MARTEN

Primitives Reference

Contents

Index

Revision 1.0

Dec. 29, 2005

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.

Andescotia LLC Contact Information

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

Contents

Chapter 1

Working with libraries

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.

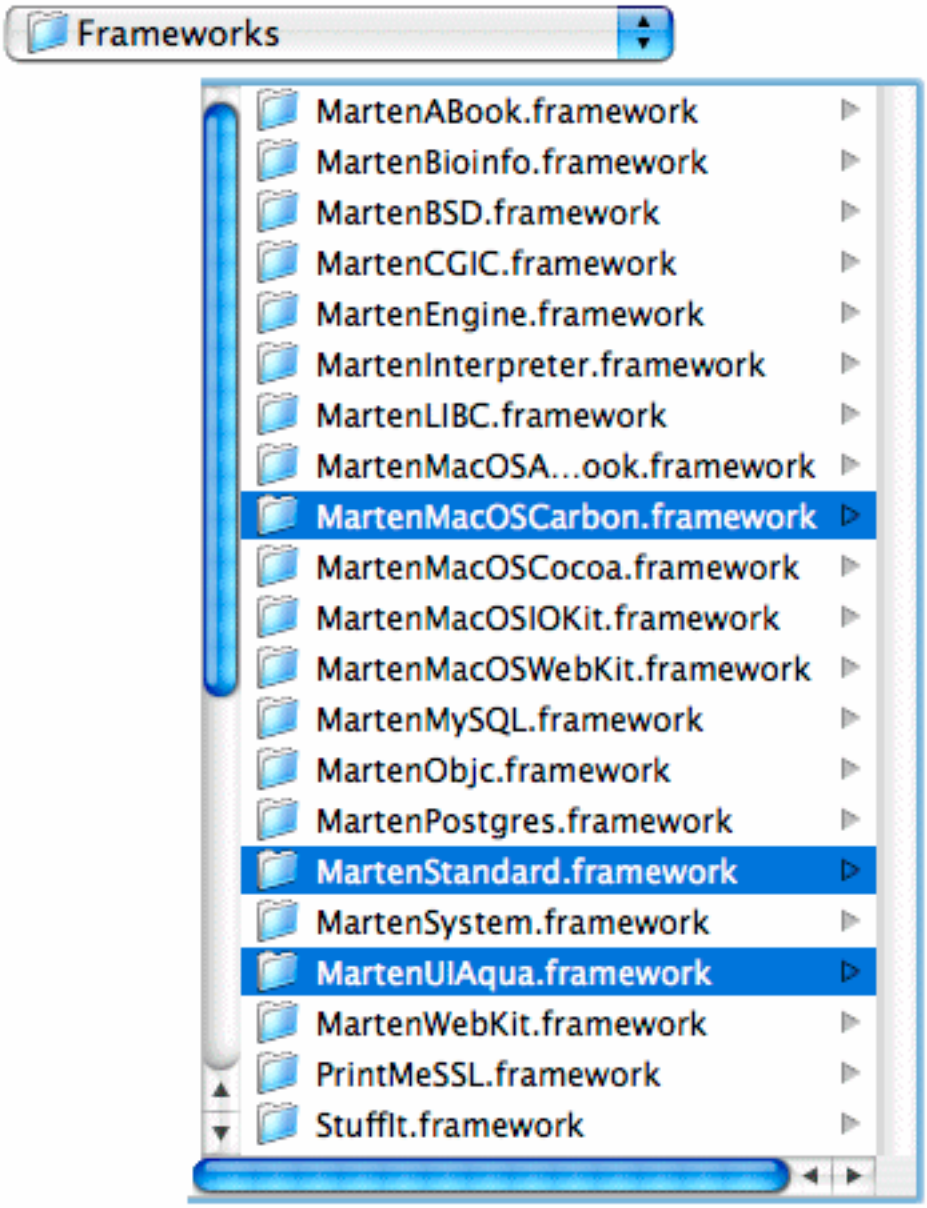
For information on contents of the standard libraries delivered with Marten, see "Library contents" on page 1.

To load a library into your project:

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

   A **Choose Object** navigation dialog opens.

2. 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**.

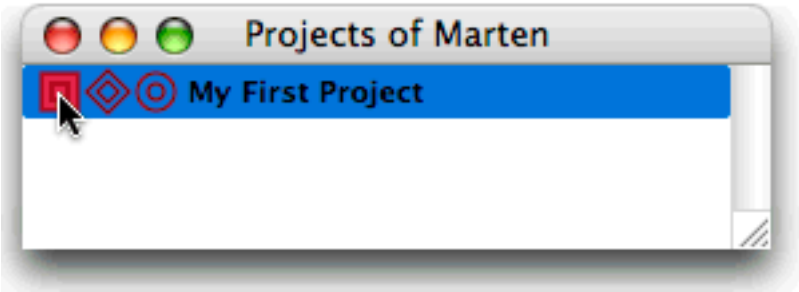
Once you save a project to which you have added a library, that library is subsequently added to the project automatically whenever you load the project into Marten.

Viewing the libraries in a project

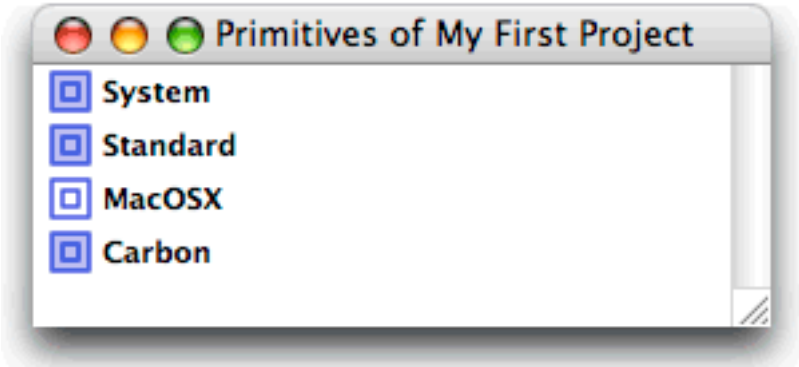
You can view the libraries loaded into a project and inspect their contents.

➢ **To view the libraries currently loaded into a project:**

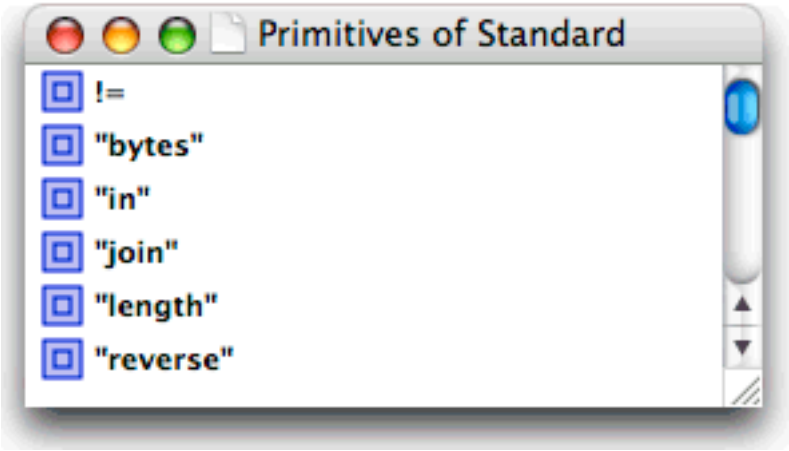
1. Double-click the primitives icon of a project item in a Projects window.



A **Primitives of** *Project Name* window opens.



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


Primitives of Standard
!=
"bytes"
"in"
"join"
"length"
"reverse"

Chapter 2

Marten Primitives

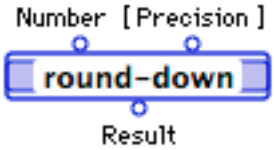
This chapter tells you everything you have to know about the built-in Marten primitives. It provides details on how primitives are documented in this chapter, provides a categorized listing of all available primitives, and includes a detailed description of each primitive.

Primitive documentation conventions

Primitive descriptions in this chapter have two components:

- Syntax description - a graphic depiction of the primitive operation that provides a syntax diagram
- Categorized primitive information - 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

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>: **Precision** <integer>:
Default(s)	**Precision** = 0 (return an integer).
Outputs	**Result** <number>:
See also	trunc, round, round-up

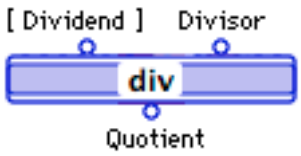
Syntax description

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 [ ].

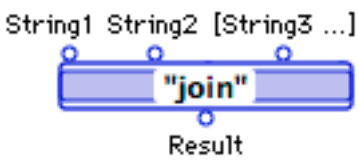
For example, the **div** primitive has an optional **Dividend** input.



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

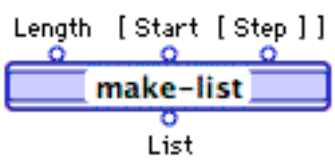
Primitives with a variable number of inputs

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

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

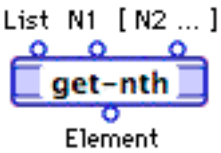
Certain primitives perform tests or manipulate boolean input values to obtain a boolean result. Every such boolean primitive can either have one root, in which case it returns TRUE or FALSE, or no roots, in which case it succeeds or fails. Use an output root on a boolean primitive if you need to do further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



get-nth

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.

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.

Outputs **Element** <any>:

See also insert-nth, set-nth, set-nth!, split-nth

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:

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

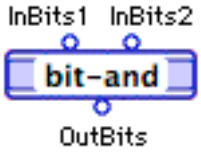
- System
- Type

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.
bit-or	The output is the bitwise OR of two integers.
bit-shift-l	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).
bit-shift-r	The output is a bitwise shift right with sign extension by one or more places on an integer.
bit-xor	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).

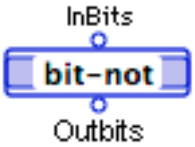
Inputs Returns the bitwise AND of two integers.

Inputs **InBits1** <integer>:

InBits2 <integer>:

Outputs **OutBits** <integer>:

See also bit-not, bit-or, bit-shift-l, bit-shift-r, bit-xor



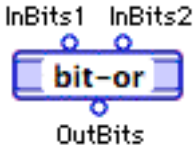
bit-not

Description The bitwise COMPLEMENT of an integer is output.

Inputs **InBits** <integer>:

Outputs **OutBits** <integer>:

See also bit-and, bit-or, bit-shift-l, bit-shift-r, bit-xor



bit-or

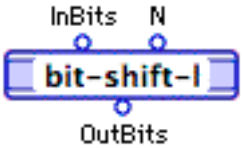
Description The bitwise OR of two integers is output. For example the result for 5 (b101) and 3 (b11) is 7 (b111).

Inputs **InBits1** <integer>:

InBits2 <integer>:

Outputs **OutBits** <integer>:

See also bit-and, bit-not, bit-shift-l, bit-shift-r, bit-xor



bit-shift-l

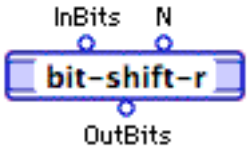
Description A bitwise shift left by one or more places for an integer is output. The N rightmost bits of the result are set to zero (0). For example the result for 5 (b101) shifted left by 2 is 20 (b10100).

Inputs **InBits** <integer>:

N <integer>: the number of places to the left that each bit is to be shifted.

Outputs **OutBits** <integer>:

See also bit-and, bit-not, bit-or, bit-shift-r, bit-xor



bit-shift-r

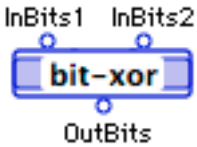
Description A bitwise shift right with sign extension by one or more places for an integer is output. For example the result for 13 (b1101) shifted right by 2 is 3 (b11) and the result for -13 (b11110011) shifted right by 2 is -4 (b11111100).

Inputs **InBits** <integer>:

N <integer>: the number of places to the right that each bit is to be shifted.

Outputs **OutBits** <integer>:

See also bit-and, bit-not, bit-or, bit-shift-l, bit-xor



bit-xor

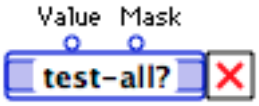
Description The bitwise EXCLUSIVE-OR of two integers is output.

Inputs **InBits1** <integer>:

InBits2 <integer>:

Outputs **OutBits** <integer>:

See also bit-and, bit-not, bit-or, bit-shift-l, bit-shift-r



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>:

Mask <integer>:

Outputs boolean

See also test-bit?, test-one?

test-bit?

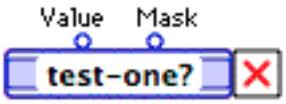
Description Returns TRUE if the bit in a specified position of an integer is set.

Inputs **Value** <integer>:

BitPosition <integer>:

Outputs boolean

See also test-all?, test-one?



test-one?

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>:

Outputs boolean

See also 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;
}
```

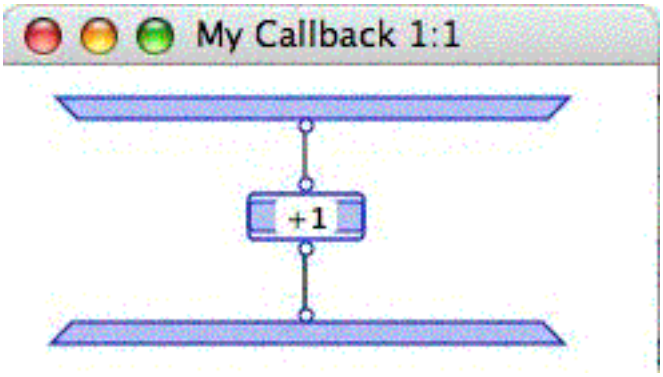
The function **VPL_TestCallback** accepts two parameters, the first is a long integer and the second is a callback. The declaration of **addOne** indicates that the callback is a function that takes a long integer as argument and returns a long integer. An example of such a callback is:

```
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:



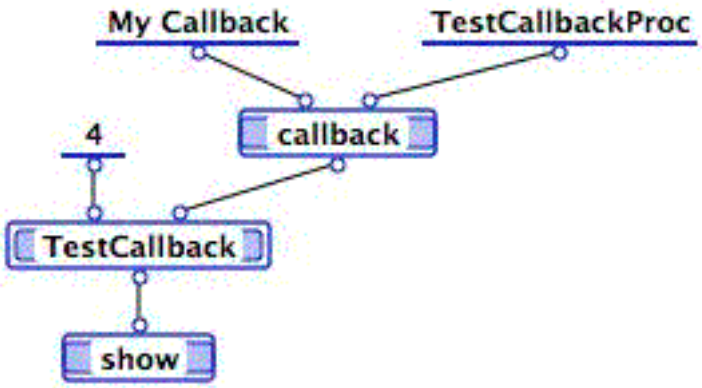
In order for this method to be used as a callback in Marten, the C function **VPL_TestCallback** must be created and a definition of it must be made available to the MacVPL engine. In addition, a definition of the structure of the callback function must also be present. An example is presented below:

```
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

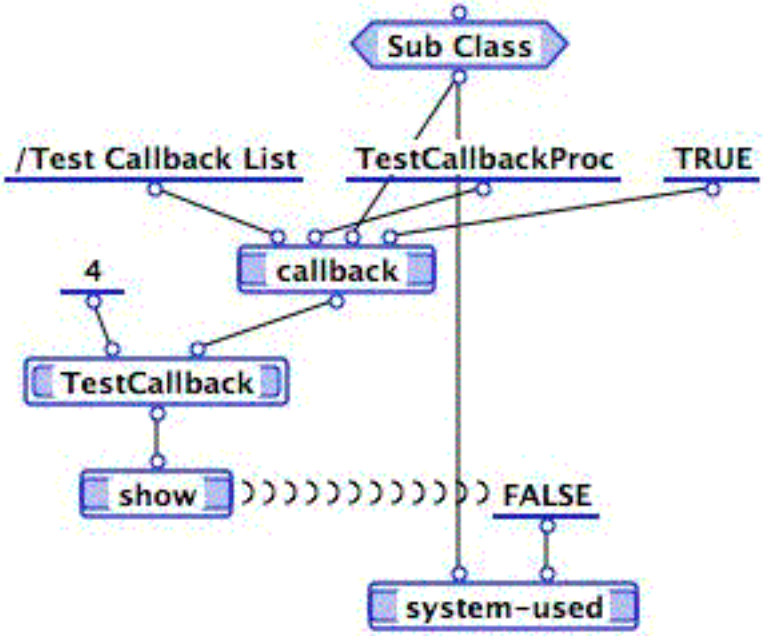
code, the Marten implementation of the callback primitive allocates a small amount of memory and writes a set of instructions into it that can be called directly by C. A reference to this memory is output from the callback primitive as an allocated block. This block is what is passed to the external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

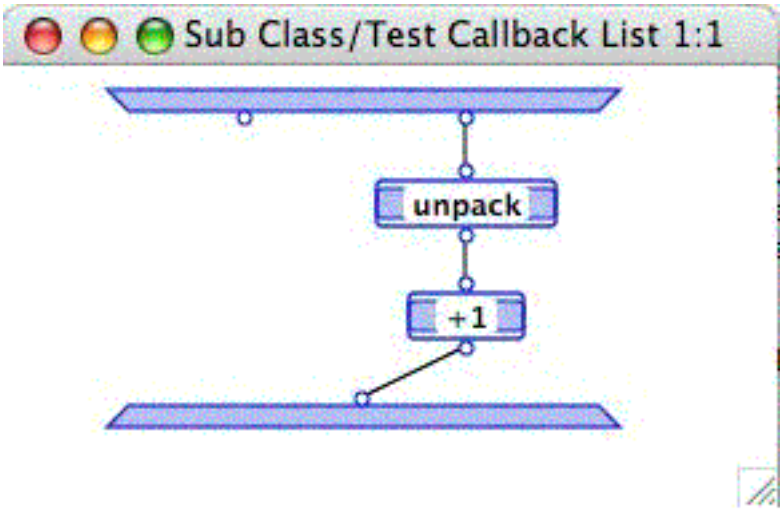
The first input to the callback primitive is the name of the Marten universal method to be used as the callback (for instance, "My Callback"). The required second input of the callback primitive is the name of the definition of the of the callback (in this case, "TestCallbackProc").

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 garbage-collected. 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:



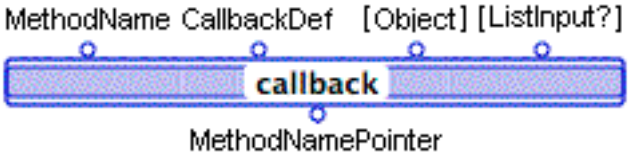
And the callback class method to be called is:



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:

- callback
- dispose-callback



callback

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 "Callbacks" 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

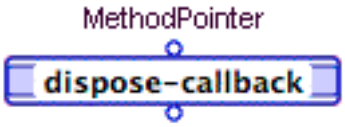
Object <<any>>: the object that will be passed as the first input into the called method.

ListInput? <boolean>:

Outputs **MethodNamePointer** <ABlock@>

Default(s) **ListInput?** = FALSE

See also dispose-callback



dispose-callback

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@>

See also callback

Data

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

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

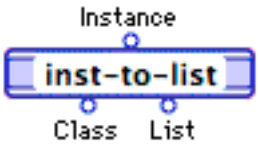
copy

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>:

Outputs **ItemCopy** <any>:

See also shallow copy



inst-to-list

Description Returns the class name and a list of instance attribute values of a provided instance.

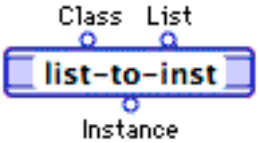
Inputs **Instance** <<any>>

Outputs **Class** <string>: the name of the class from which the instance was spawned.

List <list>: a list of the values of all instance attributes.

Note 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



list-to-inst

Description Creates and returns an instance of a specified class with a set of provided instance attribute values.

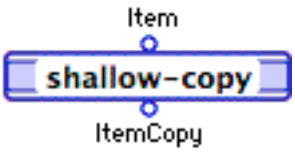
Inputs **Class** <string>: the name of the class for which an instance is to be spawned.

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>>

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 inst-to-list



shallow copy

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.

Inputs **Item** <any>:

Outputs **ItemCopy** <any>:

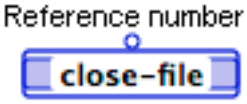
See also copy

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:

close-file	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.



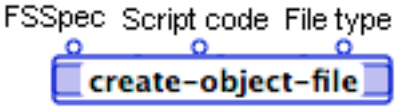
close-file

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.

Note The primitive can FAIL if the file cannot be closed.

See also create-object-file, create-text-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.

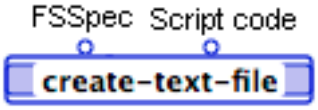
Inputs **FSSpec** <External Block>: A MacOS X FSSpec structure.

Script code <Integer>: A MacOS X script code returned by the put-file primitive.

File type <Integer>: A MacOS X file type (for example, 'TEXT').

Note The primitive can FAIL if the file cannot be created.

See also create-text-file, close-file, open-file



create-text-file

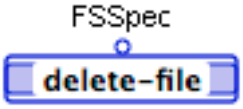
Description Creates a new file on disk of file type 'TEXT'. If a file already exists, it is deleted and then the new one created.

Inputs **FSSpec** <External Block>: A MacOS X FSSpec structure.

Script code <Integer>: A MacOS X script code returned by the put-file primitive.

Note The primitive can FAIL if the file cannot be created.

See also create-object-file, close-file, open-file

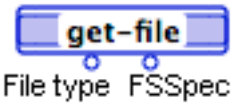


delete-file

Description Deletes the specified file.

Inputs **FSSpec** <External Block>: A MacOS X FSSpec structure.

Note The primitive can FAIL if the file cannot be created.



get-file

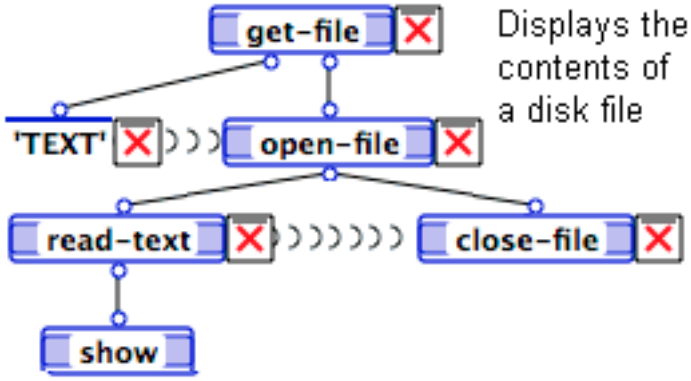
Description Opens a MacOS X Open File Navigation Dialog. If the user selects a file, the type and the FSSpec are returned. If user cancels, then get-file FAILs.

Outputs **File type** <Integer>: A MacOS X file type (for example, 'TEXT').

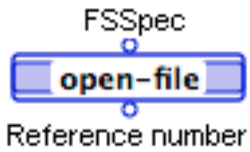
Note This primitive fails if the user cancels the dialog.

FSSpec <External Block>: A MacOS X FSSpec structure.

Example



See also put-file



open-file

Description Opens the file identified by an FSSpec and returns a file reference number to be used by primitives such as close-file and read-text.

Inputs **FSSpec** <External Block>: A MacOS X FSSpec structure.

Outputs **Reference number** <integer>: The MacOS X file reference number.

Note This primitive can FAIL if file cannot be opened.

See also create-object-file, create-text-file, close-file

put-file

Description Opens a MacOS X Save Location Navigation Dialog. If the user saves the file, the type, the FSSpec, and the script code are returned. If the user cancels, then this primitive FAILs.

Inputs Name <String>: The default name for the file to be saved.

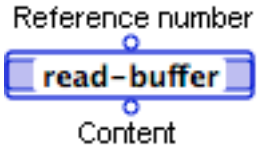
Outputs **File type** <Integer>: A MacOS X file type (for example, 'TEXT').

FSSpec <External Block>: A MacOS X FSSpec structure.

Script code <Integer>: A MacOS X script code.

Note The primitive FAILs if the user cancels the dialog.

See also get-file



read-buffer

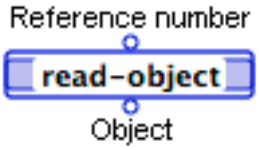
Description Reads in the contents of the file and places them into an external block.

Inputs **Reference number** <Integer>: A MacOS X file reference number

Outputs **Content** <External Block> The contents of the file. The type of block is "void".

Note This primitive can FAIL if the contents cannot be read.

See also read-object, read-text, write-object, write-text



read-object

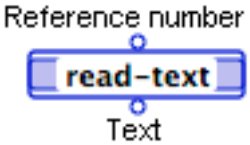
Description Reads in the contents of a Marten object file and returns the object.

Inputs **Reference number** <Integer>: The MacOS X file reference number

Outputs **Object** <any>: Any Marten object.

Can FAIL if contents cannot be read.

See also read-buffer, read-text, write-object, write-text



read-text

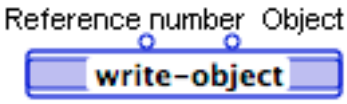
Description Reads in the contents of a text file and returns a String.

Inputs **Reference number** <Integer>: The MacOS X file reference number

Outputs Text <String>: The text contained in the file.

Note This primitive can FAIL if the content cannot be read.

See also read-object, read-buffer, write-object, write-text



write-object

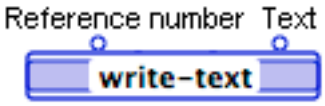
Description Writes the provided object to the specified file.

Inputs **Reference number** <Integer>: A MacOS X file reference number.

Object <any>: Any Marten object.

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

Description Writes the string to the text file identified by the file reference number.

Inputs **Reference number** <Integer>: A MacOS X file reference number.

Text <String>: The text to be written to the file.

Note This primitive can FAIL if the text cannot be written to file.

See also read-object, read-text, read-buffer, write-object

Graphics

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

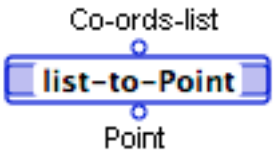
- Point
- RGB
- Rect.

These are represented textually in Marten as two, three, or four integers, respectively, separated by spaces, inside braces. For example, **{ 5 8 }** 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.

Outputs **Point** <External Block>: a MacOS X Point data structure.

See also Point-to-list

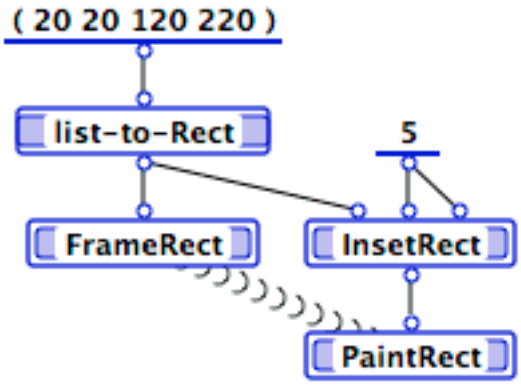
list-to-Rect

Description Returns the Rect specified by a set of co-ordinates.

Inputs **Co-ords-list** <list>: the four integers specifying the Rect co-ordinates.

Outputs **Point** <External Block>: a MacOS X Rect structure.

Example



See also Rect-to-list

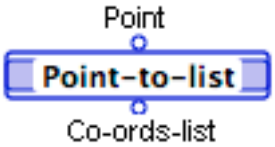
list-to-RGB

Description Returns the specified RGB structure.

Inputs **Specifiers-list** <list>: the three integers specifying the RGB.

Outputs **RGB** <External Block>: a MacOS X RGB structure.

See also RGB-to-list



Point-to-list

Description Returns the list of two coordinates specified by a Point.

Inputs **Point** <External Block>: a MacOS X Point structure.

Outputs **Specifiers-list** <list>: the two integers specifying the co-ordinates of the Point.

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.

Outputs **Co-ords-list** <list>: the four integers specifying the co-ordinates of the Rect.

RGB-to-list

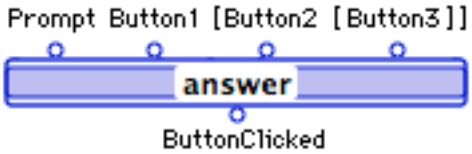
Description Returns the list of three colour specifiers of a Rect.

Inputs **RGB** <External Block>: a MacOS X RGB structure.

Outputs **Specifiers-list** <list>: the three integers specifying the individual colors of the RGB.

See also list-to-RGB

Input/Output



answer

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.

Button1 <any>: any valid Marten data item.

Button2 <any>: any valid Marten data item.

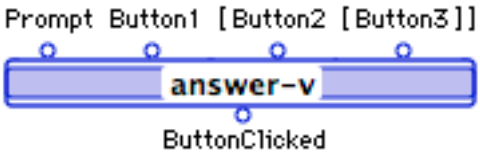
Button3 <any>: any valid Marten data item.

Outputs

ButtonClicked <any>: the value of the input parameter corresponding to the user's selection.

See also

answer-v, ask, select



answer-v

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.

Button1 <any>: any valid Marten data item.

Button2 <any>: any valid Marten data item.

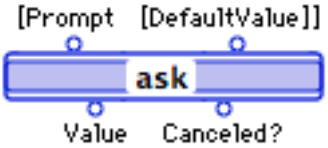
Button3 <any>: any valid Marten data item.

Outputs

ButtonClicked <any>: the value of the input corresponding to the user's selection.

See also

answer, ask, select



ask

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.

DefaultValue <any>: an initial, default value to be used as input if the user presses OK without typing a value in the editable area.

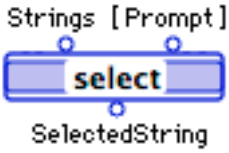
Note The **DefaultValue** parameter cannot contain an External structure or an instance of a class

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.

See also answer, answer-v, select



select

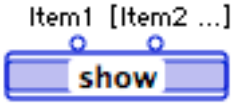
Description Opens a modal dialog prompting the user to make a selection from a list of provided alternatives. The modal dialog has a scrolling list, two buttons (**Select** and **Cancel**), and, optionally, a textual prompt.

Inputs **Strings** <list>: the items that are to be displayed in the scroll list.

Prompt <string>: a textual prompt displayed in the dialog.

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.

See also answer, answer-v, ask



show

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.

Item2... <any>: one terminal for each remaining Marten value to be displayed.

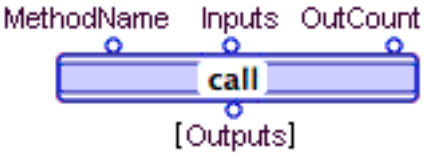
Note Inputs cannot be instances of classes or Windows types.

See also ask

Interpreter control

The following primitives are provided for use with the Interpreter:

call	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.



call

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).

Inputs <List>: A list of values to be passed into the method.

Outcount <Integer>: The number of outputs to expect.

Outputs

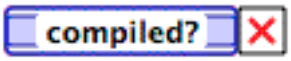
Outputs <List>: The list of outputs produced by the called method.

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.



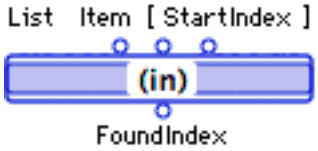
compiled?

Description

Result is TRUE if your program is compiled, and FALSE if your program is interpreted. This primitive can return a boolean result.

List

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



(in)

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 = primitive.

Inputs

List <list>: the list to be searched.

Item <any>: the value to search for.

StartIndex <integer>: the index of the first item to be checked.

Default(s)

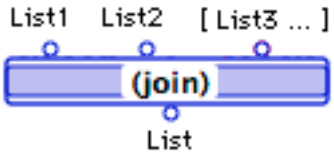
StartIndex = 1

Outputs

FoundIndex <integer>: the index of the found item or zero (0) if the item does not occur in the list.

See also

(length), (join)



(join)

Description Concatenates two or more lists.

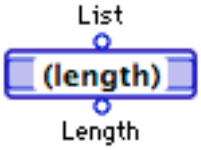
Inputs **List1** <list>: the list whose items are to be the first set of items in the new list.

List2 <list>: the list whose items are to follow the items of the List1 items in the new list.

List3 ... <list>: any remaining lists, one per terminal.

Outputs **List** <list>:

See also (in), (length)



(length)

Description Returns the length (number of elements) of a list.

Inputs **List** <list>:

Outputs **Length** <integer>:

See also (in), (join)

attach-l

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.

Element2 ... <any>: the second and subsequent items, one per terminal.

List <list>: the existing list.

Outputs **NewList** <list>: the new list.

See also attach-l, attach-r, detach-l, detach-nth, detach-r

attach-r

Description Creates a new list by concatenating an existing list and one or more provided elements.

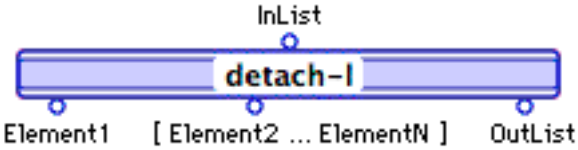
Inputs **List** <list>: the existing list.

Element1 <any>: the first item to be appended to the list.

Element2 ... <any>: any remaining items to be appended to the list, one per terminal.

Outputs **NewList** <list>: the new list.

See also attach-l, detach-l, detach-nth, detach-r



detach-l

Description For N+1 output roots, returns the first N elements of a list and a list containing the remaining elements.

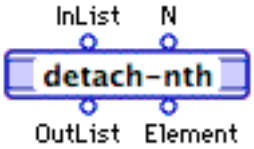
Inputs **InList** <list>: a list with at least N elements.

Outputs **Element1** <any>: the first element in the list

Element2 ... ElementN <any>: elements 2 - N of the list.

OutList <list>: the remaining elements in the list.

See also attach-l, attach-r, detach-nth, detach-r



detach-nth

Description Returns the Nth element of a list and a new list that results from removing the Nth element from the original list.

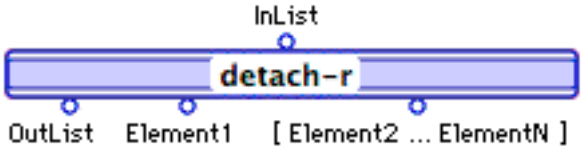
Inputs **InList** <list>: the original list.

N <integer>: the index of the item to be removed from the list.

Outputs **OutList** <list>: the new list.

Element <any>: the item removed the original list.

See also attach-l, attach-r, detach-l, detach-r



detach-r

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.

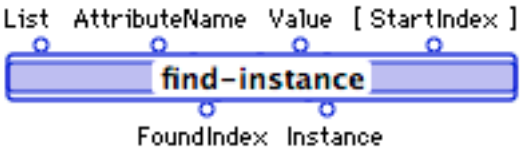
Inputs **InList** <list>: a list containing at least N elements.

Outputs **OutList** <list>: a new list containing the remaining elements of the original list.

Element1 <any>:

Element2 ... ElementN <any>:

See also attach-l, attach-r, detach-l, detach-nth



find-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.

AttributeName <string>: the name of the attribute to search.

Value <any>: the value of the attribute to search for.

StartIndex <integer>: the index within the list at which the search is to start.

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).

Instance <<instance>> | <null>: if the instance was found, this parameter contains the instance; otherwise it has a value of NULL.

See also find-sorted

find-sorted

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.

Item <string | number>: the value to be searched for.

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.

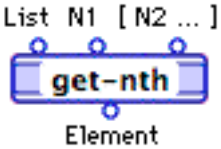
Outputs

Found <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.

See also

find-instance



get-nth

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.

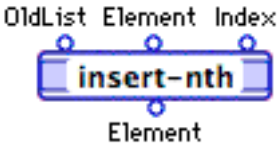
N2 ... <integer>: the index of the element within a nested list or the index of another nested list. The final terminal must be the index of the element to be returned in the deepest nested list.

Outputs

Element <any>:

See also

insert-nth, set-nth, set-nth!, split-nth



insert-nth

Description Creates a new list by inserting a provided element at a specified position in an existing list.

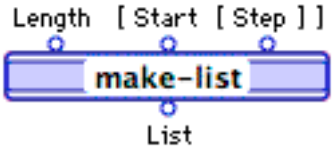
Inputs **OldList** <list>: the existing list.

Element <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, 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.

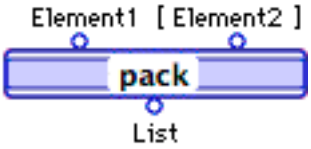
Start <any>: the value of the first item in the list and if the **Step** parameter is not provided, the value of all items in the list. If the **Start** parameter is not provided, all items of the list are created with an initial value of NULL.

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



pack

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 resulting list.

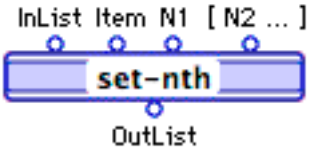
See also make-list, unpack

reverse

Description Creates a new list by reversing the order of items in an existing list.

Inputs **InList** <list>: the existing list.

Outputs **OutList** <list>: the new list.



set-nth

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 **InList** <list>:

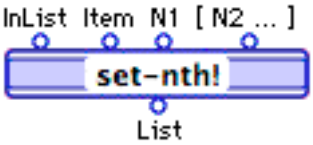
Item <any>: the new value of the specified list item.

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.

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.

Outputs **OutList** <list>: the resulting list.

See also get-nth, insert-nth, split-nth



set-nth!

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.

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.

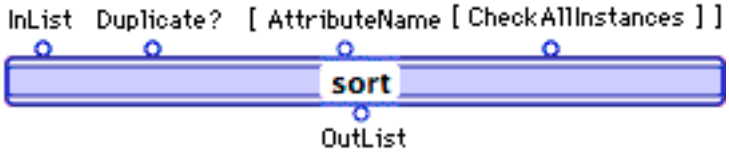
N2 ... <integer>: the index of the element within a nested list or the index of another nested list. The final terminal must be the index of the element to be given a new value in the deepest nested list.

Outputs **List** <list>:

Side effects 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

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



sort

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.

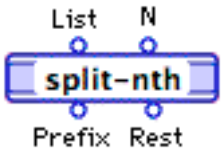
Inputs **InList** <list>: the existing list.

Duplicate <boolean>: if TRUE, items with identical values are 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.

AttributeName <string>: this parameter can only be provided when the InList parameter contains a list of instances. It specifies the name of the attribute that the instances are to be sorted on.

CheckAllInstances <boolean>: if all instances are of the same class, pass a value of TRUE with this parameter since it indicates that the specified attribute is at the same location in all instances.

Outputs **OutList** <list>: the sorted list.



split-nth

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 consisting of the remaining elements of the original list.

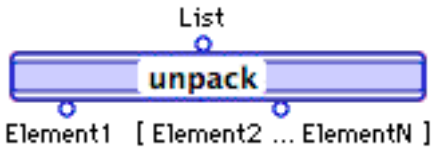
Inputs **List** <list>: the existing list.

N <integer>: the number of items to take from the existing list to create the first of the two new lists.

Outputs **Prefix** <list>: the list consisting of the first N elements of the existing list.

Rest <list>: the list consisting of the remaining elements of the existing list.

See also get-nth, insert-nth, set-nth, set-nth!



unpack

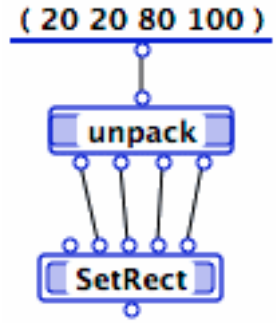
Description Given N output roots, returns the first N elements of a list. The list must contain at least N elements.

Inputs **List** <list>: the original list.

Outputs **Element1** <any>: the first element in the list.

Element2 ... ElementN <any>: elements 2 - N of the list.

Example



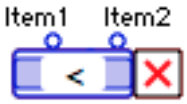
p

See also make-list, pack

Logical/Relational

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

<	<=	=	>	>=
and	choose	equals	gt	gte
lt	lte	not	or	xor



<

Description Succeeds if the first parameter is less than the second parameter.

Inputs **Item1** <string | number>

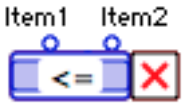
Item2 <string | number>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

Note Datatypes of the inputs MUST match.

Equivalent lt

See also <=, =, >, >=



<=

Description Succeeds if the first parameter is less than or equal to the second parameter.

Inputs **Item1** <string | number>

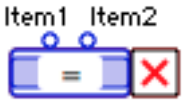
Item2 <string | number>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

Note Datatypes of the inputs MUST match.

Equivalent lte

See also <, =, >, >=



=

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.

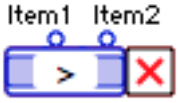
Inputs **Item1** <any>

Item2 <any>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

Equivalent equals

See also <, <=, >, >=



>

Description Succeeds if the first parameter is greater than the second parameter.

Inputs **Item1** <string | number>

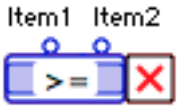
Item 2 <string | number>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

Note Datatypes of the inputs MUST match.

Equivalent gt

See also <, <=, =, >=



>=

Description Succeeds if the first parameter is greater than or equal to the second parameter.

Inputs **Item1** <string | number>

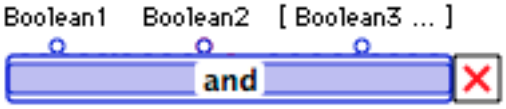
Item2 <string | number>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

Note Datatypes of the inputs MUST match.

Equivalent gte

See also <, <=, =, >



and

Description Performs a logical AND on two or more boolean values.

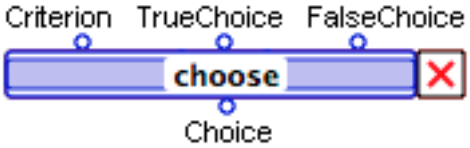
Inputs **Boolean1** <boolean>

Boolean2 <boolean>

Boolean3... <boolean>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

See also or, xor, not



choose

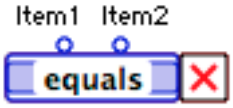
Description Returns one of two values based on the value of a supplied criterion.

Inputs **Criterion** <boolean>: the value that determines whether the **TrueChoice parameter** or **FalseChoice** parameter is returned by this primitive.

TrueChoice <any>: the value to be returned if the **Criterion** parameter is TRUE.

FalseChoice <any>: the value to be returned if the **Criterion** parameter is FALSE.

Outputs **Choice** <any>:



equals

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.

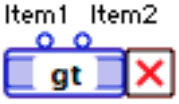
Inputs **Item1** <any>

Item2 <any>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

Equivalent =

See also <, <=, >, >=



gt

Description Succeeds if the first parameter is greater than the second parameter.

Inputs **Item1** <string | number>

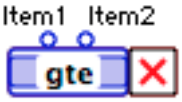
Item 2 <string | number>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

Note Datatypes of the inputs MUST match.

Equivalent >

See also <, <=, =, >=



gte

Description Succeeds if the first parameter is greater than or equal to the second parameter.

Inputs **Item1** <string | number>

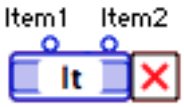
Item2 <string | number>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

Note Datatypes of the inputs MUST match.

Equivalent >=

See also <, <=, =, >



lt

Description Succeeds if the first parameter is less than the second parameter.

Inputs **Item1** <string | number>

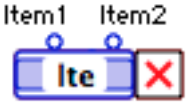
Item2 <string | number>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

Note Datatypes of the inputs MUST match.

Equivalent <

See also <=, =, >, >=



lte

Description Succeeds if the first parameter is less than or equal to the second parameter.

Inputs **Item1** <string | number>

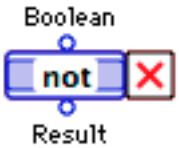
Item2 <string | number>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

Note Datatypes of the inputs MUST match.

Equivalent <=

See also <, =, >, >=



not

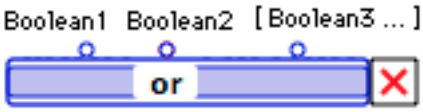
Description Returns the logical negation of a boolean value.

Inputs **Boolean** <boolean>

Outputs **Result** <boolean>

See also and, or, xor

or



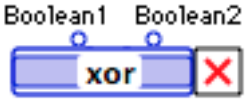
Description Performs a logical OR against two or more boolean values.

Inputs
Boolean1 <boolean>
Boolean2 <boolean>
Boolean3 <boolean>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

See also and, xor, not

xor



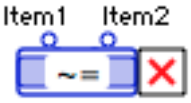
Description Performs an XOR (exclusive OR) against two boolean values.

Inputs
Boolean1 <boolean>
Boolean2 <boolean>

Outputs <boolean>: an output root can be added to this primitive if further processing of its result is required.

See also and, or, not

~=



Description Succeeds if the two parameters are not equal. This is the logical negation of the value that would be returned by the = primitive.

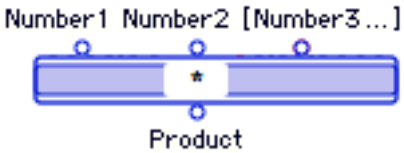
Inputs
Item1 <any>
Item2 <any>

Outputs <boolean>: an output root can be added to this primitive if further processing of the result is required.

Math

The following math primitives are provided:

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



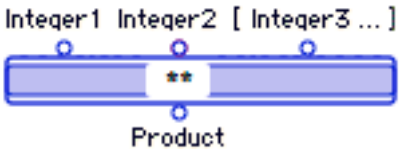
*

Description Returns the product of two or more numbers.

Inputs
Number1 <number>:
Number2 <number>:
Number3 ...<number>:

Outputs
Product <number>:

See also
+, -, div, ++, - -, **, idiv, +1, -1



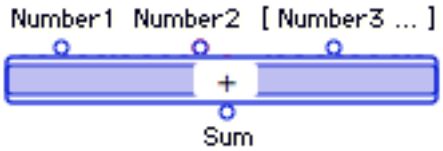
**

Description Returns the product of two or more integers.

Inputs
Integer1 <integer>:
Integer2 <integer>:
Integer3 ... <integer>:

Outputs
Product <integer>:

See also
+, -, div, ++, - -, *, idiv, +1, -1



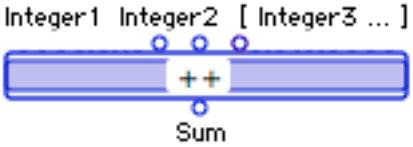
+

Description Returns the sum of two or more numbers.

Inputs
Number1 <number>:
Number2 <number>:
Number3 ...<number>:

Outputs
Sum <number>:

See also
-, div, ++, - -, *, **, idiv, +1, -1



++

Description Returns the sum of two or more integers.

Inputs **Integer1** <integer>:
Integer2 <integer>:
Integer3 ... <integer>:

Outputs **Sum** <integer>:

See also +, -, div, - -, *, **, idiv, +1, -1

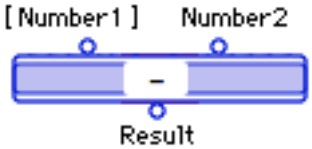
+1

Description Adds 1 to a provided number.

Inputs **Number** <number>:

Outputs **Result** <number>:

See also +, -, div, ++, - -, *, **, idiv, -1



-

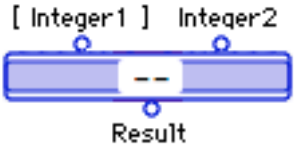
Description Subtracts the second parameter from the first parameter or negates the value of a single provided parameter.

Inputs **Number1** <number>:
Number2 <number>:

Default(s) **Number1** = 0

Outputs **Result** <number>:

See also +, div, ++, - -, *, **, idiv, +1, -1



- -

Description Subtracts the second integer parameter from the first or negates the value of a single provided integer.

Inputs **Integer1** <integer>:
Integer2 <integer>:
Integer3 ... <integer>:

Default(s) **Integer1** = 0

Outputs Result <integer>:

See also +, -, div, ++, *, **, idiv, +1, -1

-1

Description Subtracts 1 from a number.

Inputs **Number** <number>:

Outputs Result <number>:

See also +, -, div, ++, - -, *, **, idiv, +1

abs

Description Returns the absolute value of a number.

Inputs **Number** <number>:

Outputs **Result** <number>:

acos

Description **Angle** is arccosine (**Cosine**) expressed in radians.

Inputs **Cosine** <number>:

Outputs **Angle** <number>: the angle, in radians.

See also sin, cos, tan, asin, atan

asin

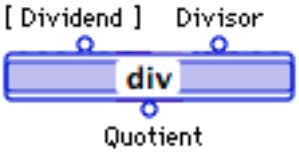
Description **Angle** is arcsine (**Sine**) expressed in radians.
Inputs **Size** <number>:
Outputs **Angle** <number>: the angle, in radians.
See also acos, sin, cos, tan, atan

atan

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

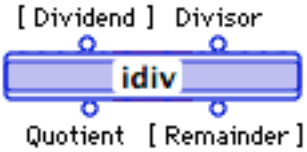
cos

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



div

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>:
Divisor <number>:
Default(s) **Dividend** = 1
Outputs **Quotient** <number>:
See also +, -, ++, - -, *, **, idiv, +1, -1



idiv

Description Performs integer division. It returns the quotient and remainder resulting from dividing the first parameter by the second.

Inputs **Dividend** <integer>:

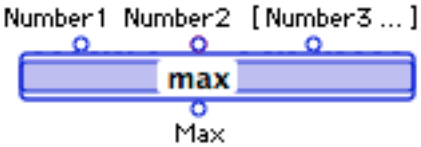
Divisor <integer>:

Default(s) **Dividend** = 1

Outputs **Quotient** <integer>:

Remainder <integer>:

See also +, -, div, ++, - -, *, **, +1, -1



max

Description Returns the maximum of two or more numbers.

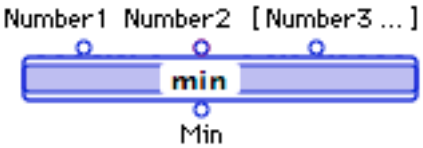
Inputs **Number1** <number>:

Number2 <number>:

Number3 ...<number>:

Outputs **Max** <number>:

See also min



min

Description Returns the minimum of two or more numbers.

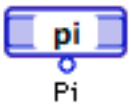
Inputs **Number1** <number>:

Number2 <number>:

Number3 ...<number>:

Outputs **Min** <number>:

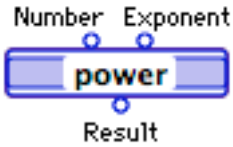
See also max



pi

Description Returns the value of pi (3.1415926...).

Outputs **Pi** <real>:



power

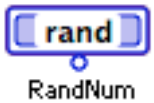
Description Calculates the value of a number to a provided exponent. If both parameters are zero (0), it returns 1.

Inputs **Number** <number>:

Exponent <number>:

Outputs **Result** <number>:

See also sqrt

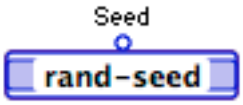


rand

Description Generates a random integer between and including 0 and (2^31) - 1.

Outputs **RandNum** <integer>:

See also rand-seed

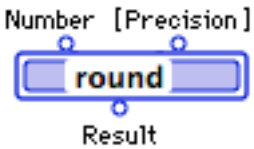


rand-seed

Description Sets the seed for the random-number generator to the integral part of Seed (1 to (2^31) - 1).

Inputs **Seed** <number>:

See also rand



round

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>:

Precision <integer>:

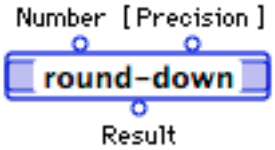
Outputs **Result** <number>:

Note If two values are equally near to the provided number, the round primitive uses the Apple SANE library convention of rounding to

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 number .95 is actually .9499999999999999999, for example. Use the format primitive to store such numbers as strings with the desired number of decimal places.

See also trunc, round-down, round-up



round-down

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.

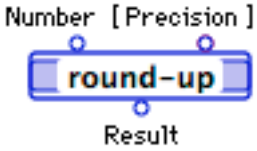
Inputs **Number** <number>:

Precision <integer>:

Default(s) **Precision** = 0 (return an integer).

Outputs **Result** <number>:

See also trunc, round, round-up



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>:

Precision <integer>:

Outputs **Result** <number>:

See also trunc, round, round-down

sin

Description Sine is sine (**Angle**).

Outputs **Angle** <number>: the angle, in radians.

Sine <number>:

See also acos, cos, tan, asin, atan

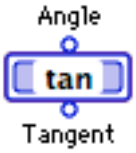
sqrt

Description Returns the square root of a number.

Inputs **Number** <number>:

Outputs **SquareRoot** <number>:

See also power



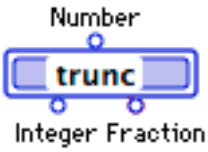
tan

Description Tangent is tangent (**Angle**).

Outputs **Angle** <number>: the angle, in radians.

Tangent <number>:

See also acos, sin, cos, asin, atan



trunc

Description Returns the integer and fraction components of a provided number.

Inputs **Number** <number>:

Outputs **Integer** <number>:

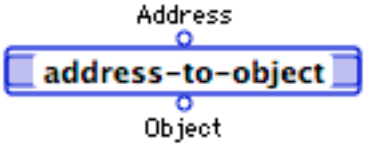
Fraction <real>:

See also round, round-down, round-up

Memory

The following primitives allow you to work directly with memory:

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.
get-real	Returns a real number of a specified length from a specified location.
get-text	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.



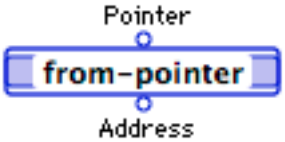
address-to-object

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>:

Outputs **Object** <any>:

See also object-to-address



from-pointer

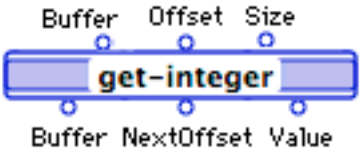
Description If the provided pointer is the address of a block of memory, then this primitive returns that address represented as an integer. In effect, it performs a cast of the pointer type to an integer.

Inputs **Pointer** <external@>:

Outputs **Address** <integer>:

Example If **Pointer** = ABlock@16#00211A00, **Address** = 16#00211A00.

See also to-pointer



get-integer

Description

Returns an integer of a specified size from a particular location within a memory block.

Inputs

Buffer <ABlock> | <ABlock@> | <ABlock@@ >| <integer>:

Offset <integer>: the offset, in bytes, at which to start reading the integer.

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

- 1
- 2
- 4

Outputs

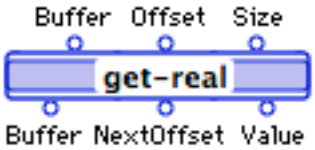
Buffer <ABlock> | <ABlock@> | <ABlock@@ >| <integer>:

NextOffset <integer>: the value of the Offset parameter added to the value of the Size parameter.

Value <integer>: the integer read.

See also

put-integer, get-real, get-text



get-real

Description

Returns a real number of a specified length from a specified location within a memory block.

Inputs

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

Outputs

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.

See also

put-real, get-integer, get-text

get-text

Description Returns a string of a specified size from a particular location within a memory block.

Inputs **Buffer** <ABlock> | <ABlock@> | <ABlock@@ >| <integer>:

Offset <integer>: the offset within the memory block at which to start reading.

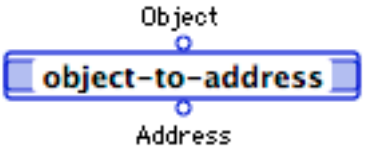
Size <integer>: must be greater than or equal to 0 and less than or equal to 65535 when using the interpreter and can be up to 4294967295 bytes when using the compiler).

Outputs **Buffer** <ABlock> | <ABlock@> | <ABlock@@ >| <integer>:

NextOffset <integer>: the value of the Offset parameter plus that of the Size parameter.

String <string>:

See also put-text, get-integer, get-real



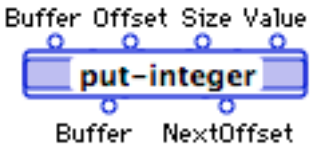
object-to-address

Description Returns the address of any Marten object.

Inputs **Object** <any>:

Outputs **Address** <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>:

Offset <integer>: the offset within the memory block at which to write the integer.

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

- 1
- 2
- 4

Value <integer>:

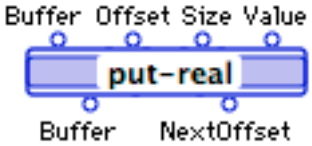
Outputs

Buffer <ABlock> | <ABlock@> | <ABlock@@ >| <integer>:

NextOffset <integer>: the value of the Offset parameter plus the value of the Size parameter.

See also

get-integer, put-real, put-text



put-real

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>:

Offset <integer>: the position in the memory block at which to write the floating point value.

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

- 4
- 8
- 10

Value <real> or <integer>:

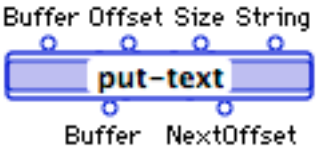
Outputs

Buffer <ABlock> | <ABlock@> | <ABlock@@ >| <integer>:

NextOffset <integer>: the value of the Offset parameter plus the value of the Size parameter.

See also

get-real, put-integer, put-text



put-text

Description Writes a specified number of bytes from a string to a particular position in a memory block.

Inputs **Buffer** <ABlock> | <ABlock@> | <ABlock@@ >| <integer>:

Offset <integer>:

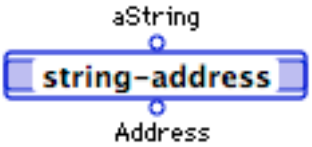
Size <integer>: must be greater than or equal to 0 and less than or equal to 65535 in the interpreter or 4294967295 bytes in the compiler.

String <string>:

Outputs **Buffer** <ABlock> | <ABlock@> | <ABlock@@ >| <integer>:

NextOffset <integer>: the value of the Offset parameter plus that of the Size parameter.

See also get-text, put-integer, put-real



string-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>:

Outputs **Address** <ABlock@>:

to-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>:

Outputs **Pointer** <ABlock@>:

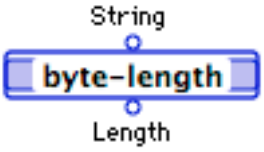
Example ExampleIf **Address** = 16#00211A00, **Pointer** = ABlock@16#00211A00.

See also from-pointer

String

The following primitives are provided for working with strings:

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.
"in"	Returns the location of a substring within a provided string.
integer-to-string	Returns the string representation of an integer.
"join"	Concatenates two or more strings.
"length"	Returns the number of characters in a string.
middle	Returns a substring of a specified string.
prefix	Returns two strings, the first *n* characters in one string and the remaining characters in the second string.
string-to-integer	Returns the integer representation of a provided string.
suffix	Returns two substrings, the last *n* characters in one string and the remaining characters in the second string.
to-ascii	Returns the ASCII integer representation of a provided string.
to-string	Returns the string representation of a provided Marten object.

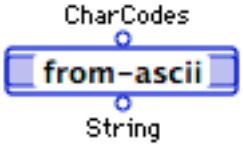


byte-length

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>:

Outputs **Length** <integer>: length of the string, in bytes.



from-ascii

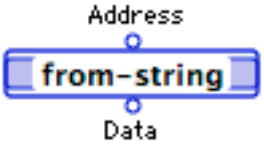
Description Returns the character representation of one or more ASCII codes.

Inputs **CharCodes** <integer | list of integers>: charCode or charCodeList.

Outputs **String** <string>: a string containing the characters represented by the provided ASCII codes.

Example If **CharCodes** = (72 101 108 108 111), then **String** = "Hello". If **CharCodes** = 72, **String** = "H".

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



from-string

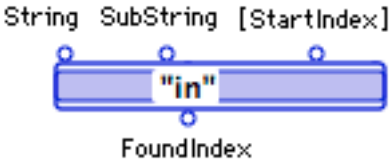
Description Returns the value textually represented by **String**. Type cannot be a class or External structure.

Note In producing output, this primitive follows Marten rules for unparsing. For details on data types, refer to the *Marten User Guide*.

Inputs **String** <string>:

Outputs **Data** <any*> | Point | Rect | RGBType

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



"in"

Description Returns the location of a substring in a provided string.

Inputs **String** <string>: the string in which to search.

SubString <string>: the substring to search for.

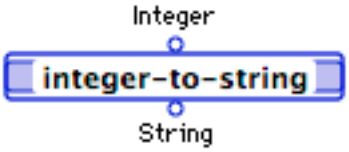
StartIndex <integer>: the index of the character in the String parameter at which to start the search. This parameter must be a positive number.

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.

Note If **SubString** is an empty string, **FoundIndex** returns 1.

See also "length", "join"



integer-to-string

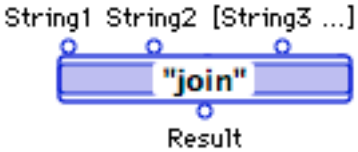
Description Returns the four character string representation of an integer.

Inputs **Integer** <integer>:

Outputs **String** <string>:

Example ExampleIf **Integer** = 16#54455854 ("TEXT"), then **String** = "TEXT".

See also string-to-integer



"join"

Description Concatenates two or more strings.

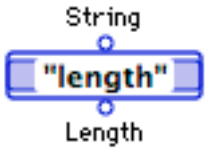
Inputs **String1** <string>:

String2 <string>:

String3 ... <string>:

Outputs **Result** <string>:

See also "in", "length"



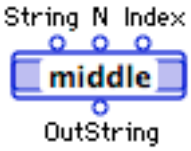
"length"

Description Returns the number of characters in a string.

Inputs **String** <string>:

Outputs **Length** <integer>:

See also "in", "join"



middle

Description Returns a substring of a specified number of characters from a provided string beginning at a specified position in the string.

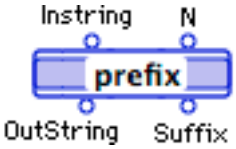
Inputs **String** <string>:

N <integer>: the number of characters to return.

Index <integer>: the index within the **String** parameter corresponding to the first character of the substring.

Outputs **OutString** <string>

See also prefix, suffix, "in"



prefix

Description Returns two substrings; the first consisting of the first **N** characters of the string and the second consisting of the remaining characters.

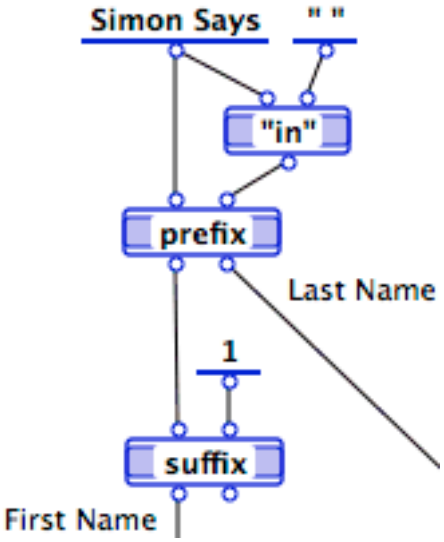
Inputs **InString** <string>:

N <integer>:

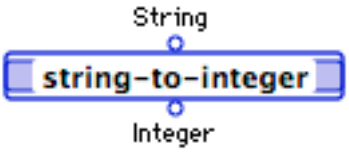
Outputs **OutString** <string>:

Suffix <string>:

Example 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.



See also middle, suffix, "in"



string-to-integer

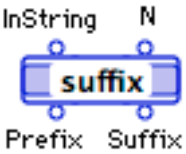
Description Returns the integer representation of a four character string.

Inputs **String** <string>:

Outputs **Integer** <integer>:

Example ExampleIf **String** = "TEXT", then **Integer** = 16#54455854 ("TEXT").

See also integer-to-string



suffix

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>:

N <integer>:

Outputs **Prefix** <string>: the initial characters of the **Instring** parameter; those not included in the **Suffix** parameter.

Suffix <string>: the last **N** characters of the **Instring** parameter.

See also middle, prefix, "in"

to-ascii

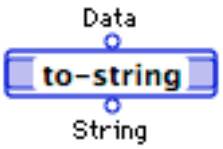
Description Returns the list of integers that are the ASCII representations of the characters in a provided string.

Inputs **String** <string>:

Outputs **CharCodeList** <list>:

Example ExampleIf **String** = "Hello", then **charCodeList** = (72 101 108 108 111).

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



to-string

Description Returns the textual representation of provided data.

Inputs **Data** <any*> | Point | Rect | RGBType: Input type cannot be a class or External Structure.

Outputs **String** <string>: the textual representation of the provided data. 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:

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

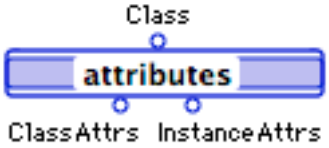
ancestors

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.

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.

See also children, descendants



attributes

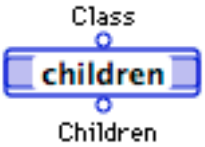
Description Returns the names of all instance attributess and class attributes of a specified class.

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.

Outputs **ClassAttrs** <list;>: a list of the names of class attributes in the specified class.

InstanceAttrs <list>: a list of the names of instance attributes in the specified class.

See also classes, methods, persistents



children

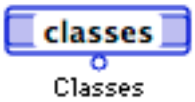
Description Returns the names of all classes that inherit DIRECTLY from a specified class.

Inputs **Class** <instance> | <string>: the name of the class or an instance of the class for which the list of child classes is to be returned.

Outputs **Children** <list>: a sorted list of class names.

Note Contrast the purpose of this primitive with that of the descendants primitive.

See also ancestors, descendants

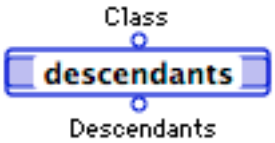


classes

Description Returns the names of all classes in a project.

Outputs **Classes** <list>: a list of the names of all classes.

See also attributes, methods, persistents



descendants

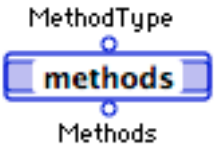
Description Return the names of all classes that are descendants of a specified class.

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.

Outputs **Descendants** <list of string>: a sorted list of the names of all classes that are descendants of the specified class.

Note Contrast the purpose of this primitive with that of the children primitive.

See also ancestors, children



methods

Description If the input is an Instance or aClass name, then Methods, GetMethods, and SetMethods are respectively lists of the names of simple or initialization methods, get methods, and set methods in the class specified by Instance or ClassName. If input is None, Methods is the list of names of Universal methods, and GetMethods and SetMethods are both ().

Inputs **MethodType** <instance> | <string> | none

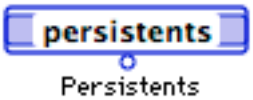
Outputs **Methods** <list>:

GetMethods <list>:

SetMethods <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



persistents

Description Returns the names of all persistents in the project.

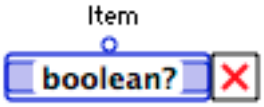
Outputs **Persistents** <list of strings>: a list of the names of all persistents.

See also attributes, classes, methods

Type

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.
type	Returns the type of a provided Marten object.



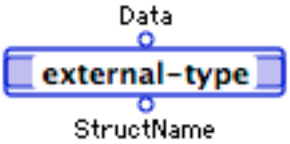
boolean?

Description Succeeds if the provided data is boolean, (TRUE or FALSE).

Inputs **Item** <any>: any Marten data item.

Outputs boolean

See also instance?, integer?, list?, external-type, number?, real?, string?, type



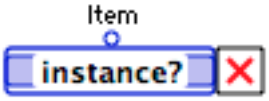
external-type

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>:

Outputs **StructName** <string>:

See also boolean?, instance?, integer?, list?, number?, real?, string?, type



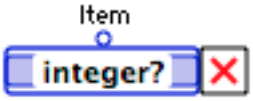
instance?

Description Succeeds if the provided data is an instance of a class.

Inputs **Item** <any>: any Marten data item.

Outputs boolean

See also boolean?, integer?, list?, external-type, number?, real?, string?, type



integer?

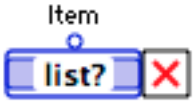
Description Succeeds if the provided data is an integer.

Inputs **Item** <any>: any Marten data item.

Outputs boolean

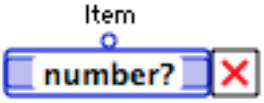
See also boolean?, instance?, list?, external-type, number?, real?, string?, type

list?



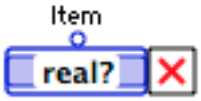
Description Succeeds if the provided data is a list.
Inputs **Item** <any>: any Marten data item.
Outputs boolean
See also boolean?, instance?, integer?, external-type, number?, real?, string?, type

number?



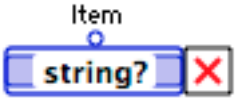
Description Succeeds if the provided data is areal or an integer.
Inputs **Item** <any>: any Marten data item.
Outputs boolean
See also boolean?, instance?, integer?, list?, external-type, real?, string?, type

real?



Description Succeeds if the provided data is a real number.
Inputs **Item** <any>: any Marten data item.
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.
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.

Outputs **Type** <string>: one of the following: boolean, integer, list, external, none, null, real, string or undefined.

See also boolean?, instance?, integer?, list?, external-type, number?, real?, string?

Index

Symbols

Numerics

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W

X