Inputs	Item1 <any>: the first of the Marten values in the concatenation of the textual representations.</any>
	Item2 <any>: one terminal for each remaining Marten value to be displayed.</any>
Note See also	Inputs cannot be instances of classes or Windows types. ask

Interpreter control

The following primitives are provided for use with the Interpreter:

<u>call</u>	Similar to injecting a method name on an operation, but it provides more flexibility in that it allows you to call class-based and universals.
compiled?	Result is TRUE if your program is compiled, and FALSE if your program is interpreted.
	MethodName Inputs OutCount
call	[Outputs]
Description	Using call is similar to injecting a method name on an operation, but it provides more flexibility in that it allows you to call class-based and universals.
Inputs	MethodName <string>: The name of the method to call; this can be class-based (beginning with a / slash or // double-slash), or a universal (no slash).</string>
	Inputs <list>: A list of values to be passed into the method.</list>
	Outcount <integer>: The number of outputs to expect.</integer>
Outputs	Outputs <list>: The list of outputs produced by the called method.</list>
Note	Since this call functions like an actual call to the method in question, if the method is not found, the system asks if you want to create it.
	Do not use Outputs on the call primitive if the method you are
	calling may terminate or fail. Under those conditions, a method produces undefined outputs.
	If the call generates an execution error (such as when the method is not defined) in the interpreter, execution will halt on the call operation itself, with the error message displaying the execution error. Similarly, if the called method fails, any control placed on the call operation itself will respond to the failure.
compiled?	compiled?
Description	Result is TRUE if your program is compiled, and FALSE if your program is interpreted. This primitive can return a boolean result.

The list primitives provide a number of functions for working with lists of all types. The following primitives are provided:

List

Contents Index Marten Primitives

<u>(in)</u>	Searches a list for a specified item and returns the index of the first occurrence of the item.
(join)	Concatenates two or more lists.
<u>(length)</u>	Returns the number of elements in a list.
attach-l	Adds elements to the front of a list.
attach-r	Adds elements to the end of a list.
detach-l	Remove the first <i>n</i> elements of a list.
detach-nth	Removes the <i>nth</i> element of a list.
detach-r	Removes the last <i>n</i> elements.
find-instance	Searches a list for a specified instance.
find-sorted	Performs a binary search on a list.
get-nth	Returns the <i>nth</i> element of a list.
<u>insert-nth</u>	Inserts a new item in a list at a specified position.
make-list	Creates a new list and lets you specify values programmatically.
pack	Creates a new list of provided items.
reverse	Reverses the order of items in a list.
<u>set-nth</u>	Sets the value of the <i>nth</i> element in a list.
set-nth!	Sets the value of the <i>nth</i> element in a list, directly modifying the elements (not copies).
sort	Sorts the elements in a list.
<u>split-nth</u>	Splits a list into two lists at a specified position.
<u>unpack</u>	Returns the first n elements of a list.

(in)	List Item [StartIndex] (in) FoundIndex
Description	Searches a list for a specified item and returns the index of the first occurrence of the item. Optionally, you can pass an index number which will be treated as the first element of the search. Note that a list element is found only if it equals Item in the sense defined by the $\underline{=}$ primitive.
Inputs	List <list>: the list to be searched.</list>
	Item <any>: the value to search for.</any>
Default(s)	StartIndex <integer>: the index of the first item to be checked. StartIndex = 1</integer>
Outputs	FoundIndex <integer>: the index of the found item or zero (0) if the item does not occur in the list.</integer>
See also	(length), (join)



(join)	List
Description	Concatenates two or more lists.
Inputs	List1 list>: the list whose items are to be the first set of items in the new list.
	List2 <list>: the list whose items are to follow the items of the</list>
	List1 items in the new list.
Outrasta	List3 list>: any remaining lists, one per terminal.
See also	(in), (length)
	List
	(length)
(length)	Length
Description	Returns the length (number of elements) of a list.
Inputs	List <list>:</list>
Outputs	Length <integer>:</integer>
See also	<u>(in)</u> , (join)
	Element1 [Element2] List
	attach-I
attach-l	NewList
Description	Creates a new list by concatenating one or more provided elements (Element1) and an existing list.
Inputs	Element1 <any>: the element that is to be the first item in the list.</any>
	Element2 <any>: the second and subsequent items, one per terminal.</any>
	List <list>: the existing list.</list>
Outputs	NewList list>: the new list.
See also	<u>attach-l, attach-r, detach-l, detach-nth, detach-r</u>

	List Element1 [Element2]
	attach-r
attach-r	NewList
Description	Creates a new list by concatenating an existing list and one or more provided elements
Inputs	List the existing list.
	Element1 <any>: the first item to be appended to the list.</any>
	Element2 <any>: any remaining items to be appended to the</any>
Outpute	list, one per terminal.
See also	NewList < list>: the new list. attach-l_detach-l_detach-nth_detach-r
	InList
	detach-l
	Flomenta [Flomenta Flomenta] Outlist
detach-l	Elementi [Elementz Elementii] OdtEist
Description	For N+1 output roots, returns the first N elements of a list and a
Inputs	InList < list>: a list with at least N elements
Outputs	Element1 <any>: the first element in the list</any>
	Element2 ElementN <any>: elements 2 - N of the list.</any>
	OutList <list>: the remaining elements in the list.</list>
See also	attach-l, attach-r, detach-nth, detach-r
	InList N
	detach-nth
detach-nth	OutList Element
Description	Returns the Nth element of a list and a new list that results from removing the Nth element from the original list.
Inputs	InList <list>: the original list.</list>
	\mathbf{N} <integer>: the index of the item to be removed from the list.</integer>
Outputs	OutList <list>: the new list.</list>
See also	Element <any>: the item removed the original list. attach-l, attach-r, detach-l, detach-r</any>

	InList
	detach-r
detach-r	OutList Element1 [Element2 ElementN]
Description	For N+1 output roots, returns the last N elements of a list and a new list containing the remaining elements of the original list.
Inputs	InList <list>: a list containing at least N elements.</list>
Outputs	OutList <list>: a new list containing the remaining elements of the original list.</list>
	Element1 <any>:</any>
	Element2 ElementN <any>:</any>
See also	attach-l, attach-r, detach-l, detach-nth
	List AttributeName Value [StartIndex]
	find-instance
find-instance	FoundIndex Instance
Description	Given a list of Instances, returns the index of the first instance containing a specified attribute with a provided value.Optionally, you can provide an index number to use as the starting point of the search.
Inputs	List <list>: a list of instances.</list>
	AttributeName <string>: the name of the attribute to search.</string>
	Value <any>: the value of the attribute to search for.</any>
	StartIndex <integer>: the index within the list at which the search is to start.</integer>
Default(s)	StartIndex = 1
Outputs	FoundIndex <integer>: if the instance was found, this parameter contains the index of the instance; otherwise it contains a value of zero (0).</integer>
See also	Instance < <instance>> <null>: if the instance was found, this parameter contains the instance; otherwise it has a value of NULL. find-sorted</null></instance>

	List Item [AttributeName]
	find-sorted
find-sorted	Found Index
Description	Uses a binary search to find an item in a SORTED list of numbers, strings, or instances. Using find-sorted, as opposed to find-instance, is faster because it uses a binary search.
Inputs	List <list>: the list to be searched.</list>
	Item <string number="" ="">: the value to be searched for.</string>
	AttributeName <string>: if the list is a list of instances, this parameter passes the name of the attribute to be searched for the given value.</string>
Outputs	Found <boolean>:</boolean>
	Index <integer>: if the Found parameter has a value of TRUE, this parameter contains the index of the item in the list that had the specified value. If the Found parameter has a value of FALSE, this parameter stores the index where the specified value can be inserted.</integer>
See also	find-instance
	List N1 [N2]
get-nth	Element
Description	Given a list, returns an element specified by index. This primitive can also return elements from nested lists by passing it additional index numbers.
Inputs	List <list>: the list to be searched.</list>
	N1 <integer>: for a simple list, the index of the element to be returned; for a list of lists, the index of the nested list.</integer>
	N2 <integer>: the index of the element within a nested list or the index of another nested list. The final terminal must be the index of the element to be returned in the deepest nested list.</integer>
Outputs	Element <any>:</any>
See also	insert-nth, set-nth, set-nth!, split-nth

insert-nth List Description Creates a new list by inserting a provided element at a specified position in an existing list. Inputs OldList <list>: the existing list. Inputs OldList Outputs Element <any>: the item to be added to the existing list. Index <integer>: the position at which the item is to be placed in the new list. Outputs Element <list>: the new list. see also get-nth, set-nth!, split-nth make-list Length [Start [Step]] make-list Length control and the subsequent item values based on addition of a provided value. Inputs Length <integer>: the number of items in the list. If the Start cany>: the value of the first item in the list and fir the Step parameter is not provided, the value of all items in the Step parameter is not provided, the value of all items in the list. If the Start parameter is not provided, the value of all items in the list. If the Start parameter is not provided, all items of the list are created with an initial value of the next item in the list. Default(s) Start = NULL Outputs Element1 [Element2] get List Default(s) Start = NULL Outputs Element1 <any>: the value of the first item in the list. If the Start parameter is not provided, all items of the list are created with an initial value of NULL. <</any></integer></list></integer></any></list>		OldList Element Index
insert-nth Description Inputs Creates a new list by inserting a provided element at a specified position in an existing list. Inputs OldList Creates a new list by inserting a provided element at a specified position in an existing list. Element <any>: the existing list. Index <integers: <li="" also="" at="" be="" element="" in="" is="" item="" list.="" new="" placed="" position="" see="" the="" to="" which="">Set = nth, set-nth, set-nth, split-nth Element Creates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide an initial value for the first item in the list and if the Step parameter is not provided, all items of the list are created with an initial value of NULL. Start <any>: the value of NULL. Start = NULL Outputs List Lement1 [Element2] pack List List Description Creates a list of is of one or more elements. Default(s) Outputs List Lement1 [Element2] pack List List List List Lement1 [Element2] pack List List List https://www.list.item item item item item item item item</any></integers:></any>		
insert-nth Element Description Creates a new list by inserting a provided element at a specified position in an existing list. Inputs OldList <list>: the existing list. Element state item to be added to the existing list. Index sinteger>: the position at which the item is to be placed in the new list. Outputs See also Element set-nth, set-nth, split-nth List Description Creates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide a initial value for the first item in the list and if the Step parameter is not provided, all items of the list are created with an initial value of NULL. Start <a <a="" a="" add="" also="" creates="" description="" element1="" elements.="" href="mailto:system: the value of the first lement1" in="" inputs="" item="" list="" list.="" mailto:system:="" more="" next="" number="" obtain="" of="" one="" or="" pack="" see="" the="" to="" value="">Element2 pack List Description Creates a list of list of one or more elements. Inputs Element1 Element1 Element2 </list>		_ insert-nth _
Description Creates a new list by inserting a provided element at a specified position in an existing list. Inputs OldList text list>: the existing list. Element <any>: the item to be added to the existing list. Index <integer>: the position at which the item is to be placed in the new list. Outputs Element tist>: the new list. See also get-nth, set-nth!, split-nth make-list Creates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide an initial value for the first item in the list and have subsequent item values based on addition of a provided value. Inputs Length <integer>: the number of items in the list. Start parameter is not provided, all items of the list are created with an initial value of the first item in the list. If the Start parameter is not provided, all items of the list are created with an initial value of NULL. Start = NULL Start = NULL Outputs Start = NULL Default(s) Start = NULL Outputs Element1 [Element2] pack Creates a list of list of one or more elements. Inputs Element1 <any>: the value of the first element in the list. Start = null. Start = NULL Outputs Start = NULL See also Creates a l</any></integer></integer></any>	insert-nth	Element
Inputs OldList <list>: the existing list. Element <any>: the item to be added to the existing list. Index <integer>: the position at which the item is to be placed in the new list. Outputs Element <list>: the new list. See also get-nth, set-nth!, split-nth make-list Length [Start [Step]] make-list Length (Start [Step]] make-list Creates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide an initial value for the first item in the list and have subsequent item values based on addition of a provided value. Inputs Length <integer>: the number of items in the list. Start <any>: the value of the first item in the list and if the Step parameter is not provided, the value of al litems of the list. If the Start parameter is not provided, all items of the list are created with an initial value of NULL. Default(s) Start = NULL Outputs Start <= NULL</any></integer></list></integer></any></list>	Description	Creates a new list by inserting a provided element at a specified position in an existing list.
Element <any>: the item to be added to the existing list. Index <integer>: the position at which the item is to be placed in the new list. See also Element <list>: the new list. get-nth, set-nth, set-nth!, split-nth Length [Start [Step]] make-list Description Creates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide an initial value for the first item in the list and have subsequent item values based on addition of a provided value. Inputs Length <integer>: the number of items in the list. Start <any>: the value of the first item in the list and if the Step parameter is not provided, all items of the list are created with an initial value of NULL. Start parameter is not provided, all items of the list are created with an initial value of the next item in the list. Default(s) Start = NULL Outputs Start <any>: the number to add to the value of a list item to obtain the value of the next item in the list. See also pack List <list>: the new list. pack List <list>: the new list. pack List <list <ist="">: the new list. Default(s) Creates a list of list of one or more elements. Inputs Element1 <any>: the value of the first element in the list. Des</any></list></list></list></any></any></integer></list></integer></any>	Inputs	OldList <list>: the existing list.</list>
Index <integer>: the position at which the item is to be placed in the new list. See also Element <list>: the new list. get-nth, set-nth, set-nth, split-nth make-list Description Creates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide an initial value for the first item in the list and have subsequent item values based on addition of a provided value. Inputs Length <integer>: the number of items in the list. Start <any>: the value of the first item in the list and if the Step parameter is not provided, all items of the list are created with an initial value of NULL. Start <any>: the value of the next item in the list. Default(s) Start = NULL Outputs Start = NULL pack List Description Creates a list of list of one or more elements. pack Element1 [Element2] pack List Description Creates a list of list of one or more elements. Inputs Element1 <any>: the value of the first element in the list. Start = any>: the value of subsequent items in the list. Description Creates a list of list of one or more elements. Inputs Element1 <any>: the value of subsequent items in the list, one per terminal.</any></any></any></any></integer></list></integer>		Element <any>: the item to be added to the existing list.</any>
Outputs See also Element <list>: the new list. get-nth, set-nth, split-nth make-list Length [Start [Step]] make-list List Description Creates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide an initial value for the first item in the list and have subsequent item values based on addition of a provided value. Inputs Length <integer>: the number of items in the list. Start <any>: the value of the first item in the list and if the Step parameter is not provided, all items of the list are created with an initial value of NULL. Default(s) Start = NULL Outputs List <list>: the new list. pack Element1 [Element2] pack Creates a list of list of one or more elements. Inputs Creates a list of list of one or more elements. Lengent1 <any>: the value of the first element in the list. Description Creates a list of list of one or more elements. Inputs Element1 <any>: the value of the first element in the list, one per terminal.</any></any></list></any></integer></list>		Index <integer>: the position at which the item is to be placed in the new list.</integer>
See also get-nth, set-nth, split-nth make-list	Outputs	Element <list>: the new list.</list>
make-list Description Creates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide an initial value for the first item in the list and have subsequent item values based on addition of a provided value. Inputs Length <integer>: the number of items in the list. Start <any>: the value of the first item in the list and if the Step parameter is not provided, the value of all items in the list. If the Start parameter is not provided, all items of the list are created with an initial value of the next item in the list. Default(s) Start = NULL Outputs Start = NULL See also pack pack Element1 [Element2] pack Creates a list of list of one or more elements. Inputs Creates a list of list of one or more elements. Lement1 <any>: the value of subsequent items in the list, one per terminal.</any></any></integer>	See also	get-nth, set-nth, set-nth!, split-nth
make-list List Description Creates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide an initial value for the first item in the list and have subsequent item values based on addition of a provided value. Inputs Length <integer>: the number of items in the list. Start <any>: the value of the first item in the list and if the Step parameter is not provided, the value of all items of the list are created with an initial value of NULL. Step <number>: the number to add to the value of a list item to obtain the value of the next item in the list. Default(s) Start = NULL Outputs List list <list>: the new list. pack Element1 [Element2] pack Creates a list of list of one or more elements. Inputs Creates a list of list of one or more elements. Element1 <any>: the value of subsequent items in the list. Creates a list of list of one or more elements. Element1 (any>: the value of subsequent items in the list. Description Creates a list of list of one or more elements. Element1 (any>: the value of subsequent items in the list. Creates a list of list of one or more elements. Element2 (any>: the value of subsequent items in the list. Element2 (any>: the value of subsequent items in the list.</any></list></number></any></integer>		Length [Start [Step]]
make-listListDescriptionCreates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide an initial value for the first item in the list and have subsequent item values based on addition of a provided value.InputsLength <integer>: the number of items in the list. Start <any>: the value of the first item in the list and if the Step parameter is not provided, the value of all items of the list are created with an initial value of NULL. Step <number>: the number to add to the value of a list item to obtain the value of the next item in the list.Default(s)Start = NULL List <list>: the new list. packDescriptionCreates a list of list of one or more elements. Element1 <any>: the value of the first element in the list. Element2 <any>: the value of subsequent items in the list. List, one per terminal.</any></any></list></number></any></integer>		make-list
Description Creates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide an initial value for the first item in the list and have subsequent item values based on addition of a provided value. Inputs Length <integer>: the number of items in the list. Start <any>: the value of the first item in the list and if the Step parameter is not provided, all items of the list are created with an initial value of NULL. Step <number>: the number to add to the value of a list item to obtain the value of the next item in the list. Start = NULL Outputs See also pack Description Inputs Creates a list of list of one or more elements. Element1 (any>: the value of subsequent items in the list. Default(s) Outputs Description Creates a list of list of one or more elements. Inputs Creates a list of list of one or more elements. Element1 Element2 Creates a list of list of one or more elements. Inputs Creates a list of list of one or more elements. Element2 Creates a list of list of one or subsequent items in the list, one per terminal.</number></any></integer>	make-list	List
Inputs Length <integer>: the number of items in the list. Start <any>: the value of the first item in the list and if the Step parameter is not provided, the value of all items in the list. If the Start parameter is not provided, all items of the list are created with an initial value of NULL. Step <number>: the number to add to the value of a list item to obtain the value of the next item in the list. Default(s) Start = NULL Outputs List <list>: the new list. See also pack Description Creates a list of list of one or more elements. Inputs Element1 <any>: the value of subsequent items in the list. Element1 <any>: the value of subsequent items in the list.</any></any></list></number></any></integer>	Description	Creates a list of a specified length. The initial values of items in the list can be all NULLs, all identical values, or you can provide an initial value for the first item in the list and have subsequent item values based on addition of a provided value.
Start <any>: the value of the first item in the list and if the Step parameter is not provided, the value of all items in the list. If the Start parameter is not provided, all items of the list are created with an initial value of NULL. Step <number>: the number to add to the value of a list item to obtain the value of the next item in the list. Default(s) Start = NULL Outputs See also pack Element1 [Element2] List List Creates a list of list of one or more elements. Inputs Element1 <any>: the value of subsequent items in the list, one per terminal.</any></number></any>	Inputs	Length <integer>: the number of items in the list.</integer>
Step <number>: the number to add to the value of a list item to obtain the value of the next item in the list. Default(s) Start = NULL Outputs List <list>: the new list. See also pack Element1 [Element2] pack Description Inputs Element1 <any>: the value of the first element in the list. Element2 <any>: the value of subsequent items in the list, one per terminal.</any></any></list></number>	·	Start <any>: the value of the first item in the list and if the Step parameter is not provided, the value of all items in the list. If the Start parameter is not provided, all items of the list are created with an initial value of NULL.</any>
Default(s) Start = NULL Outputs List <list>: the new list. See also pack Element1 [Element2] pack Description Inputs Creates a list of list of one or more elements. Element1 < any>: the value of the first element in the list. Element2 < any>: the value of subsequent items in the list, one per terminal.</list>		Step <number>: the number to add to the value of a list item to</number>
Outputs List < list>: the new list. See also pack Element1 [Element2] pack Description Inputs Creates a list of list of one or more elements. Element1 < any>: the value of the first element in the list. Element2 < any>: the value of subsequent items in the list, one per terminal.	Default(s)	Start – NIII I
See also pack pack Element1 [Element2] pack List Description Creates a list of list of one or more elements. Inputs Element1 < any>: the value of the first element in the list. Element2 < any>: the value of subsequent items in the list, one per terminal.	Outputs	List $<$ list>: the new list
pack Element1 [Element2] pack List Description Creates a list of list of one or more elements. Inputs Element1 <any>: the value of the first element in the list. Element2 <any>: the value of subsequent items in the list, one per terminal.</any></any>	See also	pack
pack List Description Creates a list of list of one or more elements. Inputs Element1 <any>: the value of the first element in the list. Element2 <any>: the value of subsequent items in the list, one per terminal.</any></any>		Element1 [Element2]
pack List Description Creates a list of list of one or more elements. Inputs Element1 <any>: the value of the first element in the list. Element2 <any>: the value of subsequent items in the list, one per terminal.</any></any>		pack
pack List Description Creates a list of list of one or more elements. Inputs Element1 <any>: the value of the first element in the list. Element2 cany>: the value of subsequent items in the list, one per terminal. Outputt List</any>		
DescriptionCreates a list of list of one or more elements.InputsElement1 <any>: the value of the first element in the list.Element2 <any>: the value of subsequent items in the list, one per terminal.OutputeList dicts allow a king list</any></any>	pack	List
Element2 <any>: the value of subsequent items in the list, one per terminal.</any>	Description Inputs	Creates a list of list of one or more elements. Element1 <any>: the value of the first element in the list.</any>
		Element2 <any>: the value of subsequent items in the list, one per terminal.</any>
Cutputs List <inst>: the resulting list.</inst>	Outputs	List <list>: the resulting list.</list>

See also <u>make-list</u>, <u>unpack</u>
	InList
	reverse
reverse	OutList
Description Inputs Outputs	Creates a new list by reversing the order of items in an existing list. InList <list>: the existing list. OutList <list>: the new list.</list></list>
	InList Item N1 [N2]
set-nth	OutList
Description	Creates a new list by changing the value of a list item, specified by index, in an existing list. This primitive can be used to change values in nested lists by providing additional indices.
inputs	 Item <any>: the new value of the specified list item.</any> N1 <integer>: for a simple list, the index of the element to be modified; for a list of lists, the index of the nested list.</integer> N2 <integer>: the index of the element within a nested list or the index of another nested list. The final terminal must be the index of the element to be given a new value in the deepest nested list.</integer>
Outputs See also	OutList <list>: the resulting list. <u>get-nth</u>, <u>insert-nth</u>, <u>split-nth</u></list>
	InList Item N1 [N2]
set-nth!	List
Description	Changes the value of a list item, specified by index. This primitive can be used to change values in nested lists by providing additional indices.
Inputs	 Item <any>: the new value of the specified list item.</any> N1 <integer>: for a simple list, the index of the element to be modified; for a list of lists, the index of the nested list.</integer> N2 <integer>: the index of the element within a nested list or the index of another nested list. The final terminal must be the index of the element to be given a new value in the deepest nested list.</integer>
Outputs Side effects	List <list>: This primitive should be used with care, as it modifies its input data directly, rather than modifying copies of that input data. This can affect the results of other operations which independently</list>

	reference the same data. It may be important, therefore, to use synchros to ensure desired results.	
See also	get-nth, insert-nth, set-nth, split-nth	
	InList Duplicate? [AttributeName[CheckAllInstances]]	
	sort	
sort	OutList	
Description	Performs a simple sort on a list of numbers or strings, or sorts a list of instances on the value of a specified attribute.	
Inputs	InList <list>: the existing list.</list>	
	Duplicate <boolean>: if TRUE, items with identical values ar not duplicated in the sorted list. In the case of a list of instances if this parameter has a value of TRUE, instances with identical attribute values are not duplicated in the sorted list.</boolean>	
	AttributeName <string>: this parameter can only be provided when the InList parameter contains a list of instances. It specific the name of the attribute that the instances are to be sorted on.</string>	
	CheckAllInstances <boolean>: if all instances are of the same class, pass a value of TRUE with this parameter since it indicate that the specified attribute is at the same location in all instance</boolean>	
Outputs	OutList <list>: the sorted list.</list>	
split-nth	List N Split-nth Prefix Rest	
Description	Splits an existing list into two new lists; the first list consisting of the first N elements of the original list and the second list	
Inputs	List the existing list.	
	N <integer>: the number of items to take from the existing list t create the first of the two new lists.</integer>	
Outputs	Prefix <list>: the list consisting of the first N elements of the existing list.</list>	
	Rest <list>: the list consisting of the remaining elements of the existing list.</list>	
See also	get-nth, insert-nth, set-nth, set-nth!	

	0		
	unpack		
unpack	Element1 [Element2 ElementN]		
Description	Given N output roots, returns the first N elements of a list. The list must contain at least N elements.		
Inputs	List <list>: the original list.</list>		
Outputs	Element1 <any>: the first element in the list.</any>		
	Element2 ElementN <any>: elements 2 - N of the list.</any>		
Example			

List



See also

make-list, pack

Logical/Relational

The Logical/Relational primitives allow you to perform standard value comparisons. The following primitives are provided:

\leq	<u><=</u>	Ξ	<u>></u>	<u>>=</u>	
and	choc	ose equals	<u>gt</u>	gte	
<u>lt</u>	<u>lte</u>	not	or	xor	
	_				
				Item 1	ltem2
<				<	Ĭ×
Description	S	Succeeds if the firs	t parameter is less	s than the second	parameter.
Inputs	I	tem1 <string nur<="" td="" =""><td>nber></td><td></td><td>F</td></string>	nber>		F
	Ι	tem2 <string nur<="" td="" =""><td>nber></td><td></td><td></td></string>	nber>		
Outputs	< r	boolean>: an outporcessing of its re	out root can be add sult is required.	ed to this primiti	ve if further
Note	Î	Datatypes of the in	puts MUST matcl	1.	
Equivalent	1	<u>t</u>			
See also	<	<u><=, =, ≥, ≥=</u>			

	ltem1 ltem2
<=	
Description	Succeeds if the first parameter is less than or equal to the second parameter.
Inputs	Item1 <string number="" =""></string>
	Item2 <string number="" =""></string>
Outputs	<boolean>: an output root can be added to this primitive if further processing of its result is required.</boolean>
Note	Datatypes of the inputs MUST match.
Equivalent	lte
See also	\leq , \equiv , \geq , \geq \equiv
	ltem1_ltem2

=	
Description	If the two parameters are instances of classes, this primitive succeeds if both parameters are located at the same address. If the two parameters are external structures, this primitive succeeds if the value fields of the two parameters are equal. Otherwise, this primitive succeeds if the two parameters are equal, or in the case of lists, all corresponding components of the two lists are equal.
Inputs	Item1 <any></any>
	Item2 <any></any>
Outputs	<body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><b< td=""></b<></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body>
Equivalent	equals
See also	$\overline{\leq, \leq=,} \geq, \geq=$

>	Item1 Item2
Description	Succeeds if the first parameter is greater than the second parameter.
Inputs	Item1 <string number="" =""></string>
Outputs	<body> <td< td=""></td<></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body>
Note Equivalent See also	Datatypes of the inputs MUST match. \underline{gt} $\leq, \leq =, =, \geq =$

	Item1 Item2
>=	
Description	Succeeds if the first parameter is greater than or equal to the
Dooonphon	second parameter.
Inputs	Item1 <string number="" =""></string>
.	Item2 <string number="" =""></string>
Outputs	<boolean>: an output root can be added to this primitive if further processing of its result is required</boolean>
Note	Datatypes of the inputs MUST match.
Equivalent	gte
See also	<u>≤, ≤=, =, ≥</u>
	Boolean1 Boolean2 [Boolean3]
	and
and	
Description	Performs a logical AND on two or more boolean values.
Inputs	Boolean1 <boolean></boolean>
	Boolean2 <boolean></boolean>
Outputs	<pre><boolean>: an output root can be added to this primitive if further</boolean></pre>
	processing of its result is required.
See also	<u>or</u> , <u>xor</u> , <u>not</u>
	Criterion TrueChoice FalseChoice
	choose
choose	Choice
Description	Returns one of two values based on the value of a supplied criterion.
Inputs	Criterion <boolean>: the value that determines whether the TrueChoice parameter or FalseChoice parameter is returned by this primitive</boolean>
	TrueChoice <any>: the value to be returned if the Criterion parameter is TRUE.</any>
	FalseChoice <any>: the value to be returned if the Criterion parameter is FALSE.</any>
Outputs	Choice <any>:</any>

equals	Item1 Item2
Description	If the two parameters are instances of classes, this primitive succeeds if both parameters are located at the same address. If the two parameters are external structures, this primitive succeeds if the value fields of the two parameters are equal. Otherwise, this primitive succeeds if the two parameters are equal, or in the case of lists, all corresponding components of the two lists are equal.
Inputs	Item1 <any></any>
	Item2 <any></any>
Outputs	<boolean>: an output root can be added to this primitive if further processing of its result is required.</boolean>
Equivalent	<u>=</u>
See also	<u><, <=, ≥, ≥=</u>

gt	Item1 Item2
Description	Succeeds if the first parameter is greater than the second parameter.
Inputs	Item1 <string number="" =""></string>
	Item 2 <string number="" =""></string>
Outputs	<boolean>: an output root can be added to this primitive if further processing of its result is required.</boolean>
Note	Datatypes of the inputs MUST match.
Equivalent	<u>></u>
See also	<u> </u>

gte	Item1 Item2
Description	Succeeds if the first parameter is greater than or equal to the second parameter.
Inputs	Item1 <string number="" =""></string>
	Item2 <string number="" =""></string>
Outputs	<boolean>: an output root can be added to this primitive if further processing of its result is required.</boolean>
Note	Datatypes of the inputs MUST match.
Equivalent	<u>>=</u>
See also	$\leq, \leq =, =, \geq$

	Item1 Item2
lt	
Description	Succeeds if the first parameter is less than the second parameter
Inputs	Item1 <string number="" =""></string>
	Item2 <string number="" =""></string>
Outputs	<body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><body><td< td=""></td<></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body></body>
Note	Datatypes of the inputs MUST match.
Equivalent	\leq
See also	<u><=, =, ≥, ≥=</u>
	Item1 Item2
lte	ite
Description	Succeeds if the first peremeter is less than or equal to the second
Description	parameter.
Inputs	Item1 <string number="" =""></string>
	Item2 <string number="" =""></string>
Outputs	<boolean>: an output root can be added to this primitive if further processing of its result is required.</boolean>
Note	Datatypes of the inputs MUST match.

Equivalent

See also

<=

 \leq , \equiv , \geq , \geq =

not

Description Returns the logical negation of a boolean value. Inputs Boolean <boolean> **Outputs** Result <boolean> See also and, or, xor

Boolean 0 not ο Result

	Boolean1 Boolean2 [Boolean3]	
	or	
or		
Description	Performs a logical OR against two or more boolean values.	
Inputs	Boolean1 <boolean></boolean>	
	Boolean2 <boolean></boolean>	
Outputs	<pre>choolean> an output root can be added to this primitive if further</pre>	
Outputs	processing of its result is required.	
See also	and, xor, not	
	Boolean1 Boolean2	
	xor	
xor		
Description	Performs an XOR (exclusive OR) against two boolean values.	
Inputs	Boolean1 <boolean></boolean>	
Outpute	BOOLEANZ < DOOLEAN>	
Outputs	processing of its result is required.	
See also	and, <u>or</u> , <u>not</u>	
	Item1 Item2	
~=		
Description	Succeeds if the two parameters are not equal. This is the logical negation of the value that would be returned by the \equiv primitive.	
Inputs	Item1 <any></any>	
	Item2 <any></any>	
Outputs	<boolean>: an output root can be added to this primitive if further processing of the result is required.</boolean>	

Math

The following math primitives are provided:

*	**	<u>+</u>	<u>++</u>	<u>+1</u>
<u> </u>	<u></u>	<u>-1</u>	<u>abs</u>	acos
<u>asin</u>	<u>atan</u>	<u>cos</u>	<u>div</u>	idiv
max	<u>min</u>	<u>pi</u>	power	rand
rand-seed	round	round-down	round-up	<u>sin</u>
<u>sqrt</u>	<u>tan</u>	trunc		



	Integer1 Integer2 [Integer3]
	++
++	Sum
Description	Returns the sum of two or more integers.
Inputs	Integer1 <integer>:</integer>
	Integer2 <integer>:</integer>
	Integer3 <integer>:</integer>
Outputs	Sum <integer>:</integer>
See also	<u>+, -, div,, *, **, idiv, +1, -1</u>
	Number
+1	Result
Description	Adds 1 to a provided number.
Inputs	Number <number>:</number>
Outputs	Result <number>:</number>
See also	<u>+, -, div, ++,, *, **, idiv, -1</u>
	[Number1] Number2
-	Result
Description	Subtracts the second parameter from the first parameter or negates the value of a single provided parameter.
Inputs	Number1 <number>:</number>
	Number2 <number>:</number>
Default(s)	Number $1 = 0$
Outputs	Result <number>:</number>
See also	<u>+, div, ++,, *, **, idiv, +1, -1</u>

	[Integer1] Integer2
	Result
Description	Subtracts the second integer parameter from the first or negates the value of a single provided integer.
Inputs	Integer1 <integer>:</integer>
	Integer2 <integer>:</integer>
Default(s)	Integer3 <integer>: Integer1 = 0</integer>
Outputs	Result <integer>:</integer>
See also	<u>+, -, div, ++, *, **, idiv, +1, -1</u>
	Number

-1

Description	Subtracts 1 from a number.
Inputs	Number <number>:</number>
Outputs	Result <number>:</number>
See also	<u>+, -, div, ++,, *, **, idiv, +1</u>

Number	
🛛 abs 🕽	
Result	

o

o Result

abs

Description	Returns the absolute value of a number.
Inputs	Number <number>:</number>
Outputs	Result <number>:</number>

Cosine
acos 🕽
Angle

acos

Description	Angle is arccosine (Cosine) expressed in radians.
Inputs	Cosine <number>:</number>
Outputs	Angle <number>: the angle, in radians.</number>
See also	<u>sin, cos, tan, asin, atan</u>



asin

Description	Angle is arcsine (Sine) expressed in radians.
Inputs	Size <number>:</number>
Outputs See also	Angle <number>: the angle, in radians. acos, sin, cos, tan, atan</number>



atan

Description	Angle is arctangent (Tangent) expressed in radians.
Inputs	Tangent <number>:</number>
Outputs	Angle <number>: the angle, in radians.</number>
See also	acos, sin, cos, tan, asin



cos

Description	Cosine is cosine (Angle).
Outputs	Angle <number>: the angle, in radians.</number>
	Cosine <number>:</number>
See also	<u>acos, sin, tan, asin, atan</u>

	[Dividend] Divisor
div	Quotient
Description	If two parameters are provided, this primitive divides the first parameter by the second and returns the result. If a single parameter is provided, this primitive returns its reciprocal.
Inputs	Dividend <number>:</number>
	Divisor <number>:</number>
Delault(S)	Dividend = 1
Outputs	Quotient <number>:</number>
See also	$\pm, -, \pm\pm,, \pm, \pm, \pm\pm, \pm 101V, \pm 1, -1$

Outputs

Pi <real>:

	[Dividend] Divisor
idiv	Quotient [Remainder]
Description	Performs integer division. It returns the quotient and remainder resulting from dividing the first parameter by the second.
Inputs	Dividend <integer>:</integer>
	Divisor <integer>:</integer>
	Dividend = 1
Outputs	Remainder <integer></integer>
See also	<u>+, -, div, ++,, *, **, +1, -1</u>
	Number1 Number2 [Number3]
	max
may	Max
Description	Returns the maximum of two or more numbers.
inputs	Number 2 <number>:</number>
	Number3 <number>:</number>
Outputs	Max <number>:</number>
See also	<u>min</u>
	Number1 Number2 [Number3]
min	Min
Description	Returns the minimum of two or more numbers.
Inputs	Number1 <number>:</number>
	Number2 <number>:</number>
Outpute	Number3 <number>:</number>
See also	max max
рі	
Description	Returns the value of pi (3.1415926).

	Number Exponent
power	Result
Description	Calculates the value of a number to a provided exponent. If both parameters are zero (0), it returns 1.
Inputs	Number <number>: Exponent <number>:</number></number>
Outputs See also	Result <number>: sqrt</number>
	[rand]
rand	Randnum
Description	Generates a random integer between and including 0 and $(2^{31}) - 1$.
Outputs See also	RandNum <integer>: rand-seed</integer>
	Seed
rand-seed	rand-seed
Description	Sets the seed for the random-number generator to the integral part of Seed $(1 \text{ to } (2^{31}) - 1)$.
Inputs See also	Seed <number>: rand</number>
	Number [Precision]
round	Result
Description	Returns the number closest to a provided number, according to a specified precision. Positive and negative values for the precision parameter dictate the number of decimal places to the right and left of the decimal point, respectively. If the precision parameter is not provided, or is 0, the result is the integer closest to the provided number.
Inputs	Number <number>:</number>
	Precision <integer>:</integer>
Outputs Note	Result <number>: If two values are equally near to the provided number, the round primitive uses the Apple SANE library convention of rounding to</number>

the even value. For instance, 12.5 rounds to 12, and 13.5 rounds to 14.

Rounding to a given number of decimal places does not necessarily mean that the floating point representation of that number has that number of decimal places: it may have more. The the format primitive to store such numbers as strings with the desired number of decimal places.

Number [Precision]

See also

trunc, round-down, round-up

	round-down
round-down	Result
Description	Returns the nearest number less than or equal to the provided number according to the provided precision. Positive and negative values for the precision parameter dictate the number of decimal places to the right and left of the decimal point, respectively.
Inputs	Number <number>:</number>
	Precision <integer>:</integer>
Default(s)	Precision = 0 (return an integer).
Outputs	Result <number>:</number>
See also	trune, round, round-up
	Number [Precision]
	round-up
round-up	Result

round-up

Description	Returns the nearest number greater than or equal to a provided number, according to a specified precision. Positive and negative values for the precision parameter dictate the number of decimal places to the right and left of the decimal point, respectively. If the precision parameter is not given, or is 0, the round-up primitive returns the nearest integer greater than (or equal to) the provided
	number.
Inputs	Number <number>:</number>
	Precision <integer>:</integer>
Outputs	Result <number>:</number>
See also	trunc, round, round-down



sin	
Description	Sine is sine (Angle).
Outputs	Angle <number>: the angle, in radians.</number>
See also	Sine <number>: acos, cos, tan, asin, atan</number>



sqrt

Description	Returns the square root of a number.
Inputs	Number <number>:</number>
Outputs	SquareRoot <number>:</number>
See also	power



tan

Description	Tangent is tangent (Angle).
Outputs	Angle <number>: the angle, in radians.</number>
	Tangent <number>:</number>
See also	<u>acos, sin, cos, asin, atan</u>



trunc

Description	Returns the integer and fraction components of a provided number.
Inputs	Number <number>:</number>
Outputs	Integer <number>:</number>
	Fraction <real>:</real>
See also	round, round-down, round-up

Memory

The following primitives allow you to work directly with memory:

Contents Index Marten Primitives

address-to-object	Returns the Marten object at the specified address.
from-pointer	Casts a pointer as an integer.
get-integer	Returns an integer from a specified location.
<u>get-real</u>	Returns a real number of a specified length from a specified location.
<u>get-text</u>	Returns a string of a specified length from a specified location.
object-to-address	Returns the address of a Marten object.
put-integer	Places an integer into a specified memory location.
put-real	Places a real number into a specified memory location.
put-text	Places a string into a specified memory location.
string-address	Returns the address of the first character in a string.
to-pointer	Casts an integer as a pointer.



address-to-object

Description	Returns the Marten object at the specified address. The address must have been generated by a call to object-to-address on the same machine (same address space), and you must be certain that the object still exists at that address.
Inputs	Address <integer>:</integer>
Outputs	Object <any>:</any>
See also	object-to-address

	Pointer
	from-pointer
from-pointer	Address
Description	If the provided pointer is the address of a block of memory, then this primitive returns that address represented as an integer. In effect, it performs a cast of the pointer type to an integer.
Inputs	Pointer <external@>:</external@>
Outputs	Address <integer>:</integer>
Example	If Pointer = ABlock@16#00211A00, Address = 16#00211A00.
See also	to-pointer

	Buffer Offset Size
	get-integer
get-integer	Buffer NextOffset Value
Description	Returns an integer of a specified size from a particular location within a memory block.
Inputs	Buffer <ablock> <ablock@> <ablock@@> <integer>:</integer></ablock@@></ablock@></ablock>
	Offset <integer>: the offset, in bytes, at which to start reading the integer.</integer>
	Size <integer>: the size, in bytes, of the integer. Acceptable values are:</integer>
	□ 1
_	• 4
Outputs	Buffer <ablock> <ablock@> <ablock@@> <integer>:</integer></ablock@@></ablock@></ablock>
	NextOffset <integer>: the value of the Offset parameter added to the value of the Size parameter</integer>
	Value <integer>: the integer read.</integer>
See also	put-integer, get-real, get-text
	Buffer Offset Size
	Buffer Offset Size
	Buffer Offset Size
get-real	Buffer Offset Size get-real Buffer NextOffset Value
get-real Description	Buffer Offset Size get-real Buffer NextOffset Value Returns a real number of a specified length from a specified location within a memory block.
get-real Description Inputs	Buffer Offset Size get-real Buffer NextOffset Value Returns a real number of a specified length from a specified location within a memory block. Buffer <ablock> <ablock@> <ablock@ @=""> <integer>:</integer></ablock@></ablock@></ablock>
get-real Description Inputs	Buffer Offset Size get-real Buffer NextOffset Value Returns a real number of a specified length from a specified location within a memory block. Buffer <ablock> <ablock@> <ablock@@> <integer>: Offset <integer>: the position within the memory block at which to begin reading the real number.</integer></integer></ablock@@></ablock@></ablock>
get-real Description Inputs	Buffer Offset Sizeget-realBuffer NextOffset ValueReturns a real number of a specified length from a specified location within a memory block.Buffer <ablock> <ablock@> <ablock@@> <integer>:Offset <integer>: the position within the memory block at which to begin reading the real number.Size <integer>: the size, in bytes, (4, 8, or 10) of the real number to be read. Acceptable values are:</integer></integer></integer></ablock@@></ablock@></ablock>
get-real Description Inputs	Buffer Offset Size get-real Buffer NextOffset Value Returns a real number of a specified length from a specified location within a memory block. Buffer <ablock> <ablock@> <ablock@@> <integer>: Offset <integer>: the position within the memory block at which to begin reading the real number. Size <integer>: the size, in bytes, (4, 8, or 10) of the real number to be read. Acceptable values are: 4</integer></integer></integer></ablock@@></ablock@></ablock>
get-real Description Inputs	Buffer Offset Size get-real Buffer NextOffset Value Returns a real number of a specified length from a specified location within a memory block. Buffer <ablock> <ablock@> <ablock@@> <integer>: Offset <integer>: the position within the memory block at which to begin reading the real number. Size <integer>: the size, in bytes, (4, 8, or 10) of the real number to be read. Acceptable values are: a a b</integer></integer></integer></ablock@@></ablock@></ablock>
get-real Description Inputs	Buffer Offset Size get-real Buffer NextOffset Value Returns a real number of a specified length from a specified location within a memory block. Buffer <ablock> <ablock@> <ablock@@> <integer>: Offset <integer>: the position within the memory block at which to begin reading the real number. Size <integer>: the size, in bytes, (4, 8, or 10) of the real number to be read. Acceptable values are: 4 8 10</integer></integer></integer></ablock@@></ablock@></ablock>
get-real Description Inputs Outputs	Buffer Offset Size get-real Buffer NextOffset Value Returns a real number of a specified length from a specified location within a memory block. Buffer <ablock> <ablock@> <ablock@@> <integer>: Offset <integer>: the position within the memory block at which to begin reading the real number. Size <integer>: the size, in bytes, (4, 8, or 10) of the real number to be read. Acceptable values are: a a b b c d b c d e d a b c d a b c c d c d d d d d d d d d d d d d d d d d d</integer></integer></integer></ablock@@></ablock@></ablock>
get-real Description Inputs Outputs	Buffer Offset Size get-real Buffer NextOffset Value Returns a real number of a specified length from a specified location within a memory block. Buffer <ablock> <ablock@> <ablock@@> <integer>: Offset <integer>: the position within the memory block at which to begin reading the real number. Size <integer>: the size, in bytes, (4, 8, or 10) of the real number to be read. Acceptable values are: 4 8 10 Buffer <ablock> <ablock@> <ablock@@> <integer>: NextOffset <integer>: the value of the Offset parameter plus that of the Size parameter.</integer></integer></ablock@@></ablock@></ablock></integer></integer></integer></ablock@@></ablock@></ablock>
get-real Description Inputs Outputs	Buffer Offset Size get-real Buffer NextOffset Value Returns a real number of a specified length from a specified location within a memory block. Buffer <ablock> <ablock@> <ablock@@> <integer>: Offset <integer>: the position within the memory block at which to begin reading the real number. Size <integer>: the size, in bytes, (4, 8, or 10) of the real number to be read. Acceptable values are: 4 8 10 Buffer <ablock> <ablock@> <ablock@@> <integer>: NextOffset <integer>: the value of the Offset parameter plus that of the Size parameter. Value <real>: the real number read.</real></integer></integer></ablock@@></ablock@></ablock></integer></integer></integer></ablock@@></ablock@></ablock>

	Buffer Offset Size
	get-text
get-text	Buffer NextOffset String
Description	Returns a string of a specified size from a particular location within a memory block.
Inputs	Buffer <ablock> <ablock@> <ablock@@> <integer>:</integer></ablock@@></ablock@></ablock>
	Offset <integer>: the offset within the memory block at which to start reading.</integer>
	Size <integer>: must be greater than or equal to 0 and less than or equal to 65535 when using the interpreter and can be up to 4294967295 bytes when using the compiler).</integer>
Outputs	Buffer <ablock> <ablock@> <ablock@@> <integer>:</integer></ablock@@></ablock@></ablock>
	NextOffset <integer>: the value of the Offset parameter plus that of the Size parameter.</integer>
	String <string>:</string>
See also	put-text, get-integer, get-real



object-to-address

Description	Returns the address of any Marten object.
Inputs	Object <any>:</any>
Outputs	Address <integer>:</integer>
See also	address-to-object



put-integer

Description	Converts a provided integer into an integer of a specified size and
	places it at a specified location in a memory block.
Inputs	Buffer <ablock> <ablock@> <ablock@@> <integer>:</integer></ablock@@></ablock@></ablock>
	Offset <integer>: the offset within the memory block at which to</integer>
	write the integer.

Size <integer>: the size, in bytes, of the integer. Acceptable values are:

Outputs

See also

	1 2 4
Va	ue <integer>:</integer>
Bu	ffer <ablock> <ablock@> <ablock@@> <integer>:</integer></ablock@@></ablock@></ablock>
Ne val get	tOffset <integer>: the value of the Offset parameter plus the ue of the Size parameter. -integer, put-real, put-text</integer>
	Buffer Offset Size Value

	0 0
put-real	Buffer NextOffset
Description	Converts a real into a floating point value of a specified size and places it in a particular location in a memory block.
Inputs	Buffer <ablock> <ablock@> <ablock@@> <integer>:</integer></ablock@@></ablock@></ablock>
	Offset <integer>: the position in the memory block at which to write the floating point value.</integer>
	Size <integer>: the size, in bytes, of the floating point value. Acceptable values are:</integer>
	• 4
	□ 8
	1 0
	Value <real> or <integer>:</integer></real>
Outputs	Buffer <ablock> <ablock@> <ablock@@> <integer>:</integer></ablock@@></ablock@></ablock>
	NextOffset <integer>: the value of the Offset parameter plus the value of the Size parameter.</integer>
See also	get-real, put-integer, put-text

	Buffer Offset Size String
put-text	Buffer NextOffset
Description	Writes a specified number of bytes from a string to a particular position in a memory block.
Inputs	Buffer <ablock> <ablock@> <ablock@@> <integer>:</integer></ablock@@></ablock@></ablock>
	Offset <integer>:</integer>
	Size <integer>: must be greater than or equal to 0 and less than or equal to 65535 in the interpreter or 4294967295 bytes in the compiler.</integer>
	String <string>:</string>
Outputs	Buffer <ablock> <ablock@> <ablock@@> <integer>:</integer></ablock@@></ablock@></ablock>
	NextOffset <integer>: the value of the Offset parameter plus that of the Size parameter.</integer>
See also	get-text, put-integer, put-real
string-address	string-address Address
Description	Returns the address of the first character in a string. The string should be locked whenever an address is taken or used.
Inputs	aString <string>:</string>
Outputs	Address <ablock@>:</ablock@>
	Address
	to-pointer
to-pointer	Pointer
Description	If Address is the integer address of a block in memory, then Pointer is that same address represented as a generic pointer type in Marten. In effect, it performs a cast of the integer type to a pointer.
Inputs	Address <integer>:</integer>
Outputs	Pointer <ablock@>:</ablock@>

ExampleExample If Address = 16#00211A00, Pointer =
ABlock@16#00211A00.See alsofrom-pointer

String

The following primitives are provided for working with strings:

Contents Index Marten Primitives

from-ascii

byte-length	Returns the length of a string in bytes
from-ascii	Returns the character represented by a specified ASCII code.
from-string	Returns the textual value of a provided string.
<u>"in"</u>	Returns the location of a substring within a provided string.
integer-to-string	Returns the string representation of an integer.
<u>"join"</u>	Concatenates two or more strings.
<u>"length"</u>	Returns the number of characters in a string.
middle	Returns a substring of a specified string.
<u>prefix</u>	Returns two strings, the first <i>n</i> characters in one string and the remaining characters in the second string.
string-to-integer	Returns the integer representation of a provided string.
<u>suffix</u>	Returns two substrings, the last <i>n</i> characters in one string and the remaining characters in the second string.
<u>to-ascii</u>	Returns the ASCII integer representation of a provided string.
to-string	Returns the string representation of a provided Marten object.

hyte-length	String byte-length Length
byte-tength	
Description	Returns the length of a specified string in bytes rather than characters. Use this primitive to return the length of a specified string in bytes.
Inputs	String <string>:</string>
Outputs	Length <integer>: length of the string, in bytes.</integer>

	CharCodes
	from-ascii
	String
Returns the character representation	n of one or more ASCII codes.

Description	Returns the character representation of one or more ASCII codes.
Inputs	CharCodes <integer integers="" list="" of="" ="">: charCode or charCodeList.</integer>
Outputs	String <string>: a string containing the characters represented by the provided ASCII codes.</string>
Example	If CharCodes = (72 101 108 108 111), then String = "Hello". If CharCodes = 72, String = "H".
See also	from-string, to-ascii, to-string

	Address
	from-string
from-string	Data
Description	Returns the value textually represented by String . Type cannot be a class or External structure.
Note	In producing output, this primitive follows Marten rules for unparsing. For details on data types, refer to the <i>Marten User Guide</i> .
Inputs	String <string>:</string>
Outputs	Data <any*> Point Rect RGBType</any*>
See also	<u>from-ascii, to-ascii, to-string</u>
"in"	String SubString [StartIndex]
Description	Returns the location of a substring in a provided string.
Inputs	String <string>: the string in which to search.</string>
	SubString <string>: the substring to search for.</string>
	StartIndex <integer>: the index of the character in the String parameter at which to start the search. This parameter must be a positive number.</integer>
Default(s)	StartIndex = 1
Outputs	FoundIndex <integer>: if the substring was found, the index in the string of the first character of the substring; zero (0) otherwise.</integer>
Note	If SubString is an empty string, FoundIndex returns 1.
See also	<u>"length"</u> , <u>"join"</u>



integer-to-string

Description	Returns the four character string representation of an integer.
Inputs	Integer <integer>:</integer>
Outputs	String <string>:</string>
Example	ExampleIf Integer = 16#54455854 ("TEXT"), then String = "TEXT".
See also	string-to-integer

	String1 String2 [String3]
"join"	Result
Description Inputs	Concatenates two or more strings. String1 <string>: String2 <string>: String3 <string>:</string></string></string>
Outputs See also	Result <string>: <u>"in"</u>, <u>"length"</u></string>
	String
"length"	Length
Description Inputs	Returns the number of characters in a string. String <string>:</string>
Outputs See also	Length <integer>: <u>"in"</u>, <u>"join"</u></integer>
	String N Index
middle	OutString
Description	Returns a substring of a specified number of characters from a provided string beginning at a specified position in the string.
Inputs	String <string>:</string>
	N <integer>: the number of characters to return.</integer>
	Index <integer>: the index within the String parameter corresponding to the first character of the substring.</integer>
Outputs See also	OutString <string> prefix, suffix, "in"</string>

	Instring N
prefix	OutString Suffix
Description	Returns two substrings; the first consisting of the first N characters of the string and the second consisting of the remaining characters
Inputs	InString <string>:</string>
	N <integer>:</integer>
Outputs	OutString <string>:</string>
Example	Suffix <string>: The following example extracts the first and last names; the space character separates the two names. The second root of the prefix primitive returns the last name, while suffix is used to strip the trailing blank from the first name.</string>



string-to-integer

Integer **Description** Returns the integer representation of a four character string. Inputs **String** <string>: **Outputs Integer** <integer>: Example ExampleIf **String** = "TEXT", then **Integer** = 16#54455854 ("TEXT"). See also integer-to-string

54

String 0

suffix

to-string

Inputs

Outputs

Description



String

Description	Returns two substrings, one consisting of the last N characters of the provided string, the other consisting of the remaining characters.
Inputs	InString <string>:</string>
	N <integer>:</integer>
Outputs	Prefix <string>: the initial characters of the Instring parameter; those not included in the Suffix parameter.</string>
See also	Suffix <string>: the last N characters of the Instring parameter. <u>middle</u>, <u>prefix</u>, <u>"in"</u></string>

	to-ascii
to-ascii	CharCodeList
Description	Returns the list of integers that are the ASCII representations of the characters in a provided string.
Inputs	String <string>:</string>
Outputs	CharCodeList <list>:</list>
Example	ExampleIf String = "Hello", then charCodeList = (72 101 108 108 111).
See also	from-ascii, from-string, to-string



	This parameter has a value of NULL if the provided data cannot
	be represented textually; instances and structures other than Point,
	Rect, and RGBType.
Note	This primitive follows Marten's unparsing rules for working with
	datatypes. For details, refer to the Marten User Guide.

See also from-ascii, from-string, to-ascii

System

The following system primitives are provided:

Contents Index Marten Primitives

ancestors	Returns the names of all ancestors of a provided class.
<u>attributes</u>	Returns the names of all attributes of a given class.
<u>children</u>	Returns the names of all immediate subclasses of a given class.
<u>classes</u>	Returns the names of all classes in a project.
descendants	Returns the names of all subclasses of a given class.
<u>methods</u>	Returns a list of methods of a given type.
persistents	Returns the list of all persistents in a project.



ancestors

Description	Returns the names of all ancestor classes of a class specified by Instance or ClassName.
Inputs	Class <instance> <string>: the class name or an instance of the class.</string></instance>
Outputs	Ancestors <list>: the list of names of ancestor classes. The list is ordered such that the first name in the list is the immediate superclass of the input class. If the input class is a top-level class, the list is empty.</list>
See also	children, descendants

C1	ass O
attri 📃	butes
ClassAttrs	o Instance Attrs

attributes	ClassAttrs InstanceAttrs
Description	Returns the names of all instance attributess and class attributes of a specified class.
Inputs	Class <instance> <string>: the name of the class or an instance of the class for which class and instance attribute names are to be returned.</string></instance>
Outputs	ClassAttrs <list;>: a list of the names of class attributes in the specified class.</list;>
0	InstanceAttrs <list>: a list of the names of instance attributes in the specified class.</list>
See also	classes, methods, persistents





descendants

Description	Return the names of all classes that are descendants of a specified class.
Inputs	Class <instance> <string>: the name of the class or an instance of the class for which the names of descendant classes are to be returned.</string></instance>
Outputs	Descendants <list of="" string="">: a sorted list of the names of all classes that are descendants of the specified class.</list>
Note	Contrast the purpose of this primitive with that of the children primitive.
See also	ancestors, children

methods Description



Persistents Persistents

	Methods is the list of names of Universal methods, and GetMethods and SetMethods are both ().
Inputs	MethodType <instance> <string> none</string></instance>
Outputs	Methods <list>:</list>
	GetMethods <list>:</list>
	SetMethods <list>:</list>
Compiler	No distinction is made between Universal methods and primitives. If no input parameter is provided, this primitive returns a list of names of Universal methods and primitives.
See also	attributes, classes, persistents



Description	Returns the names of all persistents in the project.
Outputs	Persistents <list of="" strings="">: a list of the names of all persistents.</list>
See also	attributes, classes, methods

Туре

The following type primitives are provided:

boolean?	Succeeds if the provided data is boolean.
external-type	Returns the name of a referenced External structure.
instance?	Succeeds if the provided data is an instance.
integer?	Succeeds if the provided data is an integer.
list?	Succeeds if the provided data is a list.
number?	Succeeds if the provided data is an integer to a real number.
real?	Succeeds if the provided data is a real number.
string?	Succeeds if the provided data is string.
<u>type</u>	Returns the type of a provided Marten object.



boolean?

Description	Succeeds if the provided data is boolean, (TRUE or FALSE).
Inputs	Item <any>: any Marten data item.</any>
Outputs	boolean
See also	instance?, integer?, list?, external-type, number?, real?, string?, type



external-type

Description	StructName is the name of the external type whose structure, pointer, or handle is referred to by Data. A list of possible external type names can be found in the Info window under External
	Structures.
Inputs	Data <external@@> <external@> <external>:</external></external@></external@@>
Outpute	Stars of Nione of Astrice and a

Outputs StructName <string>:

See also <u>boolean?</u>, <u>instance?</u>, <u>integer?</u>, <u>list?</u>, <u>number?</u>, <u>real?</u>, <u>string?</u>, <u>type</u>



instance?

Description	Succeeds if the provided data is an instance of a class.
Inputs	Item <any>: any Marten data item.</any>
Outputs	boolean
See also	<u>boolean?</u> , <u>integer?</u> , <u>list?</u> , <u>external-type</u> , <u>number?</u> , <u>real?</u> , <u>string?</u> , type



integer?

Description	Succeeds if the provided data is an integer.
Inputs	Item <any>: any Marten data item.</any>
Outputs	boolean
See also	boolean?, instance?, list?, external-type, number?, real?, string?, type



list?

Description	Succeeds if the provided data is a list.
Inputs	Item <any>: any Marten data item.</any>
Outputs	boolean
See also	<u>boolean?</u> , <u>instance?</u> , <u>integer?</u> , <u>external-type</u> , <u>number?</u> , <u>real?</u> , <u>string?</u> , <u>type</u>





Item o real?

Description	Succeeds if the provided data is areal or an integer.
Inputs	Item <any>: any Marten data item.</any>
Outputs	boolean
See also	boolean?, instance?, integer?, list?, external-type, real?, string? type

real?

Description Inputs	Succeeds if the provided data is a real number. Item <any>: any Marten data item.</any>
Outputs	boolean
See also	boolean?, instance?, integer?, list?, external-type, number?, string?, type



string?

Description	Succeeds if the provided data is a string.
Inputs	Item <any>: any Marten data item.</any>
Outputs	boolean
See also	boolean?, instance?, integer?, list?, external-type, number?, real?, type



type	
------	--

Description	Returns the type of the provided Marten data.
Inputs	Item <any>: any Marten data item.</any>
Outputs	Type <string>: one of the following: boolean, integer, list, external, none, null, real, string or undefined.</string>
See also	boolean?, instance?, integer?, list?, external-type, number?, real?, string?

Index

Symbols

"in" primitive 60 "join" primitive 61 "length" primitive <u>61</u> (in) primitive 32 (join) primitive 33 (length) primitive 33 * primitive 46 ** primitive <u>46</u> + primitive 46 ++ primitive 47 +1 primitive 47 < primitive> 40, 44 <= primitive> <u>41</u>, <u>44</u> = primitive <u>41</u>, <u>43</u> > primitive 41, 43 >= primitive **42**, **43**

Numerics

-1 primitive <u>48</u>, <u>48</u> 32-bit integers <u>13</u>

Α

abs primitive <u>48</u> acos primitive <u>48</u> addresses converting to pointers <u>58</u> obtaining from pointers <u>54</u> of instance <u>56</u> of strings <u>58</u> address-to-object primitive <u>54</u> ancestors primitive 64 and primitive 42 AND, bitwise 13 angles primitives for use with 45 answer primitive 29 answer-v primitive 29 arccosine 48 arcsine 49 arctangent 49 ASCII converting characters from 59 representation of strings 63 asin primitive 49 ask primitive 30 atan primitive 49 attach-l primitive 33 attach-r primitive 34 attribute providing values programmatically 21 attributes list of instance values 21 list of, in class 64 attributes primitive 64

В

binary search <u>36</u> bit arithmetic primitives <u>13</u> Bit primitives library containing <u>5</u> bit-and primitive <u>13</u>

bit-not primitive <u>14</u> bit-or primitive <u>14</u> bit-shift-l primitive <u>14</u> bit-shift-r primitive <u>15</u> bit-testing <u>15</u> bit-xor primitive <u>15</u> boolean? primitive <u>67</u> booleans testing data for <u>67, 69</u> byte-length primitive <u>59</u>

С

Callbacks primitives library containing 5 children primitive 65 classes list of ancestors 64 list of descendants 65 list of immediate children 65 list of methods in 66 list of, in project 65 classes primitive 65 close-file primitive 22 complement, bitwise 14 copy primitive 20 cos primitive <u>49</u> create-object-file primitive 23 create-text-file primitive 23

D

Data primitives library containing <u>5</u> decrement <u>48</u> delete-file primitive <u>23</u> descendants primitive <u>65</u> detach-l primitive <u>34</u> detach-nth primitive <u>34</u> detach-r primitive <u>35</u> dialog boxes displaying programmatically <u>30</u> opening programmatically <u>30</u> div primitive <u>49</u>

Ε

equals primitive <u>41</u>, <u>43</u> exclusive or primitive <u>45</u> EXCLUSIVE-OR operation, integers <u>15</u> exponents <u>51</u> externals testing data for <u>69</u> external-type primitive <u>67</u>

F

File primitives library containing 5 files buffering, External Block 25 closing 22 creating 23 creating. text 23 deleting 23 open dialog 23 opening 24 primitives for working with 22 reading objects from 25 reading text from 25 save dialog 24 writing objects to 25 writing text to 26 find-instance primitive 35 find-sorted primitive 36 floating point writing to memory 57 folders Library 6 from-ascii primitive 59 from-pointer primitive 54 from-string primitive 60

G

get-file primitive <u>23</u> get-integer primitive <u>55</u> get-nth primitive <u>11</u>, <u>36</u> get-real primitive <u>55</u> get-text primitive <u>56</u>

Graphics primitives <u>26</u> library containing <u>5</u> greater than primitive <u>41</u>, <u>43</u> greater than/equal to primitive <u>42</u>, <u>43</u> gt primitive <u>43</u> gte primitive <u>43</u>

idiv primitive 50 increment 47 Input/Output primitives library containing 5 inputs prompting programmatically 30 insert-nth primitive 37 instance attributes list of, in class 64 instance? primitive 67 instances copying 20 creating from list 21 memory address of 56 obtaining by address 54 returning class name/attributes 21 searching in lists 35 testing data for 67 inst-to-list primitive 21 integer? primitive 67 integers addition 47 bit arithmetic 13 bitwise complement 14 bitwise OR 14 division 50 EXCLUSIVE_OR 15 from reals 53 multiplication 46 obtaining from memory 55 representation of string 62 shift left 14 shifting right 15 string representation 60 subtraction 48

testing against mask <u>15</u> testing bit positions <u>15</u> testing bits against mask <u>16</u> testing data for <u>67</u>, <u>69</u> writing to memory <u>56</u> integers, AND operation <u>13</u> integer-to-string primitive <u>60</u> Interpreter control primitives library containing <u>5</u>

L

left shift, integers 14 less than primitive <u>40</u>, <u>44</u> less than/equal to primitive <u>41</u>, <u>44</u> libraries content 5 introduced 5 loading 5 location 5 packaged with Marten 5 primitive categories contained 5 Library folder 6 List primitives library containing 5 list? primitive 68 lists accessing elements by index 11, 36 accessing first elements of 40 binary search in 36 changing values in 38, 38 concatenating elements to end 34 concatenating elements to start 33 copying 20 creating/populating programmatically 37, 37 inserting elements 37 joining two 33 returning first n elements 34 returning last n elements 35 returning length 33 returning nth element 34 reversing 38 searching elements by instance 35

searching for item in <u>32</u> sorting <u>39</u> splitting into two <u>39</u> testing data for <u>68</u>, <u>69</u> list-to-inst primitive <u>21</u> list-to-point primitive <u>27</u> list-to-rect primitive <u>27</u> list-to-RGB primitive <u>27</u> logarithm primitives <u>45</u> Logical/Relational primitives library containing <u>5</u> lt primitive <u>44</u> lte primitive <u>44</u>

Μ

make-list primitive <u>37</u> mask, testing integers against 15 Math primitives library containing 5 max primitive 50 memory address to pointer conversion 58 locating instances by address 54 obtaining integers from 55 obtaining reals from 55 pointer/address conversion 54 returning text from 56 string address 58 writing integers to 56 writing real/floating point to 57 writing text to 58 Memory primitives library containing 5 methods list of, in a class 66 methods primitive 66 middle primitive 61 min primitive 50, 50 multiplication primitives 45

Ν

names

class, from instance **21** NONE testing data for 69 Not operation, bits 14 not primitive 44 NULL testing data for 69 number? primitive 68 numbers absolute value 48 addition 46 division 49 incrementing/decrementing 45 max/min primitives 45 rounding primitives 45 sorting in lists 39 square root 53 subtracting 47 testing data for 68

0

objects memory address of <u>56</u> reading from file <u>25</u> writing to file <u>25</u> Open File Navigation Dialog <u>23</u> open-file primitive <u>24</u> OR operation, bitwise <u>14</u> or primitive <u>45</u> outputs displaying programmatically <u>30</u>

Ρ

pack primitive <u>37</u> persistents list of, in project <u>66</u> persistents primitive <u>66</u> pi primitive <u>50</u> pointers obtaining address from <u>54</u> obtaining from addresses <u>58</u> Points creating from list <u>27</u>
ABCDEFGHIJKLMNOPQRSTUVWXYZ

primitives for working with 26 returning co-ordinates 28 point-to-list primitive 28 power primitive 51 prefix primitive 62 primitives bit **13** categories in libraries 5 data 20 file 22 graphics 26 libraries containing 5 projects list of classes in 65 list of persistents in $\underline{66}$ loading libraries 5 prompts, programmatic 30 put-file primitive 24 put-integer primitive 56 put-real primitive 57 put-text primitive 58

R

radians primitives that return 45 rand primitive 51 random numbers generating 51 seeding 51 rand-seed primitive 51 read-buffer primitive 25 read-object primitive 25 read-text primitive 25 real? primitive 68 reals obtaining from memory 55 testing data for 68, 69 truncating 53 writing to memory 57 Rects creating from list 27 primitives for working with 26 returning co-ordinates 28

rect-to-list primitive <u>28</u> Returns <u>54</u> reverse primitive <u>38</u> RGBs creating from list <u>27</u> primitives for working with <u>26</u> returning colour specifiers <u>28</u> RGB-to-list primitive <u>28</u> right shift, integers <u>15</u> round primitive <u>51</u> round-down primitive <u>9, 52</u> round-up primitive <u>52</u>

S

Save Location Navigation Dialog 24 search, binary 36 select primitive 30 set-nth primitive 38 set-nth! primitive 38 shallow copy primitive 21 shift left, integer 14 shift right, integers 15 show primitive 30 sin primitive 53 sort primitive 39 split-nth primitive 39 sqrt primitive 53 String primitives library containing 5 string? primitive 68 strings address of 58 ASCII representation of 63 concatenating 61 from ASCII equivalent 59 integer representation of 62 length in bytes 59 length of 61 location of substring in 60 obtaining from memory 56 representation of integer 60 returning characters from 61 returning first characters of 62

ABCDEFGHIJKLMNOPQRSTUVWXYZ

returning last characters of 63 sorting in lists 39 testing data for 68, 69 text representation of 63 textual value 60 string-to-integer primitive 62 subclasses list of by class 65 list of, by class 65 subtraction primitives 45 suffix primitive 63 System primitives library containing 5

Т

tan primitive <u>53</u> test-all? primitive <u>15</u> test-bit? primitive <u>15</u> test-one? primitive <u>16</u> text reading from file <u>25</u> representation of string <u>60</u>, <u>63</u> writing to file <u>26</u> writing to memory <u>58</u> to-ascii primitive <u>63</u> to-pointer primitive <u>58</u> to-string primitive <u>63</u> trunc primitive <u>53</u> type primitive <u>69</u> Type primitives library containing <u>5</u> types testing data for <u>69</u> testing externals <u>67</u>

U

undefined types testing data for <u>69</u> unpack primitive <u>40</u> use counts and copying <u>20</u>

V

value, absolute 48

W

write-object primitive <u>25</u> write-text primitive <u>26</u>

X

xor primitive 45