Onyx Manual, Version 4.4.3

Jason Evans

April 5, 2003

Preface

This manual primarily documents the Onyx programming language. However, Onyx is designed to be run either as a stand alone program or as an embeddable interpreter, so the manual also documents different aspects of the implementation that are important when embedding Onyx into another program.

Onyx came in to existence when the author started working on a text editor named slate (still in development) that was meant to be extensible, much in the same way as GNU emacs, JED, and Jade. One of the goals was to provide robust multi-threading in slate in order to make it simple to avoid the long pauses that afflict, for example, users of the gnus news/mail reader, which is part of emacs. Unfortunately, when work began on slate in 1999, the author was unable to find any embeddable scripting languages that provided adequate support for threads. Thus Onyx was born. The author was familiar and enamored with Adobe's PostScript$_{TM}$ language, which has basic threading support when used in a Display PostScript$_{TM}$ environment, so Onyx started off looking very similar. As Onyx matured, it deviated to the point that it is now a truly different language, with different syntax, additional and more powerful data types, better debugging capabilities, POSIX-related functionality, more powerful threading, regular expressions, etc.

As this project grew far beyond what was originally expected, it became clear that in order to justify the effort being put into Onyx's design and implementation, Onyx would have to be usable for more than just slate, or else slate would have to become *very* popular, which seems unlikely, given the plethora of text editors. Therefore, Onyx has been structured such that it can be configured in a myriad of ways, with the hope that others will be able to easily make it fit their needs. This manual documents Onyx in its full glory without mention that features may be disabled, so there are portions that do not apply to Onyx interpreters that have been configured without Onyx's full feature set.

For software distributions, news, and additional project information, see `http://www.canonware.com/`.

Contents

List of Tables

Chapter 1

Onyx Language Tutorial

This manual includes a comprehensive Onyx Language Reference chapter, which explains the details of what Onyx is. However, that chapter is rather dry, and more importantly, it does not discuss how to best utilize Onyx. This chapter introduces concepts that are important when designing and implementing Onyx programs, though it is not a complete language tutorial. You will need to read the first several sections of Chapter 2 in order to absorb all of the information in this chapter. However, you should be able to read this chapter first, then come back to it and glean additional understanding after having read later chapters.

Onyx is a stack-based language, so although the ideas that are important to program design in other more traditional languages still apply in many cases, there are different ways of approaching certain problems that integrate better with the facilities provided by Onyx. The most obvious example of this is that Onyx programs are more efficient if written to use named variables as little as possible, relying instead on the power of the operand stack. Another example is error handling. It is possible to write procedures that check for every error condition, but Onyx provides a form of exception handling that, if used correctly, can significantly improve code readability and performance.

Accomplished PostScript programmers will find little new in this chapter; Onyx differs from PostScript in the details, but the concepts are very similar. Accomplished Forth programmers will already be comfortable with stack management, but the rest of the chapter discusses concepts that either have no Forth equivalent, or that are significantly different from Forth, as is the case for dictionaries.

1.1 Syntax

Onyx syntax is extremely simple. Code is essentially composed of tokens that are delimited by whitespace or a list of self-delimiting tokens (see Section 2.2 for details). As such, there are very few ways for a syntax error to occur, but typographical mistakes may instead produce other errors. For example, say that a C programmer forgets he is writing Onyx code and types the following at the interactive *onyx* prompt:

```
onyx:0> 1000L {‘Hello\n’ print} repeat
```

The intention is to print `Hello` 1000 times, but `1000L` is invalid syntax for an integer, so Onyx creates an executable name object instead, and then tries to execute the name, resulting in the following error:

```
Error $undefined
ostack: ()
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      1000L
1:      -file-
2:      --start--
```

This is typical of the simple errors encountered when writing Onyx code. The Onyx scanner uses a simple state machine to try to create objects of various types, and when it fails, the input is instead used to create an executable name.

The scanner only deals with a few types (ignoring procedures for the moment): integers, reals, names, and strings. There are many other object types, but none of them are created directly by the scanner.

{ and } are used to delimit procedure bodies, which in actuality are executable arrays. { puts the scanner into deferred execution mode until the matching } is scanned. {} pairs can be nested, so execution is deferred until matching } characters have been scanned for all { characters. Deferred execution means that the scanner creates objects as it scans code, but does not execute any of them. While not a strictly necessary language feature, this greatly simplifies the task of constructing executable arrays, which can then be treated as procedures.

Following are equivalent examples of how a procedure associated with the name `double` can be defined:

```
onyx:0> $double {2 mul} def
onyx:0> $double [ 2 $mul load ] cvx def
```

As mentioned earlier, there are few ways of generating a syntax error, but it is possible. The most common syntax errors are due to unmatched ' and } characters. Generating other syntax errors is left as an exercise for the reader.

1.2 Data types

Onyx includes a rich set of data types. In fact, Onyx code is represented as data, which means that there is a whole range of possibilities when writing Onyx programs that are difficult or impossible with compiled languages such as C. This aspect of Onyx is discussed in Section 1.10.

Onyx is dynamically typed, which means that errors due to object type incompatibilities are detected during program execution. For example, the following code will always run without an error, even though the arguments that would be passed to the **add** operator are invalid.

```
false {
    ‘a string’ [1] add
}{
    ‘This is always printed’ 1 sprint
} ifelse
```

Dynamic typing has advantages in the flexibility that it offers, but it also means that type errors can go undetected in code for long periods of time before the invalid code is executed.

The Onyx type system is not extensible. There is no way to add new types in a program. However, there are techniques that can be combined with the built in types to simulate an extensible type system. For example, dictionaries can be created with, say, a `_type` field that indicates the type of the object:

```
<
    $_type  $fraction
    $num    3
    $den    4
>
```

Logical extensions of this can be used to create object hierarchies with single or multiple inheritance.

1.3 Execution

Onyx code is never compiled, nor is it preprocessed by the interpreter. Onyx code is simply consumed. This has some interesting implications, some of which are not typical of even other interpreted languages:

- If a source file is modified during interpreter execution, the changes may affect the currently running program, usually in unpleasant ways.
- Syntax errors are not discovered until the malformed code is scanned. In order to be sure that there are no syntax errors, a source file must be completely scanned.

In practice, these are only minor inconveniences, but it is important to keep them in mind when developing.

1.4 Memory management

Since Onyx includes an automatic mark and sweep garbage collector, memory management typically requires little thought. There is no risk of leaking memory in such a way that it cannot be freed. However, it is possible to consume large amounts of memory by creating objects, then keeping references to them long after they have outlived their usefulness.

Onyx objects fall into two categories according to type: simple and composite. Simple objects take up no virtual memory of their own; they are embedded into other composite objects. For example, an integer on the operand stack takes up only the space that the stack requires to store it. The same is true of an integer that is stored an an element of an array. Composite objects are composed of references that fit into the same places that an integer is stored, plus additional structures stored elsewhere in virtual memory. There can be multiple references to the same composite object, and as there is a chain of references that makes it possible to reach a composite object, the garbage collector will leave it alone.

It is usually pretty obvious how to remove references to objects. Objects on the operand stack can be popped off. Definitions in the dictionary stack can be undefined. However, there may be situations such an array that contains references to various objects, and the array cannot be discarded as a whole, but individual elements are no longer needed. The null type is useful for clobbering such references, and can even be effectively used to clobber portions of procedures, since when a null object is executed it does absolutely nothing. This unique aspect of null objects may not seem significant, but consider that all other objects, when executed, are either pushed onto the execution stack and executed, or pushed onto the operand stack. Doing nothing at all can be useful.

1.5 Stacks

Stacks in Onyx are pretty typical. Objects are implicitly or explicitly pushed onto stacks by operators, and the stack contents can be rearranged and removed. Although stacks are a first class object in Onyx, most Onyx programs are concerned only with the operand stack, often referred to as ostack, which is used as a place to store objects, pass arguments into operators and procedures, and return results.

Onyx is a postfix language, which means that code is written such that operands precede operators. For example, the following code Calculates $5 \times (3 + 4)$ and prints the result:

```
onyx:0> 5 3 4 add mul
onyx:1> 1 sprint
35
onyx:0>
```

There are no parentheses to clarify operator precedence, because precedence is implicit in the code.

Stacks are either written bottom to top on one line, or top to bottom on separate lines, as in the following examples. The example stack contains the numbers 0, 1, and 2, where 0 is the top object and 2 is the bottom object:

```
onyx:0> 2 1 0
onyx:3> ostack 1 sprint
(2 1 0)
onyx:3> pstack
0
1
2
onyx:3>
```

Learning to efficiently (and accurately) manage stacks is a mind-warping process that no amount of reading is likely to impress upon the reader. There are general concepts presented here, but ultimately, the reader will have to write a good bit of code to get a handle on stacks. Even the author of Onyx found himself stumbling over stacks well after Onyx was complete, despite limited exposure to stack-based languages beforehand. The problem seems to be that programmers learn to think in a different way that doesn't exercise the parts of the brain necessary for stack manipulation. Some people might argue that stack manipulation is the job of the compiler. In any case, stack manipulation is an acquired skill that requires practice.

1.5.1 Using stacks as queues

Stacks are implemented such that access to the top or bottom is very efficient, and efficiency decreases linearly, the farther into the stack an operation has to iterate. Since pushing and popping is efficient for both ends of stacks, this means that stacks are suitable for use as queues.

1.5.2 Using the operand stack as two stacks

The ability to efficiently manipulate both ends of the operand stack means that the operand stack can effectively be thought of as two stacks. This can be very useful in situations where more than a handful of objects are being manipulated, and an additional location to temporarily stash objects would be useful. If a program is repeatedly doing large stack rotations, using the bottom of the stack can often help to simplify the code and reduce stack manipulation overhead.

1.5.3 Efficiency issues

Since stacks are stored internally as doubly linked lists, the cost of indexed access to an object on a stack is proportional to its offset from the top or bottom of the stack, depending on the operator being used. Therefore, stacks are not ideal in situations where arbitrary access to an object is a common operation. An array is a better choice for indexed access, and a dictionary is a better choice for keyed access.

Indexed access doesn't just apply to operators like **idup**; operators like **nup** and **roll** are also affected. However, operators such as **rot** are not as heavily impacted, since they only need to index into the stack by the number of positions to rotate.

1.6 Dictionaries

Dictionaries are known by various other names, including hashes and associative arrays. Dictionaries in Onyx associate keys with values. Keys and values can be of any type, but for each dictionary, all keys are unique. For example, the following dictionary cannot exist:

```
<
    42 ‘Some value’
    42 ‘Another value’
>
```

To demonstrate this, the following example creates a dictionary with the first key/value pair listed above, then inserts the second key/value pair.

```
onyx:0> <42 ‘Some value’>
onyx:1> dup 1 sprint
<42 ‘Some value’>
onyx:1> dup 42 ‘Another value’ put
onyx:1> dup 1 sprint
```

```
<42 'Another value'>
onyx:1>
```

When the second key/value pair is inserted, it replaces the first pair.

1.6.1 Efficiency issues

Although dictionaries can handle keys of any type, they are optimized to use names as keys. The performance penalty for other key types is slight, but can be exacerbated by the cost of comparison for other types. Name comparison is a constant time operation, but string comparison is not. Therefore, use names rather than strings whenever possible.

1.7 Regular expressions

Onyx provides regular expression support that is very similar to what the Perl programming language provides. There are two special data types, regex and regsub, that are specific to regular expressions, but there is no special language syntax devoted to regular expressions, unlike Perl. Instead, patterns and substitution templates are specified via normal strings, and flags are specified via dictionaries.

1.7.1 Matching

The following snippet iteratively searches for capitalized words:

```
'This is an Onyx string.'

{dup '[A-Z]\w+' <$g true> match}{
    0 submatch 1 sprint
} while
```

The above code generates the following output:

```
'This'
'Onyx'
```

The `$g` flag to the **match** operator says to store the location to start the next match at, which is what makes the `while` loop possible. The **submatch** operator gets the substring of the input string that the regular expression most recently matched.

With a slight modification to the previous example, it is possible to get at the capital letters, rather than the entire capitalized words. This is achieved by using a set of capturing parentheses, and changing the argument to **submatch**:

```
'This is an Onyx string.'
```

```
{dup '([A-Z])\w+' <$g true> match}{
    1 submatch 1 sprint
} while
```

This generates the following output:

```
'T'
'O'
```

This is a trivial example of how capturing subpatterns can be used, but the possibilities are wide and varied.

1.7.2 Splitting

Sometimes it is desirable to **split** a string into pieces, such as when dealing with a comma-delimited file:

```
'Jason Evans, jasone@canonware.com, http://www.canonware.com/~jasone/'

',\s*' split
1 sprint
```

This generates the following output:

```
['Jason Evans' 'jasone@canonware.com' 'http://www.canonware.com/~jasone/']
```

If for some reason preserving the delimiters is important, capturing parentheses can be added to the splitting pattern:

```
'Jason Evans, jasone@canonware.com, http://www.canonware.com/~jasone/'

'(,)\s*' split
1 sprint
```

This generates the following output:

```
['Jason Evans' ',' 'jasone@canonware.com' ',' 'http://www.canonware.com/~jasone/']
```

1.7.3 Substituting

The **match** and **split** operators provide enough power that with some effort, it is possible to find regular expression matches, modify the matched text, and create a modified string as output. However, this is a common operation when using regular expressions for text processing, so the **subst** operator is also provided as a more convenient interface for the most common types of substitution.

Suppose that a comma-delimited file needs to be converted to a colon-delmited file, consecutive separators need to be merged, and whitespace around the separators needs to be stripped out. The following snippet does that:

```
‘Jason Evans (jasone@canonware.com)
Jason O. Evans ( jasone@canonware.com )
’

‘(\w[A-Za-z. ]*\w) \(\s*([^\s]+)\s*\)’ ‘"\1" <\2>’ <$g true> subst pop
stdout exch write pop
```

This generates the following output:

```
"Jason Evans" <jasone@canonware.com>
"Jason O. Evans" <jasone@canonware.com>
```

1.8 Code organization

As mentioned earlier, Onyx does not have named variables in the same way as many other languages. However, it does have the dictionary stack (dstack), which is dynamically used for name lookups whenever an executable name is interpreted. This section talks about how to effectively manage the namespace provided by dstack.

When the Onyx interpreter is first initialized, there are four dictionaries on dstack:

- userdict
- globaldict
- systemdict
- threaddict

dstack can be manipulated any way the application sees fit, though it is generally dangerous to remove or significantly modify systemdict or threaddict, since there are definitions in those dictionaries that are critical to the correct functioning of the interpreter.

Each thread has its own userdict, which provides a thread-local namespace. globaldict is shared among all threads, so it provides a global namespace. systemdict contains all of the default global definitions, and threaddict contains the default thread-local definitions.

For some applications, it may be desirable to add definitions to systemdict, but care should be taken not to overwrite existing definitions unless you really know what you are doing. In most cases, using globaldict is a better way to go.

1.8.1 Procedures

Named procedures can be created by associating a name in one of the dictionaries on dstack with a procedure. There is nothing magical about this, but some care should be taken in choosing procedure names, and in deciding how to manage the namespace. Onyx uses only lower case letters in the names of its definitions on dstack, so it is easy to avoid namespace collisions by using some other character in program definitions. There is nothing wrong with using all lowercase letters for names in your programs, though some extra care is warranted when doing so.

An important consideration in program design is determining what belongs in the global namespace. For an example of how Onyx partitions its namespace, look at gcdict. There are several operators defined in gcdict that are only of interest when manipulating the garbage collector, so those definitions are stashed out of the way. Your program can do the same for definitions that are not of regular interest.

1.8.2 Modules

Onyx provides the infrastructure for loadable modules, which can be implemented in Onyx and/or dynamically loaded machine code (shared libraries). See the **require** and **mrequire** documentation for details on how to load modules.

When writing a module that is meant as a library of additional functionality, some choices have to be made about how to organize the module. Some of the obvious choices for how to organize the loaded definitions are:

- Insert definitions into globaldict.
- Create a new dictionary full of definitions, and insert the dictionary into globaldict as, say, foodict.
- Create a new dictionary full of definitions, and put the dictionary on dstack somewhere, such as directly above or below systemdict. Note that this can be done by the application manually, so simply creating the dictionary and letting the application insert it where it pleases could be a better choice.
- Insert definitions into systemdict (questionable practice).

Depending on the nature of the module, any of the above solutions may be the right choice. There are tradeoffs between convenience and cleanliness that should be carefully weighed.

1.9 Error handling

Onyx includes a powerful generic error handling mechanism that can be extended and customized at several levels. The **throw** procedure is used to throw an error, and virtually every aspect of the error handling machinery can be customized, extended, or replaced, since it is all written in Onyx.

Errors have names, with which error handlers can be associated in errordict. errordict's **handleerror** procedure can be modified or replaced. In fact, with some care, an entirely custom errordict can be defined, then undefined once it is no longer needed.

Following is the standard idiom for setting up and tearing down custom error handling:

```
# Set up custom error handling...
{
```

```
    # Do error-prone stuff...
} stopped {
    # An error occurred.  Do additional cleanup...
} if
# Restore error handling machinery...
```

The possibilities are extensive. However, a word of caution is in order. If you mess something up in the error handling machinery, bad things will happen, and you will have a terrible time debugging the problem. Be careful.

1.10 Introspection

The following code defines a named procedure that calculates factorials.

```
# #n factorial #result
$factorial {
    dup 1 gt {
        # #n-1 factorial
        dup 1 sub factorial
    }{
        # Terminate recursion.
        1
    } ifelse

    mul
} def
```

What this code actually does is create an executable array, which can be examined and modified. The following transcript shows the effects of modifying the code.

```
onyx:0> 3 factorial 1 sprint
6
onyx:0> $factorial load 2 sprint
{dup 1 gt {dup 1 sub factorial} {1} ifelse mul}
onyx:0> $factorial load 4 {1 pstack} put
onyx:0> 3 factorial
1
1
2
3
onyx:1> 1 sprint
6
onyx:0>
```

First $3!$ is calculated. Then the “else” clause of the ifelse construct is modified to print the stack and $3!$ is recalculated. As can be seen, the stack is printed during the calculation.

1.11 Threads

The original impetus for Onyx's creation was the need for scalable threading. Onyx's threading is therefore truly powerful, though it comes at a cost. Threading makes asynchronous garbage collection a necessity, which in turn makes many aspects of Onyx's implementation a bit more heavyweight than would be necessary for a single-threaded interpreter.

1.11.1 Implicit synchronization

Onyx provides mechanisms for implicit object synchronization. To see why implicit object synchronization is necessary, consider what happens when two threads concurrently modify globaldict (a perfectly legitimate thing to do, by the way). The internals of a dictionary are rather complex, and if two modifications were interleaved, havoc would ensue. Therefore, globaldict is implicitly locked. That is a good thing, except that it slows down every access to globaldict. In contrast, userdict is a per-thread dictionary, so it is not implicitly locked.

Implicit locking for new objects is controlled via **setlocking**, and can be queried via **currentlocking**. Implicit locking is turned off by default when Onyx is started up, so if an application needs to create an object that is shared among threads, it should temporarily turn on implicit locking. For example, the following code creates a stack in globaldict that can be used as a simple message queue.

```
currentlocking # Save for later restoration.
true currentlocking

# Push globaldict onto dstack before calling def.
globaldict begin
$queue stack def
end # globaldict

# Restore implicit locking mode.
setlocking
```

For additional details on the mechanics of implicit synchronization, see Section 2.7.1.

1.11.2 General threading concerns

Onyx's basic threading mechanisms are typical of those found in modern threading implementations. One of the aspects of Onyx's threading implementation to be aware of is that since the operating system's threading implementation is used, there are many types of programming errors that can cause undefined behavior. With some effort, it is possible to crash the Onyx interpreter without the use of threads. However, it requires skill and discipline to *not* crash the Onyx interpreter when using threads. This was a conscious design decision for Onyx; doing otherwise would have limited the scalability of threading.

Now prepare for some discouraging pontification on why threaded programming is generally harmful. In the author's experience, the vast majority of developers do not have a solid enough grasp of threading concepts to be seriously engaging in the practice. As a first step to becoming proficient at threaded programming, much reading and thought are necessary. The next step is to implement several non-trivial threaded programs that are doomed to be complete disasters. After that, it may be possible for a programmer to write threaded programs that do the right thing most of the time. However, even the most skilled developers still will be unable to consistently write code that is free of race conditions, deadlocks, livelocks, locksteps, etc. Finally, there are some developers who, despite being otherwise proficient, will *never* obtain a solid grasp of threaded programming, regardless of how many times the basic concepts are reviewed.

In summary, don't use threads unless there are significant measurable gains in performance or code complexity. Otherwise, threaded programming just is not worth the unavoidable pain that it inflicts.

1.12 Optimization

There are many fine points to optimizing Onyx code, but they can primarily be distilled down to the following simple rules:

- Avoid allocating composite objects, in order to reduce pressure on the garbage collector. This means being very careful about string manipulation in the fast path. The **cat** operator is convenient, but not friendly to the garbage collector.
- Write code with as few objects as possible, to reduce the number of times through the interpreter loop. This means getting very familiar with the stack manipulation operators.
- Use the operand stack rather than named variables.
- Use the **bind** operator for procedure definitions whenever possible, in order to reduce dstack lookups.
- Avoid the **exit**, **stop**, and **quit** operators when possible, since they are implemented via *longjmp*().

1.13 Debugging

Onyx does not have an integrated interactive debugger per se, because the introspective power of Onyx is adequate for almost all debugging purposes. In cases where it is impractical to interactively debug an application via the main thread, it is possible to launch a thread that listens for connections on a socket (or a fifo pair) and provides an interactive session.

Following is a contrived example of debugging some bad code, interleaved with explanations. The intention is to calculate $1 + 5$.

```
onyx:0> 1 5L add
Error $undefined
ostack: (1)
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      5L
1:      -file-
2:      --start--
```

```
onyx:2> pstack
5L
1
```

`5L` is not a number, nor is it defined in dstack. Try replacing `5L` with `$five`.

```
onyx:2> pop $five resume
Error $typecheck
ostack: (1 $five)
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      --add--
1:      -file-
2:      --start--
onyx:3> pstack
--add--
$five
1
```

`$five` is a literal name, so no errors occur directly due to scanning it. However, the **add** operator expects two numbers, and `$five` is not a number. Replace it with `5` and evaluate the operator.

```
onyx:3> nip 5 exch eval
onyx:1> pstack
6
```

The result is as desired. However, we forgot to **resume** after the last error.

```
onyx:1> estack 1 sprint
(--start-- -file- --add-- --ifelse-- --eval-- -array- -file- --estack--)
onyx:1> resume
```

Now the estack contents should be back to normal.

```
onyx:1> estack 1 sprint
(--start-- -file- --estack--)
onyx:1>
```

The above example only demonstrates the flavor of typical interactive debugging, but there is no magic involved in debugging, so your debugging ability should improve automatically as you gain an improved understanding of Onyx.

Chapter 2

Onyx Language Reference

Onyx is a stack-based, threaded, interpreted language. Its closest relative is Adobe PostScript$_{TM}$, followed by Forth. Experienced PostScript programmers should find most aspects of Onyx familiar, but there are significant differences that will prevent a knowledgeable PostScript programmer from programming in Onyx without first skimming this chapter. This chapter does not assume specific knowledge of other programming languages, so stands as a definitive reference for Onyx.

Onyx is different from most languages in that it is not compiled, but rather consumed. For example, there are mechanisms for creating the equivalent of named procedures that can be called at a later time, but behind the scenes, the code is actually being interpreted as it is scanned in such a way that an executable object is created. As such, Onyx is not suited for compilation, native or byte code. However, the language syntax is very simple and the scanner/parser is extremely fast. There is also an operator called **bind** that optimizes interpreted code execution to approximately the same performance level as would be expected of a byte code interpreter.

Onyx is implemented as a C library that can be embedded in other programs. Mechanisms are provided for extending the set of operators available. This manual only documents the base language; see application-specific documentation for any language extensions.

Following is a list of basic language features that are discussed in more detail later in this chapter:

- Stack-based. There are no named variables as in procedural languages. Operations are done using various stacks, so Onyx operations are coded in postfix order.
- Threaded. Onyx's threading uses the native POSIX threads implementation of the operating system (or GNU pth, if so configured).
- Interpreted. Onyx code is never compiled, but is instead interpreted as it is encountered.
- Garbage-collected. There is no need to manually track memory allocation, since the interpreter has an integrated automatic mark and sweep garbage collector.

2.1 Objects

An Onyx object has three aspects: type, attribute, and value.

Objects fall into two categories according to type: simple and composite. A simple object takes up no memory of its own; it uses space within a stack, array, or dictionary. A composite object requires space of its own in addition to the space taken up in stacks, arrays, or dictionaries to refer to the composite object. See Table 2.1 for object type classifications.

Simple	Composite
boolean	array
fino	condition
integer	dict
mark	file
name	hook
null	mutex
operator	regex
pmark	regsub
real	stack
	string
	thread

Table 2.1: Simple and composite types

There can be multiple references that refer to the same memory backing composite objects. In most cases, composite objects that refer to the same memory are indistinguishable, but for arrays and strings, composite objects may only be able to access a subset of the total memory backing them. This behavior is described in detail later.

All objects have a literal, executable, or evaluatable attribute associated with them. Composite objects each have their own attribute, even for composite objects that share the same backing memory. Objects are "interpreted" when they are encountered directly by the interpreter. Objects can also be "evaluated". One of two actions is taken when an object is interpreted or evaluated:

- The object may be treated as code (executed). When executed, an object is pushed onto the execution stack and executed.
- The object may be treated as data. A data object is push onto the operand stack.

Table 2.2 enumerates under what circumstances object interpretation results in execution. Table 2.3 enumerates under what circumstances object evaluation results in execution. Note that executable arrays are the only objects that behave differently when interpreted versus evaluated.

In practice, attributes are only useful for types that can be executed. Attributes are not considered in equality test operations.

array: An array is an ordered sequence of objects of any type. The sequence of objects contained in an array is indexed starting at 0. References to existing arrays may be constructed such that a contiguous subsequence is visible. The following code creates such an array:

```
[0 1 2 3 4]
1 3 getinterval
```

After the code executes, the array left on the operand stack looks like:

Type	Attribute		
	literal	executable	evaluatable
array	data	data	code
boolean	data	data	data
condition	data	data	data
dict	data	data	data
file	data	code	code
fino	data	data	data
hook	data	code	code
integer	data	data	data
mark	data	data	data
mutex	data	data	data
name	data	code	code
null	data	code	code
operator	data	code	code
pmark	data	data	data
real	data	data	data
regex	data	data	data
regsub	data	data	data
stack	data	data	data
string	data	code	code
thread	data	data	data

Table 2.2: Interpretation of objects by type and attribute

```
[1 2 3]
```

Executable arrays are in effect procedures. When an array is executed, its elements are sequentially interpreted.

boolean: A boolean can have two values: true or false.

condition: A condition is used for thread synchronization. The standard operations on a condition are to wait and to signal.

dict: A dict (short for dictionary) is a collection of key/value pairs. Other names for dictionaries include "associative array" and "hash". A key can be of any type, though in most cases, keys are of type name. A value can also be of any type.

file: A file is a handle to an ordered sequence of bytes with a current position. Read and write permissions are set when a file object is created.

When an executable file is executed, it is used as a source of Onyx code. Data are sequentially read from the file and interpreted until the end of the file is reached.

fino: A fino (first in, never out) is used as a stack marker when constructing stacks.

hook: The hook type is not used by the core Onyx language. It can be used by applications that extend the interpreter as a container object. Hooks can be executed, but the results are application dependent.

Each hook has a tag associated with it that can used by C extension code as a form of type checking. By default, the tag is a null object. In most cases, an application that extends the interpreter using hook objects will set hook tags to be name objects.

Type	Attribute		
	literal	executable	evaluatable
array	data	code	code
boolean	data	data	data
condition	data	data	data
dict	data	data	data
file	data	code	code
fino	data	data	data
hook	data	code	code
integer	data	data	data
mark	data	data	data
mutex	data	data	data
name	data	code	code
null	data	code	code
operator	data	code	code
pmark	data	data	data
real	data	data	data
regex	data	data	data
regsub	data	data	data
stack	data	data	data
string	data	code	code
thread	data	data	data

Table 2.3: Evaluation of objects by type and attribute

integer: An integer is a signed integer in the range -2^{63} to $2^{63} - 1$.

mark: A mark is used as a stack marker for various stack operations.

mutex: A mutex is a mutual exclusion lock. Mutexes cannot be acquired recursively, and the application must take care to unlock mutexes before allowing them to be garbage collected (whether during normal program execution or at program termination).

name: A name is a key that uniquely identifies a sequence of characters. Two name objects that correspond to the same sequence of characters can be compared for equality with the same approximate cost as comparing two integers for equality. Names are typically used as keys in dictionaries.

When an executable name is executed, the topmost value in the dictionary stack associated with the name is evaluated.

null: A null has no significance other than its existence. When an executable null is executed, it does nothing. Executable nulls can be useful as place holders that can later be replaced with useful code, or for replacing obsolete code so that the code is no longer executed.

operator: An operator is an operation that is built in to the interpreter. Operators can be executed.

pmark: A pmark is used as a stack marker when creating procedures in deferred execution mode (i.e. procedures that use the {} syntax). The application will only encounter pmarks in error conditions, and there is never a reason for an application to explicitly create a pmark.

real: A real is a double precision (64 bit) floating point number.

regex: A regex encapsulates a regular expression and associated flags, which can be used to find substring matches within an input string.

regsub: A regsub encapsulates a regular expression, substitution template, and associated flags, which can be used to do substring substitutions matches and create an output string from input string.

stack: A stack provides LIFO (last in, first out) access to objects that it contains, as well as some more advanced access methods. An application can create, then manipulate stacks in much the same way that the operand stack can be manipulated.

string: A string is an ordered sequence of 8 bit characters. The bytes contained in an string are indexed starting at 0. References to existing strings may be constructed such that a contiguous subsequence is visible. The following code creates such a string:

```
'abcde'
1 3 getinterval
```

After the code executes, the string left on the operand stack looks like:

```
'bcd'
```

When an executable string is executed, its contents are used as a source of Onyx code.

thread: A thread object serves as a handle for operations such as detaching and joining.

2.2 Syntax

Onyx's syntax is very simple in comparison to most languages. The scanner and parser are implemented as a human-understandable finite state machine (nested C switch statements with a couple of auxiliary variables), which should give the reader an idea of the simplicity of the language syntax.

CRNL (carriage return, newline) pairs are in all important cases converted to newlines during scanning.

The characters #, !, $, ∼, [,], {, }, (,), ‘, ’, <, and > are special. In most cases, any of the special characters and whitespace (space, tab, newline, formfeed, null) terminate any preceding token. All other characters including non-printing characters are considered regular characters.

A comment starts with a # character outside of a string context and extends to the next newline or formfeed.

Procedures are actually executable arrays, but Onyx provides special syntax for declaring procedures. Procedures are delimited by { and }, and can be nested. Normally, the interpreter executes code as it is scanned, but inside of procedure declarations, execution is deferred. Instead of executing a procedure body as it is encountered, the tokens of the procedure body are pushed onto the operand stack until the closing } is encountered, at which time an executable array is constructed from the tokens in the procedure body and pushed onto the operand stack.

A partial grammar specification, using BNF notation (where convenient) is as follows:

<**program**> **::=** <statement>

<**statement**> **::=** <procedure> <statement> | <object> <statement> | ϵ

<**procedure**> **::=** {<statement>}

<**object**> **::=** <integer> | <real> | <name> | <string>

<**integer**> **::=** <dec_integer> | <radix_integer>

<**real**> **::=** <dec_real> | <exp_real>

<**name**> **:** Any token that cannot be interpreted as a number or a string is interpreted as an executable name. There are four syntaxes for names: executable, evaluatable, literal, and immediately evaluated. Executable and evaluatable names are looked up in the dictionary stack and executed (unless execution is deferred). Evaluatable names behave the same as executable names, except when bing processed by the **bind** operator. Literal names are simply pushed onto the operand stack. Immediately evaluated names are replaced by their values as defined in the dictionary stack, even if execution is deferred. Examples include:

```
foo     # executable
4noth3r # executable
!bar    # evaluatable
$biz    # literal
~baz    # immediately evaluated
```

If the result of an immediately evaluated name is an executable array, the evaluatable attribute is set for the array so that when the array is interpreted, it is executed. This allows immediate evaluation to be indiscriminately used without concern for whether the result is an executable array or, say, an executable operator.

<**string**> **::=** ‘’-delimited string. Ticks may be embedded in the string without escaping them, as long as the unescaped ticks are balanced. The following sequences have special meaning when escaped by a \ character:

‘ ‘ character.

’ ’ character.

\ \ character.

0 Nul.

n Newline.

r Carriage return.

t Tab.

b Backspace.

f Formfeed.

a Alarm.

e Escape.

x[0-9a-fA-F][0-9a-fA-F] Hex encoding for a byte.

c[a-zA-Z] Control character.

n (newline) Ignore.

r**n (carriage return, newline)** Ignore.

\ has no special meaning unless followed by a character in the above list.

Examples include:

```
‘’
‘A string.’
‘An embedded \n newline.’
‘Another embedded
newline.’
‘An ignored \
newline.’
‘Balanced ‘ and ’ are allowed.’
‘Manually escaped \‘ tick.’
‘Manually escaped \‘ tick and ‘balanced unescaped ticks’.’
‘An actual \\ backslash.’
‘Another actual \ backslash.’
```

<dec_integer> : Signed decimal integer in the range -2^{63} to $2^{63}-1$. The sign is optional. Examples include:

```
0
42
-365
+17
```

<radix_integer> : Signed integer with explicit base between 2 and 36, inclusive, in the range -2^{63} to $2^{63}-1$. Integer digits are composed of decimal numbers and lower or upper case letters. The sign is optional. Examples include:

```
2@101
16@ff
16@Ff
16@FF
-10@42
10@42
+10@42
9@18
35@7r3x
35@7R3x
```

<dec_real> : Double precision floating point number in decimal notation. At least one decimal digit and a decimal point are required. Examples include:

```
0.
.0
3.
.141
3.141
42.75
+3.50
-5.0
```

<exp_real> : Floating point number in exponential notation. The format is the same as for <dec_real>, except that an exponent is appended. The exponent is composed of an "e" or "E", an optional sign, and a base 10 integer that is limited by the precision of the floating point format (approximately -308 to 307). Examples include:

```
6.022e23
60.22e22
6.022e+23
1.661e-24
1.661E-24
```

Arrays do not have explicit syntactic support, but the [and] operators support their construction. Examples of array construction include:

```
[]
[0 ‘A string’ ‘Another string.’ true]
[5
42
false]
```

Dictionaries do not have explicit syntactic support, but the < and > operators support their construction. Examples of dictionary construction include:

```
<>
<$answer 42 $question ‘Who knows’ $translate {babelfish} >
```

Stacks do not have explicit syntactic support, but the (and) operators support their construction. Examples of stack contstruction include:

```
()
(1 2 mark ‘a’)
```

2.3 Stacks

Stacks in Onyx are the core data structure that programs act on. Stacks store objects in a last in, first out (LIFO) order. Onyx includes a number of operators that manipulate stacks.

Each Onyx thread has four program-visible stacks associated with it:

Operand stack (ostack): Most direct object manipulations are done using the operand stack. Operators use the operand stack for inputs and outputs, and code generally uses the operand stack for a place to store objects as they are being manipulated.

Dictionary stack (dstack): The dictionary stack is used for looking up names. Each thread starts with with four dictionaries on its dictionary stack, which are, from top to bottom:

- userdict
- globaldict

- systemdict
- threaddict

  The dictionary stack is normally manipulated via the **begin** and **end** operators. The initial dictionaries on the dictionary stack should not generally be removed, since doing so can cause interpreter crashes.

Execution stack (estack): The interpreter uses the execution stack to store objects that are being executed. The application generally does not need to explicitly manipulate the execution stack, but its contents are accessible, mainly for debugging purposes.

Index stack (istack): The interpreter uses the index stack to store execution offsets for arrays that are being executed. There is a one to one correspondence of the elements of the execution stack to the elements of the index stack, even though the elements of the index stack that do not correspond to arrays have no meaning. The index stack does not affect execution, and exists purely to allow useful execution stack traces when errors occur.

The application can also create additional stacks and manipulate them in much the same way as the operand stack can be manipulated.

2.4 Standard I/O

Onyx provides operators to access the standard I/O file objects: **stdin**, **stdout**, and **stderr**. Under normal circumstances, these operators are adequate for all standard I/O operations. However, it may be desirable to replace these files on a per-thread basis. This can be accomplished using **setstdin**, **setstdout**, and **setstderr**. Furthermore, the file objects that are inherited by new threads can be accessed and modified via **gstdin**, **gstdout**, **gstderr**, **setgstdin**, **setgstdout**, and **setgstderr**.

2.5 Interpreter recursion

During typical Onyx interpreter initialization, the **start** operator is executed, which in turn executes a file object corresponding to stdin. However, depending on how the interpreter is invoked, the initial execution stack state may differ.

The interpreter can be recursively invoked. For example, if the following code is executed, the **eval** operator recursively invokes the interpreter to interpret the string.

```
‘2 2 add’ cvx eval
```

The depth of the execution stack directly corresponds to the recursion depth of the interpreter. Execution stack depth is limited in order to catch unbounded recursion.

Onyx converts tail calls in order to prevent unbounded execution stack growth due to tail recursion. For example, the following code does not cause the execution stack to grow:

```
$foo {foo} def
foo
```

The following code will result in an execution stack overflow:

```
$foo {foo 'filler'} def
foo
```

2.6 Error handling

The error handling mechanisms in Onyx are simple but flexible. When an error occurs, **throw** is called. An error can have any name, but only the following error names are generated internally by Onyx:

argcheck: Incorrect argument value.

estackoverflow: Maximum interpreter recursion was exceeded.

invalidaccess: Permission error.

invalidexit: The **exit** operator was called outside of any loop. This error is generated as a result of catching an exit, so the execution state for where the error really happened is gone.

invalidfileaccess: Insufficient file permissions.

ioerror: I/O error (read(), write(), etc.).

limitcheck: Value outside of legal range.

neterror: Network error (refused connection, timeout, unreachable net, etc.).

rangecheck: Out of bounds string or array access, or out of bounds value.

regexerror: Regular expression syntax error.

stackunderflow: Not enough objects on stack.

syntaxerror: Scanner syntax error.

typecheck: Incorrect argument type.

undefined: Name not defined in any of the dictionaries on dstack.

undefinedfilename: Bad filename.

undefinedresult: Attempt to divide by 0.

unmatchedfino: No fino on ostack.

unmatchedmark: No mark on ostack.

unregistered: Non-enumerated error.

The Onyx scanner handles syntax errors specially, in that it pushes an executable string onto the operand stack that represents the code that caused the syntax error and records the line and column numbers in currenterror before invoking **throw**.

The Onyx scanner also handles immediate name evaluation errors specially, in that it pushes the name that could not be evaluated onto ostack before invoking **throw**.

In addition to the **throw** operator, there are several other operators that exist specifically for the purpose of error handling. The **start** operator silently catches any uncaught **stop**, **exit**, or **quit** calls. The **start** operator is the first operator called by the Onyx interpreter during startup, and it can also be used by applications to limit execution stack unwinding. The **stopped** operator is useful for catching **stop** calls, since it reports whether a **stop** call was caught, allowing conditional error recovery.

2.7 Threads

Onyx supports multiple threads of execution by using the operating system's native threading facilities. Along with threads comes the need for methods of synchronization between threads.

2.7.1 Implicit synchronization

Implicit synchronization is a mandatory language feature, since objects such as globaldict are implicitly accessed by the interpreter, which makes it impossible to require the user to explicitly handle all synchronization. Onyx provides optional implicit synchronization capabilities for composite objects on an object by object basis. Each thread has a setting which can be accessed via **currentlocking** (initially set to false) and set via **setlocking**. If implicit locking is active, then new objects will be created such that simple accesses are synchronized.

Implicit synchronization can be a source of deadlock, so care must be taken when accessing implicitly locked objects. For example, if two threads copy two implicitly locked strings to the other string, deadlock can result.

```
# Initialization.
$A 'aaaaaa'
$B 'bbbbbb'

...

# In thread A:
A B copy

...

# In thread B:
B A copy
```

The following are descriptions of the implicit locking semantics for each type of composite object:

array: Array copying is protected. Array element modifications are protected, but element reads are not protected.

condition: No implicit locking is done for conditions.

dict: All dict operations are protected.

file: All file operations are protected. There are no potential deadlocks due to implicit file locking.

hook: No implicit locking is done for hooks.

mutex: No implicit locking is done for mutexes.

regex: No implicit locking is done for regexes.

regsub: No implicit locking is done for regsubs.

stack: All stack operations are protected. There are no potential deadlocks due to implicit stack locking. However, there are races in stack copying, such that the results of copying a stack that is concurrently being modified are unpredictable. In addition, removing an object that is being concurrently accessed from a stack is unsafe.

string: String copying is protected. Character access is protected by many operators, but string copying is the only potential cause of deadlock for string access.

thread: Implicit locking is not done for thread operations, since other synchronization is adequate to protect thread objects.

2.7.2 Explicit synchronization

Onyx includes a foundation of mutexes and condition variables, with which all other synchronization primitives can be constructed.

2.8 Memory management

Onyx programs do not need to track memory allocations, since memory reclamation is done implicitly via automatic garbage collection. Onyx uses an atomic mark and sweep garbage collector.

The atomic nature of garbage collection may sound worrisome with regard to performance, but in fact there are tangible benefits and no significant negative impacts for most applications. Total throughput is improved, since minimal locking is necessary. Concurrent garbage collection would impose a significant locking overhead.

On the down side, atomic garbage collection cannot make strong real-time guarantees. However, the garbage collector is very efficient, and for typical applications, garbage collection delays are measured in microseconds up to tens of milliseconds on current hardware as of the year 2000. For interactive applications, anything under about 100 milliseconds is undetectable by the user, so under normal circumstances the user will not notice that garbage collection is happening.

There are three parameters that can be used to control garbage collection:

1. The garbage collector can be turned off for situations where many objects are being created over a short period of time.

2. The garbage collector runs whenever a certain number of bytes of memory have been allocated since the last collection. This threshold can be changed or disabled.

3. If no composite objects have been created for an extended period of time (seconds), the garbage collector will run if any composite objects have been allocated since the last collection. This idle timeout period can be changed or disabled.

There is one situation in which it is possible for garbage to never be collected, despite the garbage collector being properly configured. Suppose that a program creates some objects, the garbage collector runs, then the program enters a code path that clobbers object references, such that the objects could be collected, but no new objects are allocated. In such a situation, neither the allocation inactivity timer (period), nor the object allocation threshold will trigger a collection, and garbage will remain uncollected. In practice this situation is unlikely, and is not a significant problem since the program size is not growing.

Garbage collection is controlled via the gcdict dictionary, which is described in Section 2.10.4.

2.9 Regular expressions

Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England. PCRE stands for "Perl-compatible regular expressions". This manual only documents how Onyx interfaces with PCRE. For more information about how PCRE's regular expressions work, see the following:

- The pcre(3) manual page.
- The official PCRE website: `http://www.pcre.org/`.
- The official PCRE download site: `ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/`.

For general information about Perl regular expressions, the following are recommended:

- Perl 5.6.1 regular expression documentation: `http://www.perldoc.com/perl5.6.1/pod/perlre.html`.
- Mastering Regular Expressions, 2nd Ed., by Jeffrey E. F. Friedl. ISBN 0-596-00289-0.
- Perl 5 Pocket Reference, 3rd Ed., by Johan Vromans. ISBN 0-596-00032-4.
- Programming Perl, 3rd Ed., by Larry Wall, Tom Christiansen, and Jon Orwant. ISBN 0-596-00027-8.

There are two special Onyx object types that support regular expressions: regex and regsub. Objects of these types are created via the **regex** and **regsub** operators, respectively. The **match** operator applies a regular expression to an input string, the **submatch** operator returns a matched substring of the input string, and the **offset** operator returns the offset of a substring match relative to the beginning of the input string. The **split** operator creates an array of substrings that are separated by strings that match a regular expression. The **subst** operator finds regular expression matches within an input string and applies a substitution template to the matches, thereby creating an output string. See Section 2.10.9 for detailed documentation on the above-mentioned operators.

Regular expressions are written as strings in Onyx, so all of the standard special sequences within strings are interpreted directly by the Onyx scanner. Note that the \ character is only special within strings if it is followed by a special sequence. This allows regular expressions to seamlessly extend the set of special sequences within strings while maintaining a reasonably consistent syntax.

The following sequences have special meaning within strings that specify substitution templates:

Table 2.4: Substitution template special characters

Sequence	Description
`\1..\9`	Refer to captured expressions 1 through 9.

2.10 Dictionary reference

All operators built in to Onyx have corresponding names that are composed entirely of lower case letters and numbers (with the exception of syntax-supporting operators like []). In order to avoid any possibility of namespace collisions with names defined by current and future versions of Onyx, use at least one character that is not a lower case letter or a number in names (for example, capital letters, underscore, etc.). In practice, namespace collisions usually aren't a problem, even if they happen, since the only effect is that the program-defined definition shadows the built in definition during name lookups.

2.10.1 currenterror

Each thread has its own currenterror dictionary, which is used by the error handling machinery to store error state.

Table 2.5: currenterror summary

Input(s) Op/Proc/Var Output(s)	Description
– **newerror** boolean	Set to true during error handling.
– **errorname** name	Name of most recent error.
– **line** number	Get line number of syntax error.
– **column** number	Get column number of syntax error.
– **ostack** stack	ostack snapshot.

Continued on next page...

Table 2.5: *continued*

Input(s) Op/Proc/Var Output(s)	Description
– **dstack** stack	dstack snaphot.
– **estack** stack	estack snapshot.
– **istack** stack	istack snapshot.

– column *integer*:

Input(s): None.

Output(s):

integer: Column number, valid only if the error was a syntaxerror. Column numbering starts at 0.

Errors(s): None.

Description: Get the column number that a syntaxerror occurred on.

Example(s):

```
onyx:0> '1 2 3}' cvx eval
At line 1, column 5: Error $syntaxerror
ostack: (1 2 3 '}')
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..3):
0:      '1 2 3}'
1:      --eval--
2:      -file-
3:      --start--
onyx:5> currenterror $column get 1 sprint
5
onyx:5>
```

– dstack *stack*:

Input(s): None.

Output(s):

stack: A dstack snapshot.

Errors(s): None.

Description: Get a stack that is a dstack snapshot as of the most recent error.

Example(s):

```
onyx:0> x
Error $undefined
ostack: ()
dstack: (-dict- -dict- -dict- -dict-)
```

```
estack/istack trace (0..2):
0:      x
1:      -file-
2:      --start--
onyx:1> currenterror begin dstack end 1 sprint
(-dict- -dict- -dict- -dict-)
onyx:1>
```

– errorname *name*:

Input(s): None.

Output(s):

name: Name of the most recent error.

Errors(s): None.

Description: Get the name of the most recent error.

Example(s):

```
onyx:0> x
Error $undefined
ostack: ()
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      x
1:      -file-
2:      --start--
onyx:1> currenterror begin errorname end 1 sprint
$undefined
onyx:1>
```

– estack *stack*:

Input(s): None.

Output(s):

stack: An estack snapshot.

Errors(s): None.

Description: Get a stack that is an estack snapshot as of the most recent error.

Example(s):

```
onyx:0> x
Error $undefined
ostack: ()
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      x
1:      -file-
2:      --start--
onyx:1> currenterror begin estack end 1 sprint
(--start-- -file- x)
onyx:1>
```

– istack *stack*:

Input(s): None.

Output(s):

stack: An istack snapshot.

Errors(s): None.

Description: Get a stack that is an istack snapshot as of the most recent error.

Example(s):

```
onyx:0> x
Error $undefined
ostack: ()
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      x
1:      -file-
2:      --start--
onyx:1> currenterror begin istack end 1 sprint
(0 0 0)
onyx:1>
```

– newerror *boolean*:

Input(s): None.

Output(s):

boolean: False if there has been no error since the last time newerror was reset; true otherwise.

Errors(s): None.

Description: Get a boolean that represents whether there has been an error since the last time newerror was set to false (as during interpreter initialization). It is the application's responsibility to reset newerror after each error if it expects the value to be useful across multiple errors.

Example(s):

```
onyx:0> currenterror begin
onyx:0> newerror 1 sprint
false
onyx:0> x
Error $undefined
ostack: ()
dstack: (-dict- -dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      x
1:      -file-
2:      --start--
onyx:1> newerror 1 sprint
true
onyx:1> $newerror false def
onyx:1> newerror 1 sprint
false
onyx:1> resume
onyx:1> y
Error $undefined
ostack: (x)
dstack: (-dict- -dict- -dict- -dict- -dict-)
```

```
estack/istack trace (0..2):
0:      y
1:      -file-
2:      --start--
onyx:2> newerror 1 sprint
true
onyx:2>
```

– line *integer*:

Input(s): None.

Output(s):

integer: Line number, valid only if the error was a syntaxerror. Line numbering starts at 1.

Errors(s): None.

Description: Get the line number that a syntaxerror occurred on.

Example(s):

```
onyx:0> ‘1 2 3}’ cvx eval
At line 1, column 5: Error $syntaxerror
ostack: (1 2 3 ‘}’)
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..3):
0:      ‘1 2 3}’
1:      --eval--
2:      -file-
3:      --start--
onyx:5> currenterror $line get 1 sprint
1
onyx:5>
```

– ostack *stack*:

Input(s): None.

Output(s):

stack: An ostack snapshot.

Errors(s): None.

Description: Get a stack that is an ostack snapshot as of the most recent error.

Example(s):

```
onyx:0> x
Error $undefined
ostack: ()
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      x
1:      -file-
2:      --start--
onyx:1> currenterror begin ostack end 1 sprint
()
onyx:1>
```

2.10.2 envdict

The envdict dictionary contains keys of type name and values of type string that correspond to the environment passed into the program. All threads share the same envdict, which is implicitly locked. Modifications to envdict should be made via the **setenv** and **unsetenv** operators. If envdict is modified directly, the changes will not be visible to programs such as *ps*.

2.10.3 errordict

Each thread has its own errordict, which is used by default by the error handling machinery.

Table 2.6: errordict summary

Input(s) Op/Proc/Var Output(s)	Description
– **handleerror** –	Print a state dump.
– **stop** –	Last operation during error handling.

– handleerror –:

Input(s): None.

Output(s): None.

Errors(s): Under normal conditions, no errors occur. However, it is possible for the application to corrupt the error handling machinery to the point that an error will occur. If that happens, the result is possible infinite recursion, and program crashes are a real possibility.

Description: Print a dump of the most recent error recorded in the currenterror dictionary.

Example(s):

```
onyx:0> {true {true 1 sprint x y} if} eval
true
Error $undefined
ostack: ()
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..5):
0:      x
1: {
        true
        1
        sprint
 3:-->  x
        y
}
2:      --if--
3:      --eval--
```

```
4:      -file-
5:      --start--
onyx:1> errordict begin handleerror end
Error $undefined
ostack: ()
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..5):
0:      x
1: {
        true
        1
        sprint
 3:-->  x
        y
}
2:      --if--
3:      --eval--
4:      -file-
5:      --start--
onyx:1>
```

– stop –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: This is called as the very last operation when an error occurs. Initially, its value is the same as that for the **stop** operator in systemdict.

Example(s):

```
onyx:0> errordict begin
onyx:0> $stop {‘Custom stop\n’ print flush quit} def
onyx:0> x
Error $undefined
ostack: ()
dstack: (-dict- -dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      x
1:      -file-
2:      --start--
Custom stop
```

2.10.4 gcdict

The gcdict dictionary provides garbage collection control and status capabilities.

Table 2.7: gcdict summary

Input(s) Op/Proc/Var Output(s)	Description
Control operators	
– **collect** –	Force a garbage collection.
boolean **setactive** –	Set whether the garbage collector is active.
seconds **setperiod** –	Set the inactivity period before the garbage collector will run.
count **setthreshold** –	Set the number of bytes of memory allocation that will trigger a garbage collection.
State and statistics operators	
– **active** boolean	Get whether the garbage collector is active.
– **period** seconds	Get the inactivity period before the garbage collector will run.
– **threshold** count	Get the number of bytes of memory allocation that will trigger a garbage collection.
– **stats** array	Get garbage collection statistics.

– active *boolean*:

Input(s): None.

Output(s):

boolean: If true, the garbage collector is active; otherwise it is not active.

Errors(s): None.

Description: Get whether the garbage collector is active.

Example(s):

```
onyx:0> gcdict begin active end 1 sprint
false
```

– collect –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Force a garbage collection.

Example(s):

```
onyx:0> gcdict begin collect end
onyx:0>
```

– period *seconds*:

Input(s): None.

Output(s):

seconds: The minimum number of seconds since the last object allocation that the garbage collector will wait before doing a garbage collection. 0 is treated specially to mean forever.

Errors(s): None.

Description: Get the minimum number of seconds of object allocation inactivity that the garbage collector will wait before doing a garbage collection. This setting is disjoint from the threshold setting, and does not prevent garbage collection due to the threshold having been reached.

Example(s):

```
onyx:0> gcdict begin period end 1 sprint
60
onyx:0>
```

***boolean* setactive –:**

Input(s):

boolean: If true (initial setting), activate the garbage collector; otherwise deactivate the garbage collector.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Set whether the garbage collector is active. This setting takes effect asynchronously, so it is possible for the garbage collector to run even after it has been deactivated. This setting overrides the allocation inactivity period and allocation threshold settings, so that if this setting is set to false, the other settings have no effect.

Example(s):

```
onyx:0> gcdict begin false setactive end
onyx:0>
```

***seconds* setperiod –:**

Input(s):

seconds: The minimum number of seconds since the last object allocation that the garbage collector will wait before doing a garbage collection. 0 is treated specially to mean forever.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

limitcheck.

Description: Set the minimum number of seconds of object allocation inactivity that the garbage collector will wait before doing a garbage collection. This setting is disjoint from the threshold setting, and does not prevent garbage collection due to the threshold having been reached.

Example(s):

```
onyx:0> gcdict begin 60 setperiod end
onyx:0>
```

***count* setthreshold –:**

Input(s):

count: Number of bytes of memory allocation since the last garbage collection that will trigger a garbage collection. 0 is treated specially to mean infinity.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

limitcheck.

Description: Set the number of bytes of memory allocation since the last garbage collection that will trigger a garbage collection. This setting is disjoint from the inactivity period setting, and does not prevent garbage collection due to the allocation inactivity period having been exceeded.

Example(s):

```
onyx:0> gcdict begin 40000 setthreshold end
onyx:0>
```

– stats *array*:

Input(s): None.

Output(s):

array: An array with the format [collections count [ccount cmark csweep] [mcount mmark msweep] [scount smark ssweep]], where the fields have the following meanings:

collections: Total number of collections the garbage collector has performed.

count: Current number of bytes of memory allocated.

ccount: Number of bytes of memory allocated as of the end of the most recent garbage collection.

cmark: Number of microseconds taken by the most recent garbage collection mark phase.

csweep: Number of microseconds taken by the most recent garbage collection sweep phase.

mcount: Largest number of bytes of memory ever allocated at any point in time.

mmark: Maximum number of microseconds taken by any garbage collection mark phase.

msweep: Number of microseconds taken by any garbage collection sweep phase.

scount: Total number of bytes of memory ever allocated.

smark: Total number of microseconds taken by all garbage collection mark phases.

ssweep: Total number of microseconds taken by all garbage collection sweep phases.

Errors(s): None.

Description: Get statistics about the garbage collector.

Example(s):

```
onyx:0> gcdict begin
onyx:0> stats 2 sprint
[23 72673 [72268 754 3467] [4752223 930 36492] [51057886 17448 136807]]
onyx:0>
```

– threshold *count*:

Input(s): None.

Output(s):

count: Number of bytes of memory allocation since the last garbage collection that will trigger a garbage collection. 0 is treated specially to mean infinity.

Errors(s): None.

Description: Get the number of bytes of memory allocation since the last garbage collection that will trigger a garbage collection. This setting is disjoint from the inactivity period setting, and does not prevent garbage collection due to the allocation inactivity period having been exceeded.

Example(s):

```
onyx:0> gcdict begin threshold end 1 sprint
65536
onyx:0>
```

2.10.5 globaldict

All threads share the same globaldict, which is meant as a repository for globally shared objects. globaldict is empty when the Onyx interpreter is initialized, and is implicitly locked.

2.10.6 onyxdict

Various portions of Onyx use the onyxdict dictionary for storage of miscellaneous objects that normally should not be part of the namespace visible to dstack searches.

Table 2.8: onyxdict summary

Input(s) / Op/Proc/Var / Output(s)	Description
– / **mpath_post** / array	Get path searched by mrequire.
– / **mpath_pre** / array	Get path searched by mrequire.
– / **rpath_post** / array	Get path searched by require.
– / **rpath_pre** / array	Get path searched by require.

– mpath_post *array*:

Input(s): None.

Output(s):

array: An array of strings.

Errors(s): None.

Description: Get an array of strings used by mrequire as prefixes for file searches. The elements of the array are tried in the order listed.

Example(s):

```
onyx:0> onyxdict $mpath_post get 1 sprint
['/usr/local/share/onyx-3.0.0/nxm']
onyx:0>
```

– mpath_pre *array*:

Input(s): None.

Output(s):

array: An array of strings.

Errors(s): None.

Description: Get an array of strings used by mrequire as prefixes for file searches. The elements of the array are tried in the order listed.

Example(s):

```
onyx:0> onyxdict $mpath_pre get 1 sprint
['' '.']
onyx:0>
```

– rpath_post *array*:

Input(s): None.

Output(s):

array: An array of strings.

Errors(s): None.

Description: Get an array of strings used by require as prefixes for file searches. The elements of the array are tried in the order listed.

Example(s):

```
onyx:0> onyxdict $rpath_post get 1 sprint
['/usr/local/share/onyx-3.0.0/nx']
onyx:0>
```

– rpath_pre *array*:

Input(s): None.

Output(s):

array: An array of strings.

Errors(s): None.

Description: Get an array of strings used by require as prefixes for file searches. The elements of the array are tried in the order listed.

Example(s):

```
onyx:0> onyxdict $rpath_pre get 1 sprint
['' '.']
onyx:0>
```

2.10.7 outputsdict

The outputsdict dictionary is primarily used to support **outputs**, but its contents may be of use to an application that wishes to extend or modify formatted printing.

There is an entry in outputsdict for each Onyx type. Each entry renders objects that correspond to its name using optional flags stored in a dictionary. The following flags are supported for all types:

`$n`: Maximum length, in bytes. Default: disabled.

`$w`: Minimum length, in bytes. Default: disabled.

`$j`: Justification. Legal values:

- `$l`: Left.
- `$c`: Center.
- `$r`: Right (default).

`$p`: Padding character. Default: `' '`.

`$r`: Syntactic rendering recursion depth. Default: 1.

The following additional flags are supported for integers:

`$b`: Base, from 2 to 36. Default: 10.

`$s`: Sign. Legal values:

- `$-`: Only print sign if output is negative (default).
- `$+`: Always print sign.

The following additional flags are supported for reals:

`$d`: Digits of precision past decimal point. Default: 6.

`$e`: Exponential notation, if true. Default: false.

Table 2.9: outputsdict summary

Input(s) Op/Proc/Var Output(s)	Description
array flags **arraytype** string	Create formatted string from array.
boolean flags **booleantype** string	Create formatted string from boolean.
condition flags **conditiontype** string	Create formatted string from condition.

Continued on next page...

Table 2.9: *continued*

Input(s) Op/Proc/Var Output(s)	Description
dict flags **dicttype** string	Create formatted string from dict.
file flags **filetype** string	Create formatted string from file.
fino flags **finotype** string	Create formatted string from fino.
hook flags **hooktype** string	Create formatted string from hook.
integer flags **integertype** string	Create formatted string from integer.
mark flags **marktype** string	Create formatted string from mark.
mutex flags **mutextype** string	Create formatted string from mutex.
name flags **nametype** string	Create formatted string from name.
null flags **nulltype** string	Create formatted string from null.
operator flags **operatortype** string	Create formatted string from operator.
pmark flags **pmarktype** string	Create formatted string from pmark.
real flags **realtype** string	Create formatted string from real.
regex flags **regextype** string	Create formatted string from regex.
regsub flags **regsubtype** string	Create formatted string from regsub.

Continued on next page...

Table 2.9: *continued*

Input(s) Op/Proc/Var Output(s)	Description
stack flags **stacktype** string	Create formatted string from stack.
string flags **stringtype** string	Create formatted string from string.
thread flags **threadtype** string	Create formatted string from thread.

array flags **arraytype** ***string*****:**

Input(s):

array: An array object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *array*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *array*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> [1 [2 3] 4]
onyx:1> dup <$w 9 $p '_' $r 0> arraytype print '\n' print flush
__-array-
onyx:1> dup <$w 9 $p '_' $r 1> arraytype print '\n' print flush
[1 -array- 4]
onyx:1>
```

boolean flags **booleantype** ***string*****:**

Input(s):

boolean: A boolean object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *boolean*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *boolean*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> false
onyx:1> dup <$n 3> booleantype print '\n' print flush
fal
onyx:1> dup <$n 7> booleantype print '\n' print flush
false
onyx:1>
```

***condition flags* conditiontype *string*:**

Input(s):

condition: A condition object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *condition*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *condition*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> condition
onyx:1> <$w 15 $p '_' $j $c> booleantype print '\n' print flush
__-condition-__
onyx:0>
```

***dict flags* dicttype *string*:**

Input(s):

dict: A dict object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *dict*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *dict*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> <$foo 'foo'> <$w 30 $p '.' $j $r> dicttype print '\n' print flush
..................<$foo 'foo'>
onyx:0>
```

***file flags* filetype *string*:**

Input(s):

file: A file object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *file*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *file*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> stdin
onyx:1> <$w 30 $p ‘.’ $j $c> filetype print ‘\n’ print flush
............-file-...........
onyx:0>
```

***fino flags* finotype *string*:**

Input(s):

fino: A fino object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *fino*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *fino*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> (
onyx:1> <$w 30 $p ‘.’ $j $c> finotype print ‘\n’ print flush
............-fino-...........
onyx:0>
```

***hook flags* hooktype *string*:**

Input(s):

hook: A hook object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *hook*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *hook*.

Example(s): The following example is a bit contrived, since there is no way to create a hook object with a stock onyx interpreter. Therefore, imagine that an operator named taggedhook exists that creates a hook with a tag that is the name “tagged”.

```
onyx:0> outputsdict begin
onyx:0> taggedhook
onyx:1> <$w 30 $p ‘.’ $j $l hooktype print ‘\n’ print flush
=tagged=.....................
onyx:0>
```

***integer flags* integertype *string*:**

Input(s):

integer: An integer object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *integer*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *integer*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> 42 <$w 6 $p '_' $j $c $s $-> integertype print '\n' print flush
__42__
onyx:0> 42 <$w 6 $p '_' $j $c $s $+> integertype print '\n' print flush
_+42__
onyx:0> '0x' print 42 <$w 6 $p '0' $b 16> integertype print '\n' print flush
0x00002a
onyx:0>
```

***mark flags* marktype *string*:**

Input(s):

mark: A mark object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *mark*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *mark*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> mark
onyx:1> <$w 30 $p '.' $j $c> marktype print '\n' print flush
............-mark-............
onyx:0>
```

***mutex flags* mutextype *string*:**

Input(s):

mutex: A mutex object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *mutex*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *mutex*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> mutex
onyx:1> <$w 30 $p '.' $j $c> mutextype print '\n' print flush
...........-mutex-............
onyx:0>
```

***name flags* nametype *string*:**

Input(s):

name: A name object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *name*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *name*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> $foo
onyx:1> <$w 30 $p '.' $j $c> nametype print '\n' print flush
.............$foo.............
onyx:0>
```

***null flags* nulltype *string*:**

Input(s):

null: A null object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *null*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *null*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> null
onyx:1> <$w 30 $p '.' $j $c> nulltype print '\n' print flush
.............null.............
onyx:0>
```

***operator flags* operatortype *string*:**

Input(s):

operator: An operator object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *operator*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *operator*.

Example(s): The following example shows an operator printed out with two leading and trailing dashes. If the interpreter cannot determine the name associated with an operator, as will be the case for custom operators, the operator will be printed as `-operator-`.

```
onyx:0> outputsdict begin
onyx:0> ˜realtime
onyx:1> <$w 30 $p ‘.’ $j $c> operatortype print ‘\n’ print flush
.........--realtime--.........
onyx:0>
```

pmark flags **pmarktype** ***string*****:**

Input(s):

pmark: A pmark object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *pmark*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *pmark*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> { ˜x
Error $undefined
ostack: (-pmark- $x)
dstack: (-dict- -dict- -dict- -dict- -dict-)
estack/istack trace (0..1):
0:      -file-
1:      --start--
onyx:3> pop pop resume
onyx:1> <$w 30 $p ‘.’ $j $c> pmarktype print ‘\n’ print flush
...........-pmark-............
onyx:0>
```

real flags **realtype** ***string*****:**

Input(s):

real: A real object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *real*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *real*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> 6.022e23 <$d 4> realtype print ‘\n’ print flush
602200000000000027262976.0000
onyx:0> 6.022e23 <$d 4 $e true> realtype print ‘\n’ print flush
6.0220e+23
onyx:0> 6.022e23 <$d 0 $e true> realtype print ‘\n’ print flush
6e+23
onyx:0> 6.022e23 <$d 4 $w 40 $p ‘0’> realtype print ‘\n’ print flush
00000000000602200000000000027262976.0000
onyx:0>
```

***regex flags* regextype *string*:**

Input(s):

regex: A regex object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *regex*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *regex*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> ‘’ regex
onyx:1> <$w 30 $p ‘.’ $j $c> regextype print ‘\n’ print flush
...........-regex-............
onyx:0>
```

***regsub flags* regsubtype *string*:**

Input(s):

regsub: A regsub object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *regsub*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *regsub*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> ‘’ ‘’ regsub
```

```
onyx:1> <$w 30 $p '.' $j $c> regsubtype print '\n' print flush
...........-regsub-...........
onyx:0>
```

***stack flags* stacktype *string*:**

Input(s):

stack: A stack object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *stack*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *stack*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> (1 (2 3) 4)
onyx:1> dup <$w 9 $p '_' $r 0> stacktype print '\n' print flush
__-stack-
onyx:1> <$w 9 $p '_' $r 1> stacktype print '\n' print flush
(1 -stack- 4)
onyx:0>
```

***string flags* stringtype *string*:**

Input(s):

string: A string object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *string*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *string*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> 'A string'
onyx:1> <$w 30 $p '.' $j $c> stringtype print '\n' print flush
..........A string..........
onyx:0>
```

***thread flags* threadtype *string*:**

Input(s):

thread: A thread object.

flags: Formatting flags.

Output(s):

string: Formatted string representation of *thread*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *thread*.

Example(s):

```
onyx:0> outputsdict begin
onyx:0> () {} thread
onyx:1> <$w 30 $p '.' $j $c> threadtype print '\n' print flush
...........-thread-...........
onyx:0>
```

2.10.8 sprintsdict

The sprintsdict dictionary is primarily used to support **sprints**, but its contents may be of use to an application that wishes to extend or modify syntactical printing.

There is an entry in sprintsdict for each Onyx type. If there is a syntactically valid representation for an object and the recursion depth is greater than 0, the corresponding operator creates a string that syntactically represents the object. Otherwise, a string with a non-syntictical representation of the object is created, except for booleans, integers, names, nulls, reals, and strings, for which the results are always syntactical. If the recursion depth is greater than 0, the operators will recursively convert any contained objects.

The implementation of **sprints** is useful in illustrating a useful method of doing type-dependent operations:

```
$sprints {
        1 idup type $sprintsdict load exch get eval
} def
```

Table 2.10: sprintsdict summary

Input(s) Op/Proc/Var Output(s)	Description
array depth **arraytype** string	Create syntactical string from array.
boolean depth **booleantype** string	Create syntactical string from boolean.
condition depth **conditiontype** string	Create syntactical string from condition.
dict depth **dicttype** string	Create syntactical string from dict.

Continued on next page...

Table 2.10: *continued*

Input(s) Op/Proc/Var Output(s)	Description
file depth **filetype** string	Create syntactical string from file.
fino depth **finotype** string	Create syntactical string from fino.
hook depth **hooktype** string	Create syntactical string from hook.
integer depth **integertype** string	Create syntactical string from integer.
mark depth **marktype** string	Create syntactical string from mark.
mutex depth **mutextype** string	Create syntactical string from mutex.
name depth **nametype** string	Create syntactical string from name.
null depth **nulltype** string	Create syntactical string from null.
operator depth **operatortype** string	Create syntactical string from operator.
pmark depth **pmarktype** string	Create syntactical string from pmark.
real depth **realtype** string	Create syntactical string from real.
regex depth **regextype** string	Create syntactical string from regex.
regsub depth **regsubtype** string	Create syntactical string from regsub.
stack depth **stacktype** string	Create syntactical string from stack.

Continued on next page...

Table 2.10: *continued*

Input(s) Op/Proc/Var Output(s)	Description
string depth **stringtype** string	Create syntactical string from string.
thread depth **threadtype** string	Create syntactical string from thread.

***array depth* arraytype *string*:**

Input(s):

array: An array object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *array*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *array*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> [1 [2 3] 4]
onyx:1> dup 0 arraytype print ‘\n’ print flush
-array-
onyx:1> dup 1 arraytype print ‘\n’ print flush
[1 -array- 4]
onyx:1> dup 2 arraytype print ‘\n’ print flush
[1 [2 3] 4]
onyx:1>
```

***boolean depth* booleantype *string*:**

Input(s):

boolean: A boolean object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *boolean*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *boolean*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> true
onyx:1> dup 0 booleantype print ‘\n’ print flush
true
onyx:1>
```

***condition depth* conditiontype *string*:**

Input(s):

condition: A condition object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *condition*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *condition*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> condition
onyx:1> dup 0 conditiontype print ‘\n’ print flush
-condition-
onyx:1> dup 1 conditiontype print ‘\n’ print flush
-condition-
onyx:1>
```

***dict depth* dicttype *string*:**

Input(s):

dict: A dict object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *dict*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *dict*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> <$a ‘a’ $subdict <$b ‘b’>>
onyx:1> dup 0 dicttype print ‘\n’ print flush
-dict-
onyx:1> dup 1 dicttype print ‘\n’ print flush
<$subdict -dict- $a ‘a’>
onyx:1> dup 2 dicttype print ‘\n’ print flush
<$subdict <$b ‘b’> $a ‘a’>
onyx:1>
```

***file depth* filetype *string*:**

Input(s):

file: A file object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *file*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *file*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> stdout
onyx:1> dup 0 filetype print ‘\n’ print flush
-file-
onyx:1> dup 1 filetype print ‘\n’ print flush
-file-
onyx:1>
```

***fino depth* finotype *string*:**

Input(s):

fino: A fino object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *fino*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *fino*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> (
onyx:1> dup 0 finotype print ‘\n’ print flush
-fino-
onyx:1> dup 1 finotype print ‘\n’ print flush
-fino-
onyx:1>
```

***hook depth* hooktype *string*:**

Input(s):

hook: A hook object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *hook*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *hook*.

Example(s): The following example is a bit contrived, since there is no way to create a hook object with a stock onyx interpreter. Therefore, imagine that an operator named taggedhook exists that creates a hook with a tag that is the name “tagged”, and that an operator named untaggedhook exists that creates an untagged hook.

```
onyx:0> sprintsdict begin
onyx:0> taggedhook
onyx:1> dup 0 hooktype print ‘\n’ print flush
=tagged=
onyx:1> 1 hooktype print ‘\n’ print flush
=tagged=
onyx:0> untaggedhook
onyx:1> dup 0 hooktype print ‘\n’ print flush
-hook-
onyx:1> 1 hooktype print ‘\n’ print flush
-hook-
onyx:0>
```

***integer depth* integertype *string*:**

Input(s):

integer: An integer object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *integer*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *integer*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> 42
onyx:1> dup 0 integertype print ‘\n’ print flush
42
onyx:1> dup 1 integertype print ‘\n’ print flush
42
onyx:1>
```

***mark depth* marktype *string*:**

Input(s):

mark: A mark object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *mark*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *mark*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> mark
onyx:1> dup 0 marktype print '\n' print flush
-mark-
onyx:1> dup 1 marktype print '\n' print flush
-mark-
onyx:1>
```

***mutex depth* mutextype *string*:**

Input(s):

mutex: A mutex object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *mutex*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *mutex*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> mutex
onyx:1> dup 0 mutextype print '\n' print flush
-mutex-
onyx:1> dup 1 mutextype print '\n' print flush
-mutex-
onyx:1>
```

***name depth* nametype *string*:**

Input(s):

name: A name object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *name*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *name*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> $foo
onyx:1> dup 0 nametype print '\n' print flush
$foo
onyx:1> dup 1 nametype print '\n' print flush
$foo
onyx:1>
```

***null depth* nulltype *string*:**

Input(s):

null: A null object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *null*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *null*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> null
onyx:1> dup 0 nulltype print ‘\n’ print flush
-null-
onyx:1> dup 1 nulltype print ‘\n’ print flush
-null-
onyx:1>
```

***operator depth* operatortype *string*:**

Input(s):

operator: An operator object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *operator*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *operator*.

Example(s): The following example shows an operator printed out with two leading and trailing dashes. If the interpreter cannot determine the name associated with an operator, as will be the case for custom operators, the operator will be printed as `-operator-`.

```
onyx:0> sprintsdict begin
onyx:0> ˜realtime
onyx:1> dup 0 operatortype print ‘\n’ print flush
--realtime--
onyx:1> 1 operatortype print ‘\n’ print flush
--realtime--
onyx:0>
```

***pmark depth* pmarktype *string*:**

Input(s):

pmark: A pmark object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *pmark*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *pmark*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> { ˜x
Error $undefined
ostack: (-pmark- $x)
dstack: (-dict- -dict- -dict- -dict- -dict-)
estack/istack trace (0..1):
0:      -file-
1:      --start--
onyx:3> pop pop resume
onyx:1> dup 0 pmarktype print ‘\n’ print flush
-pmark-
onyx:1> dup 1 pmarktype print ‘\n’ print flush
-pmark-
onyx:1>
```

***regex depth* regextype *string*:**

Input(s):

regex: A regex object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *regex*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *regex*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> ‘’ regex
onyx:1> dup 0 regextype print ‘\n’ print flush
-regex-
onyx:1> dup 1 regextype print ‘\n’ print flush
-regex-
onyx:1>
```

***regsub depth* regsubtype *string*:**

Input(s):

regsub: A regsub object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *regsub*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *regsub*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> `' `' regsub
onyx:1> dup 0 regsubtype print `\n' print flush
-regsub-
onyx:1> dup 1 regsubtype print `\n' print flush
-regsub-
onyx:1>
```

real depth **realtype** ***string*:**

Input(s):

real: A real object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *real*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *real*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> 42.0
onyx:1> dup 0 realtype print `\n' print flush
4.200000e+01
onyx:1> dup 1 realtype print `\n' print flush
4.200000e+01
onyx:1>
```

stack depth **stacktype** ***string*:**

Input(s):

stack: A stack object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *stack*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *stack*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> (1 (2 3) 4)
onyx:1> dup 0 stacktype print `\n' print flush
-stack-
```

```
onyx:1> dup 1 stacktype print ‘\n’ print flush
(1 -stack- 4)
onyx:1> dup 2 stacktype print ‘\n’ print flush
(1 (2 3) 4)
onyx:1>
```

***string depth* stringtype *string*:**

Input(s):

string: A string object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *string*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *string*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> ‘abcd’
onyx:1> dup 0 stringtype print ‘\n’ print flush
‘abcd’
onyx:1> dup 1 stringtype print ‘\n’ print flush
‘abcd’
onyx:1>
```

***thread depth* threadtype *string*:**

Input(s):

thread: A thread object.

depth: Recursion depth.

Output(s):

string: Syntactical string representation of *thread*.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *thread*.

Example(s):

```
onyx:0> sprintsdict begin
onyx:0> thread
onyx:1> dup 0 threadtype print ‘\n’ print flush
-thread-
onyx:1> dup 1 threadtype print ‘\n’ print flush
-thread-
onyx:1>
```

2.10.9 systemdict

The systemdict dictionary contains most of the operators that are of general use. Although there are no mechanisms that prevent modification of systemdict, programs should not normally need to modify systemdict, since globaldict provides a place for storing globally shared objects. All threads share the same systemdict, which is implicitly locked.

Table 2.11 summarizes the contents of systemdict, and is broken into the following categories:

- Operand stack operators
- Execution, control, and execution stack operators
- Stack operators
- Number (integer, real) and math operators
- String operators
- Name operators
- Array operators
- Dictionary and dictionary stack operators
- File and filesystem operators
- Socket and networking operators
- Logical and bitwise operators
- Type, conversion, and attribute operators
- Threading and synchronization operators
- Regular expression operators
- Miscellaneous operators

Table 2.11: systemdict summary

Input(s) Op/Proc/Var Output(s)	Description
Operand stack operators	
– **mark** mark	Create a mark.
,,,obj **aup** obj ,,,	Rotate stack up one position.
obj ,,, **adn** ,,,obj	Rotate stack down one position.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
– **count** count	Get the number of objects on ostack.
mark . . . **counttomark** mark . . . count	Get the depth of the topmost mark on ostack.
obj **dup** obj dup	Duplicate an object.
obj ,,, **bdup** obj ,,,dup	Duplicate bottom object.
objects count **ndup** objects objects	Duplicate objects.
obj . . . index **idup** obj . . . dup	Duplicate object on ostack at index.
. . . obj ,,,index **ibdup** . . . obj ,,,dup	Duplicate object on ostack at index from bottom.
a b **tuck** b a b	Tuck duplicate of top object under second object.
a b **under** a a b	Duplicate second object.
a b **over** a b a	Duplicate second object.
a b **exch** b a	Exchange top two objects.
a b c **up** c a b	Roll top three objects up one.
a . . . b count **nup** b a . . .	Roll count objects up one.
a b c **dn** b c a	Roll top three objects down one.
a . . . b count **ndn** . . . b a	Roll count objects down one.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
. . . amount **rot** . . .	Rotate stack up by *amount*.
region count amount **roll** rolled	Roll *count* objects up by *amount*.
obj **pop** –	Remove object.
obj ,,, **bpop** ,,,	Remove bottom object.
objects count **npop** –	Remove count objects.
objects . . . count **nbpop** . . .	Remove count objects from bottom.
obj . . . index **ipop** . . .	Remove object at index.
. . . obj ,,,index **ibpop** . . . ,,,	Remove object at index from bottom.
a b **nip** b	Remove second object.
objects **clear** –	Pop all objects off ostack.
mark . . . **cleartomark** –	Remove objects from ostack through topmost mark.
– **ostack** stack	Get a current ostack snapshot.
thread **threadostack** stack	Get a reference to thread's ostack.
Execution, control, and execution stack operators	
obj **eval** –	Evaluate object.
boolean obj **if** –	Conditionally evaluate object.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
boolean obj **unless** –	Conditionally evaluate object.
boolean a b **ifelse** –	Conditionally evaluate one of two objects.
init inc limit proc **for** –	Iterate with a control variable.
count proc **repeat** –	Iterate a set number of times.
cond proc **while** –	Loop while cond is true.
proc cond **until** –	Loop until cond is false.
proc **loop** –	Loop indefinitely.
array proc **foreach** –	Iterate on array elements.
dict proc **foreach** –	Iterate on dictionary key/value pairs.
stack proc **foreach** –	Iterate on stack elements.
string proc **foreach** –	Iterate on string elements.
– **exit** –	Terminate innermost looping context.
file/string **token** false	Unsuccessfully scan for a token.
file/string **token** rem obj true	Successfully scan for a token
obj **start** –	Evaluate object.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
– **quit** –	Unwind to innermost start context.
obj **stopped** boolean	Evaluate object.
– **stop** –	Unwind to innermost stopped or start context.
name **throw** obj	Throw an error.
– **estack** stack	Get a current estack snapshot.
thread **threadestack** stack	Get a reference to thread's estack.
– **countestack** count	Get current estack depth.
– **istack** stack	Get a current istack snapshot.
thread **threadistack** stack	Get a reference to thread's istack.
status **die** –	Exit program.
path symbol **modload** –	Load a module.
file symbol **mrequire** –	Search for and load a module.
file **require** –	Search for and evaluate a source file.
args **exec** –	Overlay a new program and execute it.
args **forkexec** pid	Fork and exec a new process.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
pid **waitpid** status	Wait for a program to terminate.
args **system** status	Execute a program.
– **pid** pid	Get process ID.
– **ppid** pid	Get parent's process ID.
– **uid** uid	Get the process's user ID.
uid **setuid** boolean	Set the process's user ID.
– **euid** uid	Get the process's effective user ID.
uid **seteuid** boolean	Set the process's effective user ID.
– **gid** gid	Get the process's group ID.
gid **setgid** boolean	Set the process's group ID.
– **egid** gid	Get the process's effective group ID.
gid **setegid** boolean	Set the process's effective group ID.
– **realtime** nsecs	Get the number of nanoseconds since the epoch.
nsecs **localtime** dict	Get a dict with local time definitions.
nanoseconds **nsleep** –	Nanosleep.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
Stack operators	
– (fino	Begin a stack declaration.
fino objects) stack	Create a stack.
– **stack** stack	Create a stack.
stack obj **spush** –	Push object onto stack.
stack obj **sbpush** –	Push object onto bottom of stack.
stack **scount** count	Get the number of objects on a stack.
stack **scounttomark** count	Get the depth of the topmost mark on stack.
stack **sdup** –	Duplicate an object.
stack **sbdup** –	Duplicate bottom object.
stack count **sndup** –	Duplicate objects on stack.
stack index **sidup** –	Duplicate object on stack at index.
stack index **sibdup** –	Duplicate object on stack at index from bottom.
stack **stuck** –	Tuck duplicate of top object on stack under next object on stack.
stack **sunder** –	Duplicate second object on stack.
stack **sover** –	Duplicate second object on stack.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
stack **sexch** –	Exchange top objects on stack.
stack **sup** –	Roll top three objects on stack up one.
stack count **snup** –	Roll count objects on stack up one.
stack **saup** –	Roll objects on stack up one.
stack **sdn** –	Roll top three objects on stack down one.
stack count **sndn** –	Roll count objects on stack down one.
stack **sadn** –	Roll objects on stack down one.
stack amount **srot** –	Rotate objects on stack up by *amount*.
stack count amount **sroll** –	Roll objects on stack.
stack **spop** obj	Pop object off stack.
stack **sbpop** obj	Pop object off bottom of stack.
stack count **snpop** array	Pop count objects off stack.
stack count **snbpop** array	Pop count objects off bottom of stack.
stack index **sipop** obj	Remove object on stack at index.
stack index **sibpop** obj	Remove object on stack at index from bottom.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
stack **snip** obj	Remove second object on stack.
stack **sclear** –	Remove all objects on stack.
stack **scleartomark** –	Remove objects from stack down through topmost mark.
(a) (b) **cat** (a b)	Catenate two stacks.
stacks count **ncat** stack	Catenate stacks.
srcstack dststack **copy** dststack	Copy stack contents.
Number (integer, real) and math operators	
a b **add** r	Add a and b.
a **inc** r	Add 1 to a.
a b **sub** r	Subtract b from a.
a **dec** r	Subtract 1 from a.
a b **mul** r	Multiply a and b.
a b **div** r	Divide a by b.
a b **idiv** r	Divide a by b (integers).
a b **mod** r	Mod a by b (integers and reals).
a b **exp** r	Raise a to the power of b.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
x **exp** r	e (base of natural logarithm) raised to x.
a **sqrt** r	Square root.
a **ln** r	Natural log.
a **log** r	Base 10 log.
a **abs** r	Get the absolute value of a.
a **neg** r	Get the negative of a.
a **ceiling** r	Integer ceiling of a real.
a **floor** r	Integer floor of a real.
a **round** r	Real rounded to integer.
a **trunc** r	Integer from real with truncated fractional.
a **sin** r	Sine in radians.
a **sinh** r	Hyperbolic sine.
a **asin** r	Arcsine.
a **asinh** r	Hyperbolic arcsine.
a **cos** r	Cosine in radians.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
a **cosh** r	Hyperbolic cosine.
a **acos** r	Arc cosine.
a **acosh** r	Hyperbolic arc cosine.
x **tan** r	Tangent of x in radians.
x **tanh** r	Hyperbolic tangent.
x **atan** r	Arctangent.
y x **atan2** r	Arctangent in radians of $\frac{y}{x}$.
x **atanh** r	Hyperbolic arctangent.
seed **srand** –	Seed pseudo-random number generator.
– **rand** integer	Get a pseudo-random number.
String operators	
length **string** string	Create a string.
string **length** count	Get string length.
string index **get** integer	Get string element.
string index integer **put** –	Set string element.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
string index length **getinterval** substring	Get a string interval.
string index substring **putinterval** –	Copy substring into string.
‘a’ ‘b’ **cat** ‘ab’	Catenate two strings.
strings count **ncat** string	Catenate strings.
srcstring dststring **copy** dstsubstring	Copy string.
obj depth **sprints** string	Create syntactical string from object.
obj flags **outputs** string	Create formatted string from object.
string pattern **search** post pattern pre true string pattern **search** string false	Successfully search for pattern. Unsuccessfully earch for pattern.
Name operators	
name **length** count	Get name length.
Array operators	
– **argv** args	Get program arguments.
– **[** mark	Begin an array declaration.
mark objects **]** array	Construct an array.
length **array** array	Create an array.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
array **length** count	Get array length.
array index **get** obj	Get array element.
array index obj **put** –	Set array element.
array index length **getinterval** subarray	Get an array interval.
array index subarray **putinterval** –	Copy subarray into array.
[a] [b] **cat** [a b]	Catenate two arrays.
arrays count **ncat** array	Catenate arrays.
srcarray dstarray **copy** dstsubarray	Copy array.
Dictionary and dictionary stack operators	
– **gcdict** dict	Get gcdict.
– **userdict** dict	Get userdict.
– **globaldict** dict	Get globaldict.
– **systemdict** dict	Get systemdict.
– **onyxdict** dict	Get onyxdict.
– **sprintsdict** dict	Get sprintsdict.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
– **outputsdict** dict	Get outputsdict.
– **envdict** dict	Get envdict.
– **threadsdict** dict	Get threadsdict.
key val **setenv** –	Set environment variable.
key **unsetenv** –	Unset environment variable.
– < mark	Begin a dictionary declaration.
mark kvpairs > dict	Construct a dictionary.
– **dict** dict	Create a dictionary.
dict **begin** –	Pust dict onto dstack.
– **end** –	Pop a dictionary off dstack.
key val **def** –	Define key/value pair.
dict key **undef** –	Undefine key in dict.
key **load** val	Look up a key's value.
dict key **known** boolean	Check for key in dict.
key **where** false	Unsuccessfully get topmost dstack dictionary that defines key.

Continued on next page...

Table 2.11: *continued*

Input(s) / Op/Proc/Var / Output(s)	Description
key **where** dict true	Successfully get topmost dstack dictionary that defines key.
dict **length** count	Get number of dictionary key/value pairs.
dict key **get** value	Get dict value associate with key.
dict key value **put** –	Set dict key/value pair.
srcdict dstdict **copy** dstdict	Copy dictionary contents.
– **currentdict** dict	Get topmost dstack dictionary.
– **dstack** stack	Get dstack snapshot.
thread **threaddstack** stack	Get a reference to thread's dstack.
– **countdstack** count	Get number of stacks on dstack.
File and filesystem operators	
filename flags **open** file	Open a file.
filename flags mode **open** file	Open a file, creation mode specified.
– **pipe** rfile wfile	Create a pipe.
file **close** –	Close file.
file **read** integer boolean	Read from file.
file string **read** substring boolean	Read from file.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
file **readline** string boolean	Read a line from file.
<file dict . . . > timeout **poll** [file . . .]	Wait for file(s) to change status.
file **bytesavailable** count	Get number of buffered readable bytes.
file **iobuf** count	Get size of I/O buffer.
file count **setiobuf** –	Set size of I/O buffer.
file **nonblocking** boolean	Get non-blocking mode.
file boolean **setnonblocking** –	Set non-blocking mode.
file integer/string **write** false file integer/string **write** integer/substring true	Write to file. Write to file.
string **print** –	Print string to stdout.
obj depth **sprint** –	Syntactically print object to stdout.
obj flags **output** –	Formatted print to stdout.
– **pstack** –	Syntactically print ostack elements.
file **flushfile** –	Flush file buffer.
– **flush** –	Flush stdout buffer.

Continued on next page...

Table 2.11: *continued*

Input(s) / Op/Proc/Var / Output(s)	Description
file length **truncate** –	Truncate file.
file offset **seek** –	Move file position pointer.
file **tell** offset	Get file position pointer offset.
path **mkdir** –	Create a directory.
path mode **mkdir** –	Create a directory, mode specified.
path **mkfifo** –	Create a named pipe.
path mode **mkfifo** –	Create a named pipe, mode specified.
old new **rename** –	Rename a file or directory.
file/filename mode **chmod** –	Change file permissions.
file/filename uid gid **chown** –	Change file owner and group.
filename linkname **link** –	Create a hard link.
filename linkname **symlink** –	Create a symbolic link.
filename **unlink** –	Unlink a file.
path **rmdir** –	Remove an empty directory.
file/filename flag **test** boolean	Test a file.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
file/filename **status** dict	Get file information.
linkname **readlink** string	Get symbolic link data.
path proc **dirforeach** –	Iterate on directory entries.
– **pwd** path	Get present working directory.
path **cd** –	Change present working directory.
path **chroot** –	Change root directory.
– **stdin** file	Get thread's stdin.
– **stdout** file	Get thread's stdout.
– **stderr** file	Get thread's stderr.
– **gstdin** file	Get global stdin.
– **gstdout** file	Get global stdout.
– **gstderr** file	Get global stderr.
file **setstdin** –	Set thread's stdin.
file **setstdout** –	Set thread's stdout.
file **setstderr** –	Set thread's stderr.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
file **setgstdin** –	Set global stdin.
file **setgstdout** –	Set global stdout.
file **setgstderr** –	Set global stderr.
Socket and networking operators	
family type proto **socket** sock	Create a socket.
family type **socket** sock	Create a socket.
sock addr port **bindsocket** –	Bind socket to address/port.
sock addr **bindsocket** –	Bind socket to address.
sock path **bindsocket** –	Bind socket to port.
sock backlog **listen** –	Listen for socket connections.
sock **listen** –	Listen for socket connections.
sock **accept** sock	Accept a socket connection.
sock addr port **connect** –	Connect a socket.
sock path **connect** –	Connect a socket.
service **serviceport** port	Get port number for service name.
sock **sockname** dict	Get socket information.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
sock level optname **sockopt** optval	Get socket option.
sock optname **sockopt** optval	Get socket option.
sock level optname optval **setsockopt** –	Set socket option.
sock optname optval **setsockopt** –	Set socket option.
sock **peername** dict	Get peer socket information.
sock mesg flags **send** nsend	Send a message.
sock mesg **send** count	Send a message.
sock string flags **recv** substring	Receive a message.
sock string **recv** substring	Receive a message.
family type proto **socketpair** sock sock	Create a socket pair.
family type **socketpair** sock sock	Create a socket pair.
Logical and bitwise operators	
a b **lt** boolean	a less than b? (integer/real, string)
a b **le** boolean	a less than or equal to b? (integer/real, string)
a b **eq** boolean	a equal to b? (any type)

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
a b **ne** boolean	a not equal to b? (any type)
a b **ge** boolean	a greater than or equal to b? (integer/real, string)
a b **gt** boolean	a greater than b? (integer/real, string)
a b **and** r	Logical/bitwise and. (boolean/integer)
a b **or** r	Logical/bitwise or. (boolean/integer)
a b **xor** r	Logical/bitwise exclusive or. (boolean/integer)
a **not** r	Logical/bitwise not. (boolean/integer)
a shift **shift** integer	Bitwise shift.
– **false** false	Return true.
– **true** true	Return false.
Type, conversion, and attribute operators	
obj **type** name	Get object type.
obj **echeck** boolean	Evaluatable?
obj **xcheck** boolean	Executable?
obj **cve** obj	Set evaluatable attribute.

Continued on next page...

Table 2.11: *continued*

Input(s) / Op/Proc/Var / Output(s)	Description
obj **cvx** obj	Set executable attribute.
obj **cvlit** obj	Set literal attribute.
string **cvn** name	Convert string to name.
obj **cvs** string	Convert object to string.
integer radix **cvrs** string	Convert integer to radix string.
real precision **cvds** string	Convert real to decimal string.
real precision **cves** string	Convert real to exponential string.
hook **hooktag** tag	Get hook tag.
Threading and synchronization operators	
stack entry **thread** thread	Create and run a thread.
– **self** thread	Get a thread object for the running thread.
thread **join** –	Wait for thread to exit.
thread **detach** –	Detach thread.
– **yield** –	Voluntarily yield the processor.
– **mutex** mutex	Create a mutex.

Continued on next page...

Table 2.11: *continued*

Input(s) / Op/Proc/Var / Output(s)	Description
mutex proc **monitor** –	Evaluate an object under the protection of a mutex.
mutex **lock** –	Acquire mutex.
mutex **trylock** boolean	Try to acquire mutex.
mutex **unlock** –	Release mutex.
– **condition** condition	Create a condition variable.
condition mutex **wait** –	Wait on condition.
condition mutex timeout **timedwait** boolean	Wait on condition with timeout.
condition **signal** –	Signal a condition waiter.
condition **broadcast** –	Signal all condition waiters.
– **currentlocking** boolean	Get implicit locking mode.
boolean **setlocking** –	Set implicit locking mode.
obj **lcheck** boolean	Implicitly locked?
Regular expression operators	
string flags **regex** regex string **regex** regex	Create a regex object. Create a regex object.
input pattern flags **match** boolean	Find pattern matches in input string.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
input pattern **match** boolean	Find pattern matches in input string.
input regex **match** boolean	Find regex matches in input string.
input pattern flags limit **split** array	Split input into an array of substrings.
input pattern flags **split** array	Split input into an array of substrings.
input pattern limit **split** array	Split input into an array of substrings.
input pattern **split** array	Split input into an array of substrings.
input regex limit **split** array	Split input into an array of substrings.
input regex **split** array	Split input into an array of substrings.
integer **submatch** substring	Get capturing subpattern match.
input submatch **offset** offset	Get submatch offset from beginning of input.
pattern template flags **regsub** regsub	Create a regsub object.
pattern template **regsub** regsub	Create a regsub object.
input pattern template flags **subst** output count	Substitute template for pattern matches.
input pattern template **subst** output count	Substitute template for pattern matches.
input regsub **subst** output count	Substitute.

Continued on next page...

Table 2.11: *continued*

Input(s) Op/Proc/Var Output(s)	Description
Miscellaneous operators	
– **product** string	Get the product string.
– **version** string	Get the version string.
proc **bind** proc	Bind names to operators.
– **null** null	Create a null object.

– (*fino*:

Input(s): None.

Output(s):

fino: A fino object.

Errors(s): None.

Description: Push a fino object onto ostack to denote the bottom of a stack that has not yet been constructed.

Example(s):

```
onyx:0> (
onyx:1> pstack
-fino-
onyx:1>
```

***fino objects*) *stack*:**

Input(s):

fino: A fino object, usually created by the **)** operator.

objects: 0 or more objects.

Output(s):

stack: A stack object.

Errors(s):

unmatchedfino.

Description: Create a stack object and move all objects from ostack down to the first fino object to the new stack.

Example(s):

```
onyx:0> ()
onyx:1> 1 sprint
```

```
()
onyx:0> (1 2
onyx:3> pstack
2
1
-fino-
onyx:3> )
onyx:1> 1 sprint
(1 2)
onyx:0>
```

– < *mark*:

Input(s): None.

Output(s):

mark: A mark object.

Errors(s): None.

Description: Begin a dictionary declaration. See the **¿** operator documentation for more details on dictionary construction.

Example(s):

```
onyx:0> < 1 sprint
-mark-
onyx:0>
```

***mark kvpairs* > *dict*:**

Input(s):

mark: A mark object.

kvpairs: Zero or more pairs of non-mark objects, where the first is a key and the second is an associated value.

Output(s):

dict: A dictionary that contains *kvpairs*.

Errors(s):

rangecheck.

unmatchedmark.

Description: Construct a dictionary that contains *kvpairs*.

Example(s):

```
onyx:0> <
onyx:1> $foo 'foo'
onyx:3> $bar 'bar'
onyx:5> $biz 'biz'
onyx:7> $pop ~pop
onyx:9> >
onyx:1> pstack
<$pop --pop-- $biz 'biz' $bar 'bar' $foo 'foo'>
onyx:1>
```

– [*mark*:

Input(s): None.

Output(s):

mark: A mark object.

Errors(s): None.

Description: Begin an array declaration. See the **]** operator documentation for more details on array construction.

Example(s):

```
onyx:0> [ 1 sprint
-mark-
onyx:0>
```

***mark objects*] *array*:**

Input(s):

mark: A mark object.

objects: Zero or more non-mark objects.

Output(s):

array: An array that contains *objects*.

Errors(s):

unmatchedmark.

Description: Construct an array that contains all *objects* on ostack down to the first *mark*.

Example(s):

```
onyx:0> mark 1 2 3 ] 1 sprint
[1 2 3]
```

***a* abs *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Absolute value of a.

Errors(s):

stackunderflow.

typecheck.

Description: Return the absolute value of a.

Example(s):

```
onyx:0> 5 abs 1 sprint
5
onyx:0> -5 abs 1 sprint
5
onyx:0> 3.14 abs 1 sprint
3.140000e+00
onyx:0> -3.14 abs 1 sprint
3.140000e+00
onyx:0>
```

***sock* accept *sock*:**

Input(s):

sock: A listening socket.

Output(s):

sock: A socket that is connected to a client.

Errors(s):

argcheck.

invalidfileaccess.

ioerror.

neterror.

stackunderflow.

typecheck.

unregistered.

Description: Accept a connection and create a socket that is connected to a client.

Example(s):

```
onyx:0> $AF_INET $SOCK_STREAM socket
onyx:1> dup ‘localhost’ 7777 bindsocket
onyx:1> dup listen
onyx:1> dup accept
onyx:2> dup peername 1 sprint
<$family $AF_INET $address 2130706433 $port 33742>
onyx:2>
```

***a* acos *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Arc cosine of a in radians.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Return the arc cosine of a in radians.

Example(s):

```
onyx:0> 1 acos 1 sprint
0.000000e+00
onyx:0>
```

***a* acosh *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Hyperbolic arc cosine of a.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Return the hyperbolic arc cosine of a.

Example(s):

```
onyx:0> 10 acosh 1 sprint
2.993223e+00
onyx:0>
```

***a b* add *r*:**

Input(s):

a: An integer or real.

b: An integer or real.

Output(s):

r: The sum of a and b.

Errors(s):

stackunderflow.

typecheck.

Description: Return the sum of a and b.

Example(s):

```
onyx:0> 2 2 add 1 sprint
4
onyx:0> -1 3 add 1 sprint
2
onyx:0> 2.0 3.1 add 1 sprint
5.100000e+00
onyx:0> -1.5 +3e1 add 1 sprint
2.850000e+01
onyx:0>
```

***obj* ,,, adn ,,,*obj*:**

Input(s):

obj: An object.

,,,: Zero or more objects.

Output(s):

,,,: Zero or more objects.

obj: An object.

Errors(s):

stackunderflow.

Description: Rotate stack down one position.

Example(s):

```
onyx:0> 1 2 3 adn pstack
1
3
2
onyx:3>
```

***a b* and *r*:**

Input(s):

a: An integer or boolean.

b: The same type as a.

Output(s):

r: If *a* and *b* are integers, their bitwise and, otherwise their logical and.

Errors(s):

stackunderflow.

typecheck.

Description: Return the bitwise and of two integers, or the logical and of two booleans.

Example(s):

```
onyx:0> false true and 1 sprint
false
onyx:0> true true and 1 sprint
true
onyx:0> 5 3 and 1 sprint
1
onyx:0>
```

– argv *args*:

Input(s): None.

Output(s):

args: An array of strings. The first string in *args* is the path of this program, and any additional array elements are the arguments that were passed during invocation.

Errors(s): None.

Description: Get the argument vector that was used to invoke this program.

Example(s):

```
onyx:0> argv 1 sprint
['/usr/local/bin/onyx']
onyx:0>
```

***length* array *array*:**

Input(s):

length: Non-negative number of array elements.

Output(s):

array: An array of *length* elements.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Create an array of *length* elements. The elements are initialized to null objects.

Example(s):

```
onyx:0> 3 array 1 sprint
[null null null]
onyx:0> 0 array 1 sprint
[]
onyx:0>
```

***a* asin *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Arc sine of a in radians.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Return the arc sine of a in radians.

Example(s):

```
onyx:0> -1 asin 1 sprint
-1.570796e+00
onyx:0>
```

***a* asinh *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Hyperbolic arc sine of a.

Errors(s):

stackunderflow.

typecheck.

Description: Return the hyperbolic arc sine of a.

Example(s):

```
onyx:0> 10 asinh 1 sprint
2.998223e+00
onyx:0>
```

***x* atan *r*:**

Input(s):

x: An integer or real.

Output(s):

r: Arctangent of x in radians.

Errors(s):

stackunderflow.

typecheck.

Description: Return the arctangent of x in radians.

Example(s):

```
onyx:0> 1 atan 1 sprint
7.853982e-01
onyx:0>
```

***y x* atan2 *r*:**

Input(s):

y: An integer or real.

x: An integer or real.

Output(s):

r: Arctangent of $\frac{y}{x}$ in radians.

Errors(s):

stackunderflow.

typecheck.

Description: Return the arctangent of $\frac{y}{x}$ in radians.

Example(s):

```
onyx:0> 1 1 atan2 1 sprint
7.853982e-01
onyx:0> 0 1 atan2 1 sprint
0.000000e+00
onyx:0> -1.0 0 atan2 1 sprint
-1.570796e+00
onyx:0>
```

***x* atanh *r*:**

Input(s):

x: An integer or real.

Output(s):

r: Hyperbolic arctangent of x.

Errors(s):

stackunderflow.

typecheck.

rangecheck.

Description: Return the hyperbolic arctangent of x.

Example(s):

```
onyx:0> 0.5 atanh 1 sprint
5.493061e-01
onyx:0>
```

***,,,obj* aup *obj ,,,*:**

Input(s):

,,,: Zero or more objects.

obj: An object.

Output(s):

obj: An object.

,,,: Zero or more objects.

Errors(s):

stackunderflow.

Description: Rotate stack up one position.

Example(s):

```
onyx:0> 1 2 3 aup pstack
2
1
3
onyx:3>
```

obj ,,, ***bdup obj ,,,dup***:

Input(s):

obj: An object.

,,,: Zero or more objects.

Output(s):

obj: An object.

,,,: Zero or more objects.

dup: A duplicate of *obj*.

Errors(s):

stackunderflow.

Description: Create a duplicate of the bottom object on ostack and put it on top of ostack.

Example(s):

```
onyx:0> 1 2 3
onyx:3> bdup pstack
1
3
2
1
onyx:4>
```

***dict* begin –:**

Input(s):

dict: A dictionary.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Push *dict* onto dstack, thereby adding its keys to the namespace.

Example(s):

```
onyx:0> <$foo 'foo'> begin
onyx:0> foo 1 sprint
'foo'
onyx:0>
```

***proc* bind *proc*:**

Input(s):

proc: A procedure (array). *proc* will be bound even if it is literal, but contained literal arrays will not be recursively bound.

Output(s):

proc: The same procedure as was passed in.

Errors(s):

stackunderflow.

typecheck.

Description: Recursively bind unbound procedures. Executable names within a procedure are replaced with their values if defined in dstack, in any of the following cases:

- The value is a literal object.
- The value is an executable or evaluatable operator.
- The value is an executable or evaluatable hook.
- The value is an executable or evaluatable array.

Binding has a large positive impact on performance, since name lookups are thereafter avoided. However, binding is not done by default because there are situations where it is useful to leave procedures unbound:

- Debugging is easier, since the names associated with objects are still available.
- Behavior is more dynamic. It is possible to replace a definition on dstack and have it immediately take effect on unbound procedures. Note however that care must be taken when relying on this, since binding is recursive, and a lack of complete understanding of what procedures reference each other can result in undesired bound procedures. For this reason, it is generally best to make dynamic behavior explicit by using evaluatable names.
- There are situations where a program needs to do some setup before binding a procedure, and providing manual control over when binding happens allows more sophisticated use of binding.

Example(s):

```
onyx:0> {pop sprint {pop sprint}}
onyx:1> dup 2 sprint
{pop sprint {pop sprint}}
onyx:1> bind
onyx:1> dup 2 sprint
{--pop-- _{sprints --print-- ‘\n’ --print-- --flush--}_ {--pop-- -array-}}
onyx:1>
```

sock addr port **bindsocket –:**

sock addr **bindsocket –:**

sock path **bindsocket –:**

Input(s):

sock: A socket.

addr: An IPv4 address or DNS hostname.

port: An IPv4 port number. If not specified, the OS chooses a port number.

path: A filesystem path for a Unix-domain socket.

Output(s): None.

Errors(s):

argcheck.

invalidfileaccess.

neterror.

rangecheck.

stackunderflow.

typecheck.

unregistered.

Description: Bind an address/port to an IPv4 socket, or a filesystem path to a Unix-domain socket.

Example(s):

```
onyx:0> $AF_INET $SOCK_STREAM socket
onyx:1> dup 'localhost' 7777 bindsocket
onyx:1> dup sockname 1 sprint
<$family $AF_INET $address 2130706433 $port 7777>
onyx:1> close
onyx:0> $AF_LOCAL $SOCK_STREAM socket
onyx:1> dup '/tmp/socket' bindsocket
onyx:1> dup sockname 1 sprint
<$family $AF_LOCAL $path '/tmp/socket'>
onyx:1>
```

***obj* ,,, bpop ,,,:**

Input(s):

,,,: Zero or more objects.

obj: An object.

Output(s):

,,,: Zero or more objects.

Errors(s):

stackunderflow.

Description: Remove the bottom object from ostack and discard it.

Example(s):

```
onyx:0> 1 2
onyx:2> bpop pstack
2
onyx:1>
```

***condition* broadcast –:**

Input(s):

condition: A condition object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Signal all threads that are waiting on *condition*. If there are no waiters, this operator has no effect.

Example(s):

```
onyx:0> condition mutex dup lock ostack
onyx:3> {dup lock exch broadcast unlock}
onyx:4> thread 3 1 roll
onyx:3> dup 3 1 roll
onyx:4> wait unlock join
onyx:0>
```

***file* bytesavailable *count*:**

Input(s):

file: A file object.

Output(s):

count: Number of buffered readable bytes.

Errors(s):

stackunderflow.

typecheck.

Description: Get the number of buffered readable bytes that can be read without the possibility of blocking.

Example(s):

```
onyx:0> '/tmp/foo' 'w+' open
onyx:1> dup 'Hello\n' write
onyx:1> dup 'Goodbye\n' write
onyx:1> dup 0 seek
onyx:1> dup readline 1 sprint 1 sprint
false
'Hello'
onyx:1> dup bytesavailable 1 sprint
8
onyx:1>
```

***[a] [b]* cat *[a b]*:**

***(a) (b)* cat *(a b)*:**

***'a' 'b'* cat *'ab'*:**

Input(s):

a: An array, stack, or string.

b: An array, stack, or string.

Output(s):

ab: The catenation of a and b.

Errors(s):

stackunderflow.

typecheck.

Description: Catenate two arrays, strings, or stacks.

Example(s):

```
onyx:0> ['a'] ['b'] cat
onyx:1> 1 sprint
['a' 'b']
onyx:0> ('a') ('b') cat
onyx:1> 1 sprint
('a' 'b')
onyx:0> 'a' 'b' cat
onyx:1> 1 sprint
'ab'
onyx:0>
```

***path* cd –:**

Input(s):

path: A string that represents a filesystem path.

Output(s): None.

Errors(s):

invalidaccess.

ioerror.

stackunderflow.

typecheck.

Description: Change the present working directory to *path*.

Example(s):

```
onyx:0> pwd 1 sprint
'/usr/local'
onyx:0> 'bin' cd
onyx:0> pwd 1 sprint
'/usr/local/bin'
onyx:0>
```

***a* ceiling *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Integer ceiling of a.

Errors(s):

stackunderflow.

typecheck.

Description: Return the integer ceiling of a.

Example(s):

```
onyx:0> -1.51 ceiling 1 sprint
-1
onyx:0> -1.49 ceiling 1 sprint
-1
onyx:0> 0 ceiling 1 sprint
0
onyx:0> 1.49 ceiling 1 sprint
2
onyx:0> 1.51 ceiling 1 sprint
2
onyx:0>
```

***file*/*filename* *mode* chmod –:**

Input(s):

file: A file object.

filename: A string that represents a filename.

mode: An integer that represents a Unix file mode.

Output(s): None.

Errors(s):

invalidfileaccess.

ioerror.

rangecheck.

stackunderflow.

typecheck.

unregistered.

Description:

Example(s):

```
onyx:0> '/tmp/tdir' 8@755 mkdir
onyx:0> '/tmp/tdir' status $mode get 1 sprint
16877
onyx:0> '/tmp/tdir' 'r' open
onyx:1> dup 8@555 chmod
onyx:1> '/tmp/tdir' status $mode get 1 sprint
16749
onyx:1>
```

file/filename uid gid **chown –:**

Input(s):

file: A file object.

filename: A string that represents a filename.

uid: An integer that represents a user ID.

gid: An integer that represents a group ID.

Output(s): None.

Errors(s):

invalidfileaccess.

ioerror.

rangecheck.

stackunderflow.

typecheck.

unregistered.

Description: Change the owner and group of a file.

Example(s):

```
onyx:0> '/tmp/tdir' 8@755 mkdir
onyx:0> '/tmp/tdir' status
onyx:1> dup $uid get 1 sprint
1001
onyx:1> $gid get 1 sprint
0
onyx:0> '/tmp/tdir' 1001 1001 chown
onyx:0> '/tmp/tdir' status
onyx:1> dup $uid get 1 sprint
1001
onyx:1> $gid get 1 sprint
1001
onyx:0>
```

path **chroot –:**

Input(s):

path: A string that represents a filesystem path.

Output(s): None.

Errors(s):

invalidaccess.

ioerror.

stackunderflow.

typecheck.

Description: Change the root directory to *path*. This operator requires super-user priviledges.

Example(s):

```
onyx:0> pwd 1 sprint
'/home/jasone/cw/devroot'
onyx:0> '/home/jasone' chroot
onyx:0> pwd 1 sprint
'/cw/devroot'
onyx:0>
```

***objects* clear –:**

Input(s):

objects: All objects on ostack.

Output(s): None.

Errors(s): None.

Description: Pop all objects off of ostack.

Example(s):

```
onyx:0> 1 2 3 pstack
3
2
1
onyx:3> clear pstack
onyx:0>
```

***mark* . . . cleartomark –:**

Input(s):

. . . : Zero or more objects.

mark: A mark object.

Output(s): None.

Errors(s):

unmatchedmark.

Description: Remove objects from ostack down to and including the topmost mark.

Example(s):

```
onyx:0> 3 mark 1 0 pstack
0
1
-mark-
3
onyx:4> cleartomark pstack
3
onyx:1>
```

file **close –:**

Input(s):

file: A file object.

Output(s): None.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Close a file.

Example(s):

```
onyx:0> '/tmp/foo' 'w' open
onyx:1> close
onyx:0>
```

– condition *condition*:

Input(s): None.

Output(s):

condition: A condition object.

Errors(s): None.

Description: Create a condition object.

Example(s):

```
onyx:0> condition 1 sprint
-condition-
onyx:0>
```

sock addr port **connect –:**

sock path **connect –:**

Input(s):

sock: A socket.

addr: An IPv4 address or DNS hostname.

port: An IPv4 port number. If not specified, the OS chooses a port number.

path: A filesystem path for a Unix-domain socket.

Output(s): None.

Errors(s):

argcheck.

invalidfileaccess.

neterror.

stackunderflow.

typecheck.

unregistered.

Description: Connect *sock*.

Example(s):

```
onyx:0> $AF_INET $SOCK_STREAM socket
onyx:1> dup 'localhost' 7777 connect
onyx:1>
```

***srcarray dstarray* copy *dstsubarray*:**

***srcdict dstdict* copy *dstdict*:**

***srcstack dststack* copy *dststack*:**

***srcstring dststring* copy *dstsubstring*:**

Input(s):

srcarray: An array object.
srcdict: A dict object.
srcstack: A stack object.
srcstring: A string object.
dstarray: An array object, at least as long as *srcarray*.
dstdict: A dict object.
dststack: A stack object.
dststring: A string object, at least as long as *srcstring*.

Output(s):

dstsubarray: A subarray of *dstarray*, with the same contents as *srcarray*.
dstdict: The same object as the input *dstdict*, but with the contents of *srcdict* inserted.
dststack: The same object as the input *dststack*, but with the contents of *srcstack* pushed.
dstsubstring: A substring of *dststring*, with the same contents as *srcstring*.

Errors(s):

rangecheck.
stackunderflow.
typecheck.

Description: Copy from one object to another. Array and string copying are destructive; dictionary and stack copying are not.

Example(s):

```
onyx:0> ['a'] ['b' 'c'] copy 1 sprint
['a']
onyx:0> <$foo 'foo'> <$bar 'bar'> copy 1 sprint
<$bar 'bar' $foo 'foo'>
onyx:1> (1 2) (3 4) copy 1 sprint
(3 4 1 2)
onyx:1> 'a' 'bc' copy 1 sprint
'a'
onyx:1>
```

***a* cos *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Cosine of a in radians.

Errors(s):

stackunderflow.
typecheck.

Description: Return the cosine of a in radians.

Example(s):
```
onyx:0> 0 cos 1 sprint
1.000000e+00
onyx:0> 3.14 cos 1 sprint
-9.999987e-01
onyx:0> 3.1415927 cos 1 sprint
-1.000000e+00
onyx:0>
```

***a* cosh *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Hyperbolic cosine of a in radians.

Errors(s):

stackunderflow.

typecheck.

Description: Return the hyperbolic cosine of a in radians.

Example(s):
```
onyx:0> 3 cosh 1 sprint
1.006766e+01
onyx:0>
```

– count *count*:

Input(s): None.

Output(s):

count: The number of objects on ostack.

Errors(s): None.

Description: Get the number of objects on ostack.

Example(s):
```
onyx:0> 2 1 0 count pstack
3
0
1
2
onyx:4>
```

– countdstack *count*:

Input(s): None.

Output(s):

count: Number of dictionaries on dstack.

Errors(s): None.

Description: Get the number of dictionaries on dstack.

Example(s):
```
onyx:0> countdstack 1 sprint
4
onyx:0> dict begin
```

```
onyx:0> countdstack 1 sprint
5
onyx:0>
```

– countestack *count*:

Input(s): None.

Output(s):

count: The number of objects currently on the execution stack (recursion depth).

Errors(s): None.

Description: Get the current number of objects on the execution stack.

Example(s):

```
onyx:0> countestack 1 sprint
3
onyx:0> estack 1 sprint
(--start-- -file- --estack--)
onyx:0>
```

***mark . . .* counttomark *mark . . . count*:**

Input(s):

. . . : Zero or more objects.

mark: A mark object.

Output(s):

. . . : *count* objects.

mark: The same mark that was passed in.

count: The depth of *mark* on ostack.

Errors(s):

unmatchedmark.

Description: Get the depth of the topmost mark on ostack.

Example(s):

```
onyx:0> 4 mark 2 1 0 counttomark 1 sprint
3
onyx:0>5
```

– currentdict *dict*:

Input(s): None.

Output(s):

dict: Topmost stack on dstack.

Errors(s): None.

Description: Get the topmost dictionary on dstack.

Example(s):

```
onyx:0> <$foo ‘foo’> begin
onyx:0> currentdict 1 sprint
<$foo ‘foo’>
onyx:0>
```

– currentlocking *boolean*:

Input(s): None.

Output(s):

boolean: If false, new objects are created with implicit locking disabled. Otherwise, new objects are created with implicit locking enabled.

Errors(s): None.

Description: Get the current implicit locking mode. See Section 2.7.1 for implicit synchronization details.

Example(s):

```
onyx:0> currentlocking 1 sprint
false
onyx:0> true setlocking
onyx:0> currentlocking 1 sprint
true
onyx:0>
```

***real precision* cvds *string*:**

Input(s):

real: A real.

precision: Number of digits after the decimal point to show. If negative, do not show trailing zeros.

Output(s):

string: A string representation of *real* in decimal form with *precision* digits of decimal precision.

Errors(s):

stackunderflow.

typecheck.

Description: Convert *real* to a string representation in decimal notation, with *precision* digits of decimal precision.

Example(s):

```
onyx:0> 42.3 0 cvds 1 sprint
'42'
onyx:0> 42.3 1 cvds 1 sprint
'42.3'
onyx:0> -42.3 4 cvds 1 sprint
'-42.3000'
onyx:0> -43.3 -4 cvds 1 sprint
'-42.3'
onyx:0>
```

***obj* cve *obj*:**

Input(s):

obj: An object.

Output(s):

obj: The same object that was passed in, but with the evaluatable attribute set.

Errors(s):

stackunderflow.

Description: Set the evaluatable attribute for *obj*.

Example(s):

```
onyx:0> [1 2 3] cve 1 sprint
_{1 2 3}_
onyx:0>
```

***real precision* cves *string*:**

Input(s):

real: A real.

precision: Number of digits after the decimal point to show.

Output(s):

string: A string representation of *real* in exponential form with *precision* digits of decimal precision.

Errors(s):

stackunderflow.

typecheck.

Description: Convert *real* to a string representation in exponential notation, with *precision* digits of decimal precision.

Example(s):

```
onyx:0> 42.3 0 cves 1 sprint
'4e+01'
onyx:0> 42.3 1 cves 1 sprint
'4.2e+01'
onyx:0> 42.3 2 cves 1 sprint
'4.23e+01'
onyx:0> -42.3 5 cves 1 sprint
'-4.23000e+01'
onyx:0>
```

***obj* cvlit *obj*:**

Input(s):

obj: An object.

Output(s):

obj: The same object that was passed in, but with the literal attribute set.

Errors(s):

stackunderflow.

Description: Set the literal attribute for *obj*.

Example(s):

```
onyx:0> {1 2 3} cvlit 1 sprint
[1 2 3]
onyx:0>
```

***string* cvn *name*:**

Input(s):

string: A string.

Output(s):

name: A literal name that corresponds to *string*.

Errors(s):

stackunderflow.

typecheck.

Description: Convert *string* to a literal name.

Example(s):

```
onyx:0> 'foo' cvn 1 sprint
$foo
onyx:0>
```

***integer radix* cvrs *string*:**

Input(s):

integer: An integer.

radix: A numerical base, from 2 to 36, inclusive.

Output(s):

string: A string representation of *integer* in base *radix*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Convert *integer* to a string representation in base *radix*.

Example(s):

```
onyx:0> 42 2 cvrs 1 sprint
'101010'
onyx:0> 42 16 cvrs 1 sprint
'2a'
onyx:0>
```

***obj* cvs *string*:**

Input(s):

obj: An object.

Output(s):

string: A string representation of *obj*. The string depends on the type of *obj*:

boolean: `'true'` or `'false'`.

name: The string representation of the name.

integer: The integer in base 10.

operator: The string representation of the operator name or `'-operator-'`.

real: The real in exponential notation.

string: A printable representation of *obj*. The result can be evaluated to produce the original string.

Other types: `'--nostringval--'`.

Errors(s):

stackunderflow.

Description: Convert *obj* to a string representation.

Example(s):

```
onyx:0> true cvs 1 sprint
'true'
onyx:0> $foo cvs 1 sprint
```

```
'foo'
onyx:0> 42 cvs 1 sprint
'42'
onyx:0> ~pop cvs 1 sprint
'pop'
onyx:0> 42.0 cvs 1 sprint
'4.200000e+01'
onyx:0> 'foo\nbar\\biz\'baz' cvs 1 sprint
'\'foo\\nbar\\\\biz\\\'baz\''
onyx:0> mutex cvs 1 sprint
'--nostringval--'
onyx:0>
```

***obj* cvx *obj*:**

Input(s):

obj: An object.

Output(s):

obj: The same object that was passed in, but with the executable attribute set.

Errors(s):

stackunderflow.

Description: Set the executable attribute for *obj*.

Example(s):

```
onyx:0> [1 2 3] cvx 1 sprint
{1 2 3}
onyx:0>
```

***a* dec *r*:**

Input(s):

a: An integer.

Output(s):

r: $a - 1$.

Errors(s):

stackunderflow.

typecheck.

Description: Subtract one from a.

Example(s):

```
onyx:0> 1 dec 1 sprint
0
onyx:0>
```

***key val* def –:**

Input(s):

key: An object.

val: A value associated with *key*.

Output(s): None.

Errors(s):

stackunderflow.

Description: Define *key* with associated value *val* in the topmost dictionary on dstack. If *key* is already defined in that dictionary, the old definition is replaced.

Example(s):

```
onyx:0> $foo ‘foo’ def
onyx:0> foo 1 sprint
‘foo’
onyx:0> $foo ‘FOO’ def
onyx:0> foo 1 sprint
‘FOO’
onyx:0>
```

***thread* detach –:**

Input(s):

thread: A thread object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Detach *thread* so that its resources will be automatically reclaimed after it exits. A thread may only be detached or joined once; any attempt to do so more than once results in undefined behavior (likely crash).

Example(s):

```
onyx:0> (1 2) {add 1 sprint self detach} thread
3
onyx:1>
```

– dict *dict*:

Input(s): None.

Output(s):

dict: An empty dictionary.

Errors(s): None.

Description: Create an empty dictionary.

Example(s):

```
onyx:0> dict 1 sprint
<>
onyx:0>
```

***status* die –:**

Input(s):

status: A integer from 0 to 255 that is used as the program exit code.

Output(s): None.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Exit the program with exit code *status*.

Example(s):

```
onyx:0> 1 die
```

***path proc* dirforeach –:**

Input(s):

path: A string that represents a filesystem path.

proc: An object to be executed.

Output(s): None.

Errors(s):

invalidaccess.

ioerror.

stackunderflow.

typecheck.

Description: For each entry in the directory represented by *path* except for "." and "..", push a string that represents the entry onto ostack and execute *proc*. This operator supports the **exit** operator.

Example(s):

```
onyx:0> pwd {1 sprint} dirforeach
'CVS'
'.cvsignore'
'Cookfile'
'Cookfile.inc'
'latex'
'Cookfile.inc.in'
onyx:0> pwd {'Cookfile.inc' search
     {pop 'Yes: ' print 1 sprint pop exit}
     {'Not: ' print 1 sprint} ifelse
} dirforeach
Not: 'CVS'
Not: '.cvsignore'
Not: 'Cookfile'
Yes: 'Cookfile.inc'
onyx:0>
```

***a b* div *r*:**

Input(s):

a: An integer or real.

b: A non-zero integer or real.

Output(s):

r: The quotient of a divided by b.

Errors(s):

stackunderflow.

typecheck.

undefinedresult.

Description: Return the quotient of a divided by b.

Example(s):

```
onyx:0> 4 2 div 1 sprint
2.000000e+00
onyx:0> 5 2.0 div 1 sprint
2.500000e+00
onyx:0> 5.0 0 div
Error $undefinedresult
ostack: (5.000000e+00 0)
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      --div--
1:      -file-
2:      --start--
onyx:3>
```

***a b c* dn *b c a*:**

Input(s):

a: An object.

b: An object.

c: An object.

Output(s):

b: An object.

c: An object.

a: An object.

Errors(s):

stackunderflow.

Description: Rotate the top three objects on ostack down one position.

Example(s):

```
onyx:0> ‘a’ ‘b’ ‘c’ ‘d’ dn pstack
‘b’
‘d’
‘c’
‘a’
onyx:4>
```

– dstack *stack*:

Input(s): None.

Output(s):

stack: A snapshot of dstack.

Errors(s): None.

Description: Get a snapshot of dstack.

Example(s):

```
onyx:0> dstack 1 sprint
(-dict- -dict- -dict- -dict-)
onyx:0>
```

***obj* dup *obj dup*:**

Input(s):

obj: An object.

Output(s):

obj: The same object that was passed in.

dup: A duplicate of *obj*.

Errors(s):

stackunderflow.

Description: Create a duplicate of the top object on ostack. For composite objects, the new object is a reference to the same composite object.

Example(s):

```
onyx:0> 1 dup pstack
1
1
onyx:2>
```

***obj* echeck *boolean*:**

Input(s):

obj: An object.

Output(s):

boolean: True if *obj* has the evaluatable attribute, false otherwise.

Errors(s):

stackunderflow.

Description: Check *obj* for evaluatable attribute.

Example(s):

```
onyx:0> {1 2 3} cve
onyx:1> dup 1 sprint
_{1 2 3}_
onyx:1> echeck 1 sprint
true
onyx:0> {1 2 3} echeck 1 sprint
false
onyx:0> [1 2 3] echeck 1 sprint
false
onyx:0>
```

– egid *gid*:

Input(s): None.

Output(s):

gid: Process's effective group ID.

Errors(s): None.

Description: Get the process's effective group ID.

Example(s):

```
onyx:0> egid 1 sprint
1001
onyx:0>
```

– end –:

Input(s): None.

Output(s): None.

Errors(s):

stackunderflow.

Description: Pop the topmost dictionary off dstack, thereby removing its contents from the namespace.

Example(s):

```
onyx:0> <$foo 'foo'> begin
onyx:0> foo 1 sprint
'foo'
onyx:0> end
onyx:0> foo 1 sprint
Error $undefined
ostack: ()
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      foo
1:      -file-
2:      --start--
onyx:1>
```

– envdict *dict*:

Input(s): None.

Output(s):

dict: A dictionary.

Errors(s): None.

Description: Get envdict. See Section 2.10.2 for details on envdict.

Example(s):

```
onyx:0> envdict 0 sprint
-dict-
onyx:0>
```

***a b* eq *boolean*:**

Input(s):

a: An object.

b: An object.

Output(s):

boolean: True if a is equal to b, false otherwise.

Errors(s):

stackunderflow.

Description: Compare two objects for equality. Equality has the following meaning, depending on the types of a and b:

array, condition, dict, file, hook, mutex, stack, thread: a and b are equal iff they refer to the same memory.

operator: a and b are equal iff they refer to the same function.

name, string: a and b are equal iff they are lexically equivalent. A name can be equal to a string.

boolean: *a* and *b* are equal iff they are the same value.

integer, real: *a* and *b* are equal iff they are the same value.

Example(s):

```
onyx:0> mutex mutex eq 1 sprint
false
onyx:0> mutex dup eq 1 sprint
true
onyx:0> $foo ‘foo’ eq 1 sprint
true
onyx:0> true true eq 1 sprint
true
onyx:0> true false eq 1 sprint
false
onyx:0> 1 1 eq 1 sprint
true
onyx:0> 1 2 eq 1 sprint
false
onyx:0> 1.0 1 eq 1 sprint
true
onyx:0> 1.0 1.1 eq 1 sprint
false
onyx:0>
```

– estack *stack*:

Input(s): None.

Output(s):

stack: A current snapshot (copy) of the execution stack.

Errors(s): None.

Description: Get a current snapshot of the execution stack.

Example(s):

```
onyx:0> estack 1 sprint
(--start-- -file- --estack--)
onyx:0>
```

– euid *uid*:

Input(s): None.

Output(s):

uid: Process's effective user ID.

Errors(s): None.

Description: Get the process's effective user ID.

Example(s):

```
onyx:0> euid 1 sprint
1001
onyx:0>
```

***obj* eval –:**

Input(s):

obj: An object.

Output(s): None.

Errors(s):

stackunderflow.

Description: Evaluate object. See Section 2.1 for details on object evaluation.

Example(s):

```
onyx:0> ‘‘hi’ 1 sprint’ cvx eval
‘hi’
onyx:0>
```

***a b* exch *b a*:**

Input(s):

a: An object.

b: An object.

Output(s):

b: The same object that was passed in.

a: The same object that was passed in.

Errors(s):

stackunderflow.

Description: Exchange the top two objects on ostack.

Example(s):

```
onyx:0> 1 2 pstack
2
1
onyx:2> exch pstack
1
2
onyx:2>
```

***args* exec –:**

Input(s):

args: An array of strings. The first string in *args* is the path of the program to invoke, and any additional array elements are passed as command line arguments to the invoked program.

Output(s): None (this operator does not return).

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Overlay a new program and execute it. The current contents of envdict are used to construct the new program’s environment.

Example(s):

```
onyx:0> ‘Old program’
onyx:1> [‘/usr/local/bin/onyx’] exec
Canonware Onyx, version 1.0.0.
onyx:0>
```

– exit –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Exit the innermost enclosing looping context immediately. This operator can be called within the looping context of **for**, **repeat**, **while**, **until**, **loop**, **foreach**, and **dirforeach**.

Example(s):

```
onyx:0> {‘hi’ 1 sprint exit ‘bye’ 1 sprint} loop
‘hi’
onyx:0>
```

***b* exp *r*:**

Input(s):

a: An integer or real.

Output(s):

r: e raised to the b power.

Errors(s):

stackunderflow.

typecheck.

Description: Return e (the base of natural logarithm) raised to the b power.

Example(s):

```
onyx:0> 3 exp 1 sprint
2.008554e+01
onyx:0>
```

– false *false*:

Input(s): None.

Output(s):

false: The boolean value false.

Errors(s): None.

Description: Return false.

Example(s):

```
onyx:0> false 1 sprint
false
onyx:0>
```

***a* floor *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Integer floor of a.

Errors(s):

stackunderflow.

typecheck.

Description: Return the integer floor of a.

Example(s):

```
onyx:0> -1.51 floor 1 sprint
-2
onyx:0> -1.49 floor 1 sprint
-2
onyx:0> 0 floor 1 sprint
0
onyx:0> 1.49 floor 1 sprint
1
onyx:0> 1.51 floor 1 sprint
1
onyx:0>
```

– flush –:

Input(s): None.

Output(s): None.

Errors(s):

ioerror.

Description: Flush any buffered data associated with stdout.

Example(s):

```
onyx:0> 'Hi\n' print
onyx:0> flush
Hi
onyx:0>
```

file **flushfile –:**

Input(s):

file: A file object.

Output(s): None.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Flush any buffered data associated with *file*.

Example(s):

```
onyx:0> 'Hi\n' print
onyx:0> stdout flushfile
Hi
onyx:0>
```

init inc limit proc **for –:**

Input(s):

init: Initial value of control variable.

inc: Amount to increment control variable by at the end of each iteration.

limit: Inclusive upper bound for control variable if less than or equal to *init*, otherwise inclusive lower bound for control variable.

proc: An object.

Output(s): At the beginning of each iteration, the current value of the control variable is pushed onto ostack.

Errors(s):

stackunderflow.

typecheck.

Description: Iteratively evaluate *proc*, pushing a control variable onto ostack at the beginning of each iteration, until the control variable has exceeded *limit*. This operator supports the **exit** operator.

Example(s):

```
onyx:0> 0 1 3 {1 sprint} for
0
1
2
3
onyx:0> 0 -1 -3 {1 sprint} for
0
-1
-2
-3
onyx:0> 0 2 7 {1 sprint} for
0
2
4
6
onyx:0> 0 1 1000 {dup 1 sprint 3 eq {exit} if} for
0
1
2
3
onyx:0>
```

array proc **foreach –:**

dict proc **foreach –:**

stack proc **foreach –:**

string proc **foreach –:**

Input(s):

array: An array object.

dict: A dict object.

stack: A stack object.

string: A string object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: For each entry in the first input argument (*array*, *dict*, *stack*, or *string*), push the entry onto ostack and execute *proc*. This operator supports the **exit** operator.

The object being iterated over can be modified during iteration, with the expectation of no ill consequences, and in most cases the modifications are immediately apparent. However, there are some cases in which behavior does not follow this guideline:

- Objects inserted into a dictionary during iteration may or may not be iterated over.
- In the case of stack iteration, a snapshot is taken before iteration begins, so any changes to the stack during iteration will not affect iteration in any way.

Example(s):

```
onyx:0> [1 2] {1 sprint} foreach
1
2
onyx:0> <$foo ‘foo’ $bar ‘bar’> {pstack clear} foreach
‘bar’
$bar
‘foo’
$foo
onyx:0> (1 2) {pstack clear} foreach
2
1
onyx:0> ‘ab’ {pstack clear} foreach
97
98
onyx:0>
```

***args* forkexec *pid*:**

Input(s):

args: An array of strings. The first string in *args* is the path of the program to invoke, and any additional array elements are passed as command line arguments to the invoked program.

Output(s):

pid: Process identifier for the new process, or 0 if the child process.

Errors(s):

limitcheck.

rangecheck.

stackunderflow.

typecheck.

Description: Fork and exec a new process. The current contents of envdict are used to construct the new program’s environment.

Example(s):

```
onyx:0> [‘/bin/date’] forkexec dup 1 sprint waitpid 1 sprint
6516
Sat Jul 13 20:47:54 PDT 2002
0
onyx:0>
```

– gcdict *dict*:

Input(s): None.

Output(s):

dict: A dictionary.

Errors(s): None.

Description: Get gcdict. See Section 2.10.4 for details on gcdict.

Example(s):

```
onyx:0> gcdict 0 sprint
-dict-
onyx:0>
```

***a b* ge *boolean*:**

Input(s):

a: A number (integer or real) or string.

b: An object of a type compatible with a.

Output(s):

boolean: True if a is greater than or equal to b, false otherwise.

Errors(s):

stackunderflow.

typecheck.

Description: Compare two numbers or strings.

Example(s):

```
onyx:0> 1 2 ge 1 sprint
false
onyx:0> 1 1 ge 1 sprint
true
onyx:0> 2 1 ge 1 sprint
true
onyx:0> 1 1.1 ge 1 sprint
false
onyx:0> 1.1 1.1 ge 1 sprint
true
onyx:0> 1.1 1 ge 1 sprint
true
onyx:0> ‘a’ ‘b’ ge 1 sprint
false
onyx:0> ‘a’ ‘a’ ge 1 sprint
true
onyx:0> ‘b’ ‘a’ ge 1 sprint
true
onyx:0>
```

***array index* get *obj*:**

***dict key* get *value*:**

***string index* get *integer*:**

Input(s):

array: An array object.

dict: A dict object.

string: A string object.

index: Offset of *array* element or *string* element.

key: A key in *dict*.

Output(s):

obj: The object in *array* at offset *index*.

value: The value in *dict* corresponding to *key*.

integer: The ascii value of the character in *string* at offset *index*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

undefined.

Description: Get an element of *array*, a value in *dict*, or an element of *string*.

Example(s):

```
onyx:0> ['a' 'b' 'c'] 1 get 1 sprint
'b'
onyx:0> <$foo 'foo' $bar 'bar'> $bar get 1 sprint
'bar'
onyx:0> 'abc' 1 get 1 sprint
98
onyx:0>
```

***array index length* getinterval *subarray*:**

***string index length* getinterval *substring*:**

Input(s):

array: An array object.

string: A string object.

index: The offset into *array* or *string* to get the interval from.

length: The length of the interval in *array* or *string* to get.

Output(s):

subarray: A subarray of *array* at offset *index* and of length *length*.

substring: A substring of *string* at offset *index* and of length *length*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Get an interval of *array* or *string*.

Example(s):

```
onyx:0> [0 1 2 3] 1 2 getinterval 1 sprint
[1 2]
onyx:0> 'abcd' 1 2 getinterval 1 sprint
'bc'
onyx:0>
```

– gid *gid*:

Input(s): None.

Output(s):

gid: Process's group ID.

Errors(s): None.

Description: Get the process's group ID.

Example(s):
```
onyx:0> gid 1 sprint
1001
onyx:0>
```

– globaldict *dict*:

Input(s): None.

Output(s):

dict: A dictionary.

Errors(s): None.

Description: Get globaldict. See Section 2.10.5 for details on globaldict.

Example(s):
```
onyx:0> globaldict 1 sprint
<>
onyx:0>
```

– gstderr *file*:

Input(s): None.

Output(s):

file: A file object corresponding to the global stderr.

Errors(s): None.

Description: Get the global stderr that is inherited by new threads. See Section 2.4 for standard I/O details.

Example(s):
```
onyx:0> gstderr pstack
-file-
onyx:1>
```

– gstdin *file*:

Input(s): None.

Output(s):

file: A file object corresponding to the global stdin.

Errors(s): None.

Description: Get the global stdin that is inherited by new threads. See Section 2.4 for standard I/O details.

Example(s):
```
onyx:0> gstdin pstack
-file-
onyx:1>
```

– gstdout *file*:

Input(s): None.

Output(s):

file: A file object corresponding to the global stdout.

Errors(s): None.

Description: Get the global stdout that is inherited by new threads. See Section 2.4 for standard I/O details.

Example(s):

```
onyx:0> gstdout pstack
-file-
onyx:1>
```

***a b* gt *boolean*:**

Input(s):

a: A number (integer or real) or string.

b: An object of a type compatible with a.

Output(s):

boolean: True if a is greater than b, false otherwise.

Errors(s):

stackunderflow.

typecheck.

Description: Compare two numbers or strings.

Example(s):

```
onyx:0> 1 1 gt 1 sprint
false
onyx:0> 2 1 gt 1 sprint
true
onyx:0> 1.1 1.1 gt 1 sprint
false
onyx:0> 1.1 1 gt 1 sprint
true
onyx:0> ‘a’ ‘a’ gt 1 sprint
false
onyx:0> ‘b’ ‘a’ gt 1 sprint
true
onyx:0>
```

***hook* hooktag *tag*:**

Input(s):

hook: A hook object.

Output(s):

tag: The tag associated with *hook*.

Errors(s):

stackunderflow.

typecheck.

Description: Get the tag associated with *hook*.

Example(s):

. . . *obj* „,*index* ibdup . . . *obj* „,*dup*:

Input(s):

. . . : *index* objects.

obj: An object.

,,,: Zero or more objects.

index: Offset from bottom of ostack, counting from 0.

Output(s):

. . . : *index* objects.

obj: An object.

,,,: Zero or more objects.

dup: Duplicate of *obj*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Create a duplicate of the object on ostack that is at offset *index* from the bottom of ostack.

Example(s):

```
onyx:4> 2 ibdup pstack
2
3
2
1
0
onyx:5>
```

***. . . obj ,,,index* ibpop *. . . ,,,*:**

Input(s):

. . . : *index* objects.

obj: An object.

,,,: Zero or more objects.

index: Offset from bottom of ostack, counting from 0.

Output(s):

. . . : *index* objects.

,,,: Zero or more objects.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Remove the object from ostack that is at offset *index* from the bottom of ostack.

Example(s):

```
onyx:0> 0 1 2 3
onyx:4> 2 ibpop pstack
3
1
0
onyx:3>
```

***a b* idiv *r*:**

Input(s):

a: An integer.

b: A non-zero integer.

Output(s):

r: The integer quotient of a divided by b.

Errors(s):

stackunderflow.

typecheck.

undefinedresult.

Description: Return the integer quotient of a divided by b.

Example(s):

```
onyx:0> 4 2 idiv 1 sprint
2
onyx:0> 5 2 idiv 1 sprint
2
onyx:0> 5 0 idiv
Error $undefinedresult
ostack: (5 0)
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      --idiv--
1:      -file-
2:      --start--
onyx:3>
```

***obj . . . index* idup *obj . . . dup*:**

Input(s):

obj: An object.

index: Offset from top of ostack, counting from 0, not counting *index*), of the object to duplicate on ostack.

Output(s):

obj: The same object that was passed in.

dup: A duplicate of *obj*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Create a duplicate of the object on ostack at *index*.

Example(s):

```
onyx:0> 3 2 1 0 2 idup pstack
2
0
1
2
3
onyx:5>
```

***boolean obj* if –:**

Input(s):

boolean: A boolean.

obj: An object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Evaluate *obj* if *boolean* is true.

Example(s):

```
onyx:0> true {‘yes’ 1 sprint} if
‘yes’
onyx:0> false {‘yes’ 1 sprint} if
onyx:0>
```

***boolean a b* ifelse –:**

Input(s):

boolean: A boolean.

a: An object.

b: An object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Evaluate *a* if *boolean* is true, evaluate *b* otherwise. See Section 2.1 for details on object evaluation.

Example(s):

```
onyx:0> true {‘yes’}{‘no’} ifelse 1 sprint
‘yes’
onyx:0> false {‘yes’}{‘no’} ifelse 1 sprint
‘no’
onyx:0>
```

***a* inc *r*:**

Input(s):

a: An integer.

Output(s):

r: $a + 1$.

Errors(s):

stackunderflow.

typecheck.

Description: Add one to a.

Example(s):

```
onyx:0> 1 inc 1 sprint
2
onyx:0>
```

file **iobuf** ***count*****:**

Input(s):

file: A file object.

Output(s):

count: The size in bytes of the I/O buffer associated with *file*.

Errors(s):

stackunderflow.

typecheck.

Description: Get the size of the I/O buffer associated with *file*.

Example(s):

```
onyx:0> stdout iobuf 1 sprint
512
onyx:0> stderr iobuf 1 sprint
0
onyx:0>
```

obj ... index **ipop ...** **:**

Input(s):

obj: An object.

index: Offset from top of ostack, counting from 0, not counting *index*), of the object to remove from ostack.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Remove the *obj* at *index* from ostack.

Example(s):

```
onyx:0> 2 1 0
onyx:3> 1 ipop pstack
0
2
onyx:2>
```

– istack ***stack*****:**

Input(s): None.

Output(s):

stack: A current snapshot (copy) of the index stack.

Errors(s): None.

Description: Get a current snapshot of the index stack.

Example(s):

```
onyx:0> istack 1 sprint
(0 0 0)
onyx:0>
```

thread **join –:**

Input(s):

thread: A thread object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Wait for *thread* to exit. A thread may only be detached or joined once; any attempt to do so more than once results in undefined behavior (likely crash).

Example(s):

```
onyx:0> (1 2) {add 1 sprint} thread join 'Done\n' print flush
3
Done
onyx:0>
```

***dict key* known *boolean*:**

Input(s):

dict: A dictionary.

key: A key to look for in *dict*.

Output(s):

boolean: True if *key* is defined in *dict*, false otherwise.

Errors(s):

stackunderflow.

typecheck.

Description: Check whether *key* is defined in *dict*.

Example(s):

```
onyx:1> <$foo 'foo'> $foo known 1 sprint
true
onyx:1> <$foo 'foo'> $bar known 1 sprint
false
onyx:1>
```

***obj* lcheck *boolean*:**

Input(s):

obj: An array, dict, file, or string.

Output(s):

boolean: True if *obj* is implicitly locked, false otherwise.

Errors(s):

stackunderflow.

typecheck.

Description: Check if *obj* is implicitly locked.

Example(s):

```
onyx:0> false setlocking
onyx:0> [1 2 3] lcheck 1 sprint
false
onyx:0> true setlocking
onyx:0> [1 2 3] lcheck 1 sprint
true
onyx:0>
```

***a b* le *boolean*:**

Input(s):

a: A number (integer or real) or string.

b: An object of a type compatible with a.

Output(s):

boolean: True if a is less than or equal to b, false otherwise.

Errors(s):

stackunderflow.

typecheck.

Description: Compare two numbers or strings.

Example(s):

```
onyx:0> 1 2 le 1 sprint
true
onyx:0> 1 1 le 1 sprint
true
onyx:0> 2 1 le 1 sprint
false
onyx:0> 1 1.1 le 1 sprint
true
onyx:0> 1.1 1.1 le 1 sprint
true
onyx:0> 1.1 1 le 1 sprint
false
onyx:0> ‘a’ ‘b’ le 1 sprint
true
onyx:0> ‘a’ ‘a’ le 1 sprint
true
onyx:0> ‘b’ ‘a’ le 1 sprint
false
onyx:0>
```

***array* length *count*:**

***dict* length *count*:**

***name* length *count*:**

***string* length *count*:**

Input(s):

array: An array object.

dict: A dict object.

name: A name object.

string: A string object.

Output(s):

count: Number of elements in *array*, number of entries in *dict*, number of characters in *name*, or number of characters in *string*.

Errors(s):

stackunderflow.

typecheck.

Description: Get the umber of elements in *array*, number of entries in *dict*, number of characters in *name*, or number of characters in *string*.

Example(s):

```
onyx:0> [1 2 3] length 1 sprint
3
onyx:0> <$foo 'foo' $bar 'bar'> length 1 sprint
2
onyx:0> $foo length 1 sprint
3
onyx:0> 'foo' length 1 sprint
3
onyx:0>
```

***filename linkname* link –:**

Input(s):

filename: A string that represents a filename.

linkname: A string that represents a filename.

Output(s): None.

Errors(s):

invalidfileaccess.

ioerror.

stackunderflow.

typecheck.

undefinedfilename.

unregistered.

Description: Create a hard link from *linkname* to *filename*.

Example(s):

```
onyx:0> '/tmp/foo' 'w' open
onyx:1> dup 'Hello\n' write
onyx:1> dup flushfile
onyx:1> close
onyx:0> '/tmp/foo' '/tmp/bar' link
onyx:0> '/tmp/bar' 'r' open
onyx:1> readline
onyx:2> pstack
false
'Hello'
onyx:2>
```

***sock backlog* listen –:**

Input(s):

sock: A socket.

backlog: Maximum backlog of connections to listen for. If not specified, the maximum backlog is used.

Output(s): None.

Errors(s):

invalidfileaccess.

neterror.

stackunderflow.

typecheck.

unregistered.

Description: Listen for connections on a socket.

Example(s):

```
onyx:0> $AF_INET $SOCK_STREAM socket
onyx:1> dup ‘localhost’ 7777 bindsocket
onyx:1> dup listen
onyx:1>
```

a **ln** ***r*:**

Input(s):

a: An integer or real.

Output(s):

r: Natural logarithm of a.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Return the natural logarithm of a.

Example(s):

```
onyx:0> 5 ln 1 sprint
1.609438e+00
onyx:0> 8.5 ln 1 sprint
2.140066e+00
onyx:0>
```

key **load** ***val*:**

Input(s):

key: A key to look up in dstack.

Output(s):

val: The value associated with the topmost definition of *key* in dstack.

Errors(s):

stackunderflow.

undefined.

Description: Get the topmost definition of *key* in dstack.

Example(s):

```
onyx:1> <$foo ‘foo’> begin
onyx:1> <$foo ‘FOO’> begin
onyx:1> $foo load 1 sprint
‘FOO’
onyx:1>
```

nsecs **localtime** ***dict*:**

Input(s):

nsecs: Number of nanoseconds since the epoch.

Output(s):

dict: A dictionary that contains the following entries:

sec: Seconds (0-59).
min: Minutes (0-59).
hour: Hours (0-23).
mday: Month day (1-31).
mon: Month (0-11).
year: Year.
wday: Week day (0-6, Sunday is 0).
yday: Year day (0-365).
isdst: Is daylight savings time (true or false).
zone: Time zone (string).
gmtoff: Offset from UTC in seconds.

Errors(s):

stackunderflow.
rangecheck.
typecheck.

Description: Convert a time, as returned by realtime, to a dictionary that contains time information in a more human-usable format.

Example(s):

```
onyx:0> $date {
    realtime localtime

    [‘Sunday’ ‘Monday’ ‘Tuesday’ ‘Wednesday’ ‘Thursday’ ‘Friday’ ‘Saturday’]
    over $wday get
    get
    ‘ ’ cat

    over $year get cvs
    ‘/’ 3 ncat

    over $mon get inc <$w 2 $p ‘0’> outputs
    ‘/’ 3 ncat

    over $mday get <$w 2 $p ‘0’> outputs
    ‘ ’ 3 ncat

    over $hour get <$w 2 $p ‘0’> outputs
    ‘:’ 3 ncat

    over $min get <$w 2 $p ‘0’> outputs
    ‘:’ 3 ncat

    over $sec get <$w 2 $p ‘0’> outputs
```

```
        ' (' 3 ncat

        exch $zone get
        ')\n' 3 ncat

        print flush
    } def
    onyx:0> date
    Monday 2003/03/17 01:31:49 (PST)
    onyx:0>
```

***mutex* lock –:**

Input(s):

mutex: A mutex object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Acquire *mutex*, waiting if necessary. Attempting to acquire *mutex* recursively will result in undefined behavior (likely deadlock or crash).

Example(s):

```
onyx:0> mutex dup lock unlock
onyx:0>
```

***a* log *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Base 10 logarithm of a.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Return the base 10 logarithm of a.

Example(s):

```
onyx:0> 5 log 1 sprint
6.989700e-01
onyx:0> 8.5 log 1 sprint
9.294189e-01
onyx:0>
```

***proc* loop –:**

Input(s):

proc: An object to evaluate.

Output(s): None.

Errors(s):

stackunderflow.

Description: Repeatedly evaluate *proc* indefinitely. This operator supports the **exit** operator.

Example(s):

```
onyx:0> 0 {1 add dup 1 sprint dup 3 eq {pop exit} if} loop
1
2
3
onyx:0>
```

***a b* lt *boolean*:**

Input(s):

a: A number (integer or real) or string.

b: An object of a type compatible with a.

Output(s):

boolean: True if a is less than b, false otherwise.

Errors(s):

stackunderflow.

typecheck.

Description: Compare two numbers or strings.

Example(s):

```
onyx:0> 1 2 lt 1 sprint
true
onyx:0> 1 1 lt 1 sprint
false
onyx:0> 1 1.1 lt 1 sprint
true
onyx:0> 1.1 1.1 lt 1 sprint
false
onyx:0> 1.1 1 lt 1 sprint
false
onyx:0> ‘a’ ‘b’ lt 1 sprint
true
onyx:0> ‘a’ ‘a’ lt 1 sprint
false
onyx:0>
```

– mark *mark*:

Input(s): None.

Output(s):

mark: A mark object.

Errors(s): None.

Description: Push a mark onto ostack.

Example(s):

```
onyx:0> mark pstack
-mark-
onyx:1>
```

***input pattern flags* match *boolean*:**

***input pattern* match *boolean*:**

***input regex* match *boolean*:**

Input(s):

input: An input string to find matches in.

pattern: A string that specifies a regular expression. See Section 2.9 for syntax.

flags: A dictionary of optional flags:

$c: Continue where previous match ended. Don't update the offset to start the next match from unless this match is successful. Defaults to false.

$g: Continue where previous match ended. If the match is unsuccessful, update the offset to start the next match from to the beginning of *input*. Defaults to false.

$i: Case insensitive. Defaults to false.

$m: Treat input as a multi-line string. Defaults to false.

$s: Treat input as a single line, so that the dot metacharacter matches any character, including a newline. Defaults to false.

regex: A regex object.

Output(s):

boolean:

TRUE: Match successful.

FALSE: No match found.

Errors(s):

regexerror.

stackunderflow.

typecheck.

Description: Look in *input* for a match to the regular expression specified by *regex*/*pattern*/*flags*.

Example(s):

```
onyx:0> 'input' 'I' <$i true> match {0 submatch 1 sprint} if
'i'
onyx:0> 'input' 'I' <$i true> regex match {0 submatch 1 sprint} if
'i'
onyx:0> 'input' 'I' match {0 submatch 1 sprint} if
onyx:0>
```

***path* mkdir –:**

***path mode* mkdir –:**

Input(s):

path: A string object that represents a directory path.

mode: An integer that represents a Unix file mode.

Output(s): None.

Errors(s):

invalidfileaccess.

ioerror.

rangecheck.

stackunderflow.

typecheck.

unregistered.

Description: Create a directory.

Example(s):

```
onyx:0> '/tmp/tdir' 8@755 mkdir
onyx:0> '/tmp/tdir' {1 sprint} dirforeach
'.'
'..'
onyx:0>
```

path **mkfifo –:**

path mode **mkfifo –:**

Input(s):

path: A string object that represents a directory path.

mode: An integer that represents a Unix file mode.

Output(s): None.

Errors(s):

invalidfileaccess.

ioerror.

rangecheck.

stackunderflow.

typecheck.

unregistered.

Description: Create a named pipe.

Example(s):

```
onyx:0> '/tmp/fifo' mkfifo
onyx:0>
```

a b **mod** ***r*****:**

Input(s):

a: An integer or real.

b: A non-zero integer or real.

Output(s):

r: The modulus of a and b.

Errors(s):

stackunderflow.

typecheck.

undefinedresult.

Description: Return the modulus of a and b. Note that a and b can be any combination of integers and reals.

Example(s):

```
onyx:0> 4 2 mod 1 sprint
0
onyx:0> 5 2 mod 1 sprint
1
onyx:0> 5 0 mod
Error $undefinedresult
ostack: (5 0)
dstack: (-dict- -dict- -dict- -dict-)
```

```
estack/istack trace (0..2):
0:      --mod--
1:      -file-
2:      --start--
onyx:3>
```

***path symbol* modload –:**

Input(s):

path: A string that represents a module filename.

symbol: A string that represents the symbol name of a module initialization function to be executed.

Output(s): None.

Errors(s):

invalidfileaccess.

stackunderflow.

typecheck.

undefined.

Description: Dynamically load a module, create a hook object that encapsulates the handle returned by dlopen(3) (hook data pointer) and the module initialization function (hook evaluation function), and evaluate the hook.

All objects that refer to code and/or data that are part of the module must directly and/or indirectly maintain a reference to the hook that is evaluated by this operator, since failing to do so would allow the garbage collector to unload the module, which could result in dangling pointers to unmapped memory regions.

Loadable modules present a problem for the garbage collector during the sweep phase. All objects that refer to memory that is dynamically mapped as part of the module must be destroyed before the module is unloaded. Destruction ordering constraints show up in other situations as well, but in the case of loadable modules, there is no reasonable solution except to explicitly order the destruction of objects. Therefore, by default, the hook that is evaluated by modload is destroyed during the second sweep pass (count starts at 0). It is possible for a module to override what sweep pass the hook is destroyed on, in cases where there are additional ordering constraints for the objects created by a module. This isn't important from the Onyx language perspective, but is important to understand when implementing modules.

Example(s):

```
onyx:0> '/usr/local/share/onyx/nxmod/mdprompt.nxm' 'modprompt_init'
onyx:2> modload
onyx:0>
```

***mutex proc* monitor –:**

Input(s):

mutex: A mutex.

proc: Any object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Execute *proc* while holding *mutex*.

Example(s):

```
onyx:0> mutex {‘hello\n’ print} monitor flush
hello
onyx:0>
```

file symbol **mrequire –:**

Input(s):

file: A string that represents a module filename.

symbol: A string that represents the symbol name of a module initialization function to be executed.

Output(s): None.

Errors(s):

invalidfileaccess.

stackunderflow.

typecheck.

undefined.

undefinedfilename.

Description: Search for and load a module. The module is searched for by catenating a prefix, a "/", and *file* to form a file path. Prefixes are tried in the following order:

1. The ordered elements of the mpath_pre array, which is defined in onyxdict.
2. If defined, the ordered elements of the ONYX_MPATH environment variable, which is a colon-separated list.
3. The ordered elements of the mpath_post array, which is defined in onyxdict.

Example(s):

```
onyx:0> ‘modgtk.nxm’ ‘modgtk_init’ mrequire
onyx:0>
```

a b **mul** ***r*****:**

Input(s):

a: An integer or real.

b: An integer or real.

Output(s):

r: The product of a and b.

Errors(s):

stackunderflow.

typecheck.

Description: Return the product of a and b.

Example(s):

```
onyx:0> 3 17 mul 1 sprint
51
onyx:0> -5 -6 mul 1 sprint
30
onyx:0> 3.5 4.0 mul 1 sprint
1.400000e+01
onyx:0> -1.5 3 mul 1 sprint
-4.500000e+00
onyx:0>
```

– mutex *mutex*:

Input(s): None.

Output(s):

mutex: A mutex object.

Errors(s): None.

Description: Create a mutex.

Example(s):

```
onyx:0> mutex 1 sprint
-mutex-
onyx:0>
```

***objects* . . . *count* nbpop . . . :**

Input(s):

objects: Zero or more objects.

count: Number of *objects* to pop.

Output(s): None.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Remove the bottom *count objects* from ostack and discard them.

Example(s):

```
onyx:0> ‘a’ ‘b’ ‘c’ 2 nbpop pstack
‘c’
onyx:1>
```

***arrays count* ncat *array*:**

***stacks count* ncat *stack*:**

***strings count* ncat *string*:**

Input(s):

arrays: *count* arrays.

stacks: *count* stacks.

strings: *count* strings.

count: Number of *arrays*, *stacks*, or *strings* to catenate.

Output(s):

obj: The catenation of *arrays*, *stacks*, or *strings*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Catenate *count arrays*, *stacks*, or *strings*.

Example(s):

```
onyx:0> ['a'] ['b'] ['c'] 3 ncat 1 sprint
['a' 'b' 'c']
onyx:0> ('a') ('b') ('c') 3 ncat 1 sprint
('a' 'b' 'c')
onyx:0> 'a' 'b' 'c' 3 ncat 1 sprint
'abc'
onyx:0>
```

***a . . . b count* ndn *. . . b a*:**

Input(s):

a: An object.

. . . : $count - 2$ objects.

b: An object.

count: Number of objects to rotate downward.

Output(s):

. . . : $count - 2$ objects.

b: An object.

a: An object.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Rotate *count* objects on ostack down one position.

Example(s):

```
onyx:0> 'a' 'b' 'c' 'd' 'e' 4 ndn pstack
'b'
'e'
'd'
'c'
'a'
onyx:5>
```

***objects count* ndup *objects objects*:**

Input(s):

objects: Zero or more objects.

count: The number of *objects* do duplicate.

Output(s):

objects: The same objects that were passed in.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Create duplicates of the top *count* objects on ostack. For composite objects, the new object is a reference to the same composite object.

Example(s):

```
onyx:0> 'a' 'b' 'c' 2 ndup pstack
'c'
'b'
'c'
'b'
'a'
onyx:5>
```

***a b* ne *boolean*:**

Input(s):

a: An object.

b: An object.

Output(s):

boolean: True if a is not equal to b, false otherwise.

Errors(s):

stackunderflow.

Description: Compare two objects for inequality. Inequality has the following meaning, depending on the types of a and b:

array, condition, dict, file, hook, mutex, stack, thread: a and b are not equal unless they refer to the same memory.

operator: a and b are not equal unless they refer to the same function.

name, string: a and b are not equal iff they are lexically equivalent. A name can be equal to a string.

boolean: a and b are not equal unless they are the same value.

integer, real: a and b are not equal unless they are the same value.

Example(s):

```
onyx:0> mutex mutex ne 1 sprint
true
onyx:0> mutex dup ne 1 sprint
false
onyx:0> $foo 'foo' ne 1 sprint
false
onyx:0> $foo $bar ne 1 sprint
true
onyx:0> true false ne 1 sprint
true
onyx:0> true true ne 1 sprint
false
onyx:0> 1 1 ne 1 sprint
false
onyx:0> 1 2 ne 1 sprint
true
onyx:0> 1.0 1 ne 1 sprint
false
onyx:0> 1.0 1.1 ne 1 sprint
true
onyx:0>
```

***a* neg *r*:**

Input(s):

a: An integer.

Output(s):

r: The negative of a.

Errors(s):

stackunderflow.

typecheck.

Description: Return the negative of a.

Example(s):

```
onyx:0> 0 neg 1 sprint
0
onyx:0> 5 neg 1 sprint
-5
onyx:0> -5 neg 1 sprint
5
onyx:0> 3.14 neg 1 sprint
-3.140000e+00
onyx:0> -3.14 neg 1 sprint
3.140000e+00
onyx:0>
```

***a b* nip *b*:**

Input(s):

a: An object.

b: An object.

Output(s):

b: An object.

Errors(s):

stackunderflow.

Description: Remove the second to top object from ostack.

Example(s):

```
onyx:0> ‘a’ ‘b’ ‘c’
onyx:3> nip pstack
‘c’
‘a’
onyx:2>
```

***file* nonblocking *boolean*:**

Input(s):

file: A file object.

Output(s):

boolean: Nonb-blocking mode for *file*.

Errors(s):

stackunderflow.

typecheck.

Description: Get non-blocking mode for *file*.

Example(s):

```
onyx:0> '/tmp/foo' 'w' open
onyx:1> dup nonblocking 1 sprint
false
onyx:1> dup true setnonblocking
onyx:1> dup nonblocking 1 sprint
true
onyx:1>
```

***a* not *r*:**

Input(s):

a: An integer or boolean.

Output(s):

r: If a is an integer, the bitwise negation of a, otherwise the logical negation of a.

Errors(s):

stackunderflow.

typecheck.

Description: Return the bitwise negation of an integer, or the logical negation of a boolean.

Example(s):

```
onyx:0> true not 1 sprint
false
onyx:0> false not 1 sprint
true
onyx:0> 1 not 1 sprint
-2
onyx:0>
```

***objects count* npop –:**

Input(s):

objects: Zero or more objects.

count: Number of *objects* to pop.

Output(s): None.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Remove the top *count objects* from ostack and discard them.

Example(s):

```
onyx:0> 'a' 'b' 'c' 2 npop pstack
'a'
onyx:1>
```

***nanoseconds* nsleep –:**

Input(s):

nanoseconds: Minimum number of nanoseconds to sleep. Must be greater than 0.

Output(s): None.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Sleep for at least *nanoseconds* nanonseconds.

Example(s):

```
onyx:0> 1000 nsleep
onyx:0>
```

– null *null*:

Input(s): None.

Output(s):

null: A null object.

Errors(s): None.

Description: Create a null object.

Example(s):

```
onyx:0> null pstack
null
onyx:1>
```

***a . . . b count* nup *b a . . .*:**

Input(s):

a: An object.

. . .: $count - 2$ objects.

b: An object.

count: Number of objects to rotate upward.

Output(s):

b: An object.

a: An object.

. . .: $count - 2$ objects.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Rotate *count* objects on ostack up one position.

Example(s):

```
onyx:0> ‘a’ ‘b’ ‘c’ ‘d’ ‘e’ 4 nup pstack
‘d’
‘c’
‘b’
‘e’
‘a’
onyx:5>
```

***input submatch* offset *offset*:**

Input(s):

input: A string.

submatch: A substring of *input*.

Output(s):

offset: The integer offset of *submatch*, relative to the beginning of *input*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Get the offset of *submatch*, relative to the beginning of *input*. *submatch* must be a substring of *input*.

Example(s):

```
onyx:0> 'input' dup 'n(p)u' match {1 submatch offset 1 sprint} if
2
onyx:0>
```

– onyxdict *dict*:

Input(s): None.

Output(s):

dict: A dictionary.

Errors(s): None.

Description: Get onyxdict. See Section 2.10.6 for details on onyxdict.

Example(s):

```
onyx:0> onyxdict 1 sprint
<$rpath_pre -array- $rpath_post -array- $mpath_pre -array- $mpath_post -array->
onyx:0>
```

***filename flags* open *file*:**

***filename flags mode* open *file*:**

Input(s):

filename: A string that represents a filename.

flags: A string that represents a file mode:

- **'r':** Read only.
- **'r+':** Read/write, starting at offset 0.
- **'w':** Write only. Create file if necessary. Truncate file if non-zero length.
- **'w+':** Read/write, starting at offset 0. Create file if necessary.
- **'a':** Write only, starting at end of file.
- **'a+':** Read/write, starting at end of file.

mode: Mode to use when creating a new file (defaults to 0777). Note that the process's umask also affects creation mode.

Output(s):

file: A file object.

Errors(s):

invalidfileaccess.

ioerror.

limitcheck.

rangecheck.

stackunderflow.

typecheck.

Description: Open a file.

Example(s):

```
onyx:0> '/tmp/foo' 'w' open pstack
-file-
onyx:1>
```

***a b* or *r*:**

Input(s):

a: An integer or boolean.

b: The same type as a.

Output(s):

r: If a and b are integers, their bitwise or, otherwise their logical or.

Errors(s):

stackunderflow.

typecheck.

Description: Return the bitwise or of two integers, or the logical or of two booleans.

Example(s):

```
onyx:0> false false or 1 sprint
false
onyx:0> true false or 1 sprint
true
onyx:0> 5 3 or 1 sprint
7
onyx:0>
```

– ostack *stack*:

Input(s): None.

Output(s):

stack: A current snapshot (copy) of ostack.

Errors(s): None.

Description: Get a current snapshot of ostack.

Example(s):

```
onyx:0> 1 2 3 ostack pstack
(1 2 3)
3
2
1
onyx:4>
```

***obj depth* output –:**

Input(s):

obj: An object to print syntactically.

depth: Maximum recursion depth.

Output(s): None.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Syntactically print *obj*. See Section 2.10.7 for format specifier details.

Example(s):

```
onyx:0> [1 [2 3] 4] <$w 20 $p ‘_’ $j $c $r 1> output ‘\n’ print flush
___[1 -array- 4]____
onyx:0> [1 [2 3] 4] <$w 20 $p ‘_’ $j $c $r 2> output ‘\n’ print flush
____[1 [2 3] 4]_____
onyx:0> 4242 <$s $+> output ‘\n’ print flush
+4242
onyx:0> ‘0x’ print 4242 <$b 16> output ‘\n’ print flush
0x1092
onyx:0> ‘0x’ 4242 <$b 16> outputs cat <$w 10 $p ‘.’>
onyx:2>  output ‘\n’ print flush
....0x1092
onyx:0> ‘0x’ print 4242 <$w 8 $p ‘0’ $b 16> output ‘\n’ print flush
0x00001092
onyx:0>
```

***obj flags* outputs *string*:**

Input(s):

obj: An object to print syntactically.

depth: Formatting flags. See Section 2.10.7 for details on the supported flags.

Output(s):

string: A formatted string representation of *obj*. See Section 2.10.7 for format specifier details.

Errors(s):

stackunderflow.

typecheck.

Description: Create a formatted string representation of *obj*.

Example(s):

```
onyx:0> [1 [2 3] 4] <$w 20 $p ‘_’ $j $c $r 1> outputs print ‘\n’ print flush
___[1 -array- 4]____
onyx:0> [1 [2 3] 4] <$w 20 $p ‘_’ $j $c $r 2> outputs print ‘\n’ print flush
____[1 [2 3] 4]_____
onyx:0> 4242 <$s $+> outputs print ‘\n’ print flush
+4242
onyx:0> ‘0x’ print 4242 <$b 16> outputs print ‘\n’ print flush
0x1092
onyx:0> ‘0x’ 4242 <$b 16> outputs cat <$w 10 $p ‘.’> outputs
onyx:1> print ‘\n’ print flush
....0x1092
onyx:0> ‘0x’ print 4242 <$w 8 $p ‘0’ $b 16> outputs print ‘\n’ print flush
0x00001092
onyx:0>
```

– outputsdict *dict*:

Input(s): None.

Output(s):

dict: A dictionary.

Errors(s): None.

Description: Get outputsdict. See Section 2.10.7 for details on outputsdict.

Example(s):

```
onyx:0> outputsdict 0 sprint
-dict-
onyx:0>
```

***a b* over *a b a*:**

Input(s):

a: An object.

b: An object.

Output(s):

a: An object.

b: An object.

Errors(s):

stackunderflow.

Description: Create a duplicate of the second object on ostack and push it onto ostack.

Example(s):

```
onyx:0> 0 1 2 over pstack
1
2
1
0
onyx:4>
```

***sock* peername *dict*:**

Input(s):

sock: A socket.

Output(s):

dict: A dictionary of information about the peer end of *sock*. Depending on the socket family, the following entries may exist:

family: Socket family.

address: IPv4 address.

port: IPv4 port.

path: Unix-domain socket path.

Errors(s):

argcheck.

ioerror.

neterror.

stackunderflow.

typecheck.

unregistered.

Description: Get information about the peer end of *sock*.

Example(s):

```
onyx:0> $AF_INET $SOCK_STREAM socket
onyx:1> dup 'localhost' 7777 bindsocket
onyx:1> dup listen
onyx:1> dup accept
onyx:2> dup peername 1 sprint
<$family $AF_INET $address 2130706433 $port 33746>
onyx:2>
```

– pid *pid*:

Input(s): None.

Output(s):

pid: Process identifier.

Errors(s): None.

Description: Get the process ID of the running process.

Example(s):

```
onyx:0> pid 1 sprint
80624
onyx:0>
```

– pipe *rfile wfile*:

Input(s): None.

Output(s):

rfile: A readable file object. Data read from *rfile* were previously written to *wfile*.

wfile: A writeable file object. Data written to *wfile* can subsequently be read from *rfile*.

Errors(s):

ioerror.

unregistered.

Description: Create a pipe.

Example(s):

```
onyx:0> pipe
onyx:2> $wfile exch def
onyx:1> $rfile exch def
onyx:0> wfile 'foo\n' write
onyx:0> wfile flushfile
onyx:0> rfile readline pop 1 sprint
'foo'
onyx:0>
```

***<file flags ... > timeout* poll *[file ...]*:**

Input(s):

<...>: A dictionary of *file*/*flags* key/value pairs.

file: A file object.

flags: A dictionary that contains keys corresponding to file status attributes to poll. The following keys are heeded:

$POLLIN: Normal or priority data are available for reading.

$POLLRDNORM: Normal data are available for reading.

$POLLRDBAND: Priority data are available for reading.

$POLLPRI: High-priority data are available for reading.

$POLLOUT: Normal data can be written.

$POLLWRNORM: Normal data can be written.

$POLLWRBAND: Priority data can be written.

The values associated with the keys are disregarded, but are set appropriately before **poll** returns (true/false).

timeout: Timeout, in milliseconds (maximum $2^{31} - 1$). -1 is treated specially to mean infinite timeout.

Output(s):

[. . .]: An array containing a reference to each *file* in $<\ldots>$ for which a non-zero number of status attributes is set to true. A zero-length array indicates that the poll timed out.

file: A reference to a file object passed in that has one or more attributes set to true.

Although $<\ldots>$ is not returned, its contents are modified.

flags: The dictionary passed in. For recognized key that is defined, the associated value is set to true or false, depending on the status of *file*. In addition, the following keys may defined (if not already defined) with a value of true in the case of errors:

$POLLERR: An error has occurred.

$POLLHUP: Hangup has occurred.

$POLLNVAL: *file* is not an open file.

Errors(s):

stackunderflow.

rangecheck.

typecheck.

Description: Wait for any of the *flags* associated with a *file* in $<\ldots>$ to be true.

Example(s):

```
onyx:0> <stdout <$POLLOUT null> stderr <$POLLWRNORM null>> dup 0 poll
onyx:2> 2 sprint 2 sprint
[-file- -file-]
<-file- <$POLLWRNORM true> -file- <$POLLOUT true>>
onyx:0>
```

***obj* pop –:**

Input(s):

obj: An object.

Output(s): None.

Errors(s):

stackunderflow.

Description: Remove the top object from ostack and discard it.

Example(s):

```
onyx:0> 1 2
onyx:2> pstack
```

```
2
1
onyx:2> pop
onyx:1> pstack
1
onyx:1>
```

***a b* pow *r*:**

Input(s):

a: An integer or real.

b: An integer or real.

Output(s):

r: *a* to the *b* power.

Errors(s):

stackunderflow.

typecheck.

Description: Return *a* to the *b* power. If a negative exponent is specified, the result will always be a real, even if both arguments are integers.

Example(s):

```
onyx:0> 5 0 pow 1 sprint
1
onyx:0> 5 1 pow 1 sprint
5
onyx:0> 5 2 pow 1 sprint
25
onyx:0> -5 3 pow 1 sprint
-125
onyx:0> 5 -3 pow 1 sprint
8.000000e-03
onyx:0> 2.1 3.5 pow 1 sprint
1.342046e+01
onyx:0> 100 .01 pow 1 sprint
1.000000e+02
onyx:0>
```

– ppid *pid*:

Input(s): None.

Output(s):

pid: Process identifier.

Errors(s): None.

Description: Get the process ID of the running process's parent.

Example(s):

```
onyx:0> ppid 1 sprint
352
onyx:0>
```

***string* print –:**

Input(s):

string: A string object.

Output(s): None.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Print *string* to stdout.

Example(s):
```
onyx:0> 'Hi\n' print flush
Hi
onyx:0>
```

– product *string*:

Input(s): None.

Output(s):

string: A string that contains the product name, normally 'Canonware Onyx'.

Errors(s): None.

Description: Get the product string. The string returned is a reference to the original product string.

Example(s):
```
onyx:0> product pstack
'Canonware Onyx'
onyx:1>
```

– pstack –:

Input(s): None.

Output(s): None.

Errors(s):

ioerror.

Description: Syntactically print the elements of ostack, one per line.

Example(s):
```
onyx:0> 'a' 1 mark $foo [1 2 3] (4 5 6)
onyx:6> pstack
(4 5 6)
[1 2 3]
$foo
-mark-
1
'a'
onyx:6>
```

***array index obj* put –:**

***dict key value* put –:**

***string index integer* put –:**

Input(s):

array: An array object.

dict: A dict object.

string: A string object.

index: Offset in *array* or *string* to put *obj* or *integer*, respectively.

key: An object to use as a key in *dict*.

obj: An object to insert into *array* at offset *index*.

value: An object to associate with *key* in *dict*.

integer: The ascii value of a character to insert into *string* at offset *index*.

Output(s): None.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Insert into *array*, *dict*, or *string*.

Example(s):

```
onyx:0> 3 array dup 1 ‘a’ put 1 sprint
[null ‘a’ null]
onyx:0> dict dup $foo ‘foo’ put 1 sprint
<$foo ‘foo’>
onyx:0> 3 string dup 1 97 put 1 sprint
‘\x00a\x00’
onyx:0>
```

***array index subarray* putinterval –:**

***string index substring* putinterval –:**

Input(s):

array: An array object.

string: A string object.

index: Offset into *array* or *string* to put *subarray* or *substring*, respectively.

subarray: An array object to put into *array* at offset *index*. When inserted *subarray* must not extend past the end of *array*.

substring: A string object to put into *string* at offset *index*. When inserted *substring* must not extend past the end of *string*.

Output(s): None.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Replace a portion of *array* or *string*.

Example(s):

```
onyx:0> 4 array dup 1 [‘a’ ‘b’] putinterval 1 sprint
[null ‘a’ ‘b’ null]
onyx:0> 4 string dup 1 ‘ab’ putinterval 1 sprint
‘\x00ab\x00’
onyx:0>
```

– pwd *path*:

Input(s): None.

Output(s):

path: A string that represents the present working directory.

Errors(s):

invalidaccess.

Description: Push a string onto ostack that represents the present working directory.

Example(s):

```
onyx:0> pwd
onyx:1> pstack
'/usr/local/bin'
onyx:1>
```

– quit –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Unwind the execution stack to the innermost **start** context. Under normal circumstances, there is always at least one such context.

Example(s):

```
onyx:0> stdin cvx start
onyx:0> estack 1 sprint
(--start-- -file- --start-- -file- --estack--)
onyx:0> quit
onyx:0> estack 1 sprint
(--start-- -file- --estack--)
onyx:0>
```

– rand *integer*:

Input(s): None.

Output(s):

integer: A pseudo-random non-negative integer, with 63 bits of psuedo-randomness.

Errors(s): None.

Description: Return a pseudo-random integer.

Example(s):

```
onyx:0> 0 srand
onyx:0> rand 1 sprint
9018578418316157091
onyx:0> rand 1 sprint
8979240987855095636
onyx:0>
```

***file* read *integer boolean*:**

***file string* read *substring boolean*:**

Input(s):

file: A file object.

string: A string object.

Output(s):

integer: An integer that represents the ascii value of a character that was read from *file*.

substring: A substring of *string* that contains data read from *file*.

boolean: If true, end of file reached during read.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Read from *file*.

Example(s):

```
onyx:0> '/tmp/foo' 'w+' open
onyx:1> dup 'Hello\n' write
onyx:1> dup flushfile
onyx:1> dup 0 seek
onyx:1> dup 10 string read
onyx:3> pop 1 sprint
'Hello\n'
```

***file* readline *string boolean*:**

Input(s):

file: A file object.

Output(s):

string: A string that contains a line of text from *file*.

boolean: If true, end of file reached during read.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Read a line of text from *file*. Lines are separated by "\n" or "\r\n", which is removed. The last line in a file may not have a newline at the end.

Example(s):

```
onyx:0> '/tmp/foo' 'w+' open
onyx:1> dup 'Hello\n' write
onyx:1> dup 'Goodbye\n' write
onyx:1> dup 0 seek
onyx:1> dup readline 1 sprint 1 sprint
false
'Hello'
onyx:1> dup readline 1 sprint 1 sprint
false
'Goodbye'
onyx:1> dup readline 1 sprint 1 sprint
true
''
onyx:1>
```

***linkname* readlink *string*:**

Input(s):

linkname: A string that represents the path of a symbolic link.

Output(s):

string: A string that represents the link data associated with *linkname*.

Errors(s):

invalidaccess.

invalidfileaccess.

ioerror.

stackunderflow.

typecheck.

undefinedfilename.

unregistered.

Description: Get the data for the symbolic link at *linkname*.

Example(s):

```
onyx:0> ‘bar’ ‘foo’ symlink
onyx:0> ‘foo’ readlink 1 sprint
‘bar’
onyx:0>
```

– realtime *nsecs*:

Input(s): None.

Output(s):

nsecs: Number of nanoseconds since the epoch (midnight on 1 January 1970).

Errors(s): None.

Description: Get the number of nanoseconds since the epoch.

Example(s):

```
onyx:0> realtime 1 sprint
993539837806479000
onyx:0>
```

***sock string flags* recv *substring*:**

***sock string* recv *substring*:**

Input(s):

sock: A socket.

string: A string to use as a buffer for the message being received.

flags: An array of flag names. The following flags are supported:

$MSG_OOB

$MSG_PEEK

$MSG_WAITALL

Output(s):

substring: A substring of *string* that contains message data.

Errors(s):

argcheck.

neterror.

stackunderflow.

typecheck.

unregistered.

Description:

Example(s):

```
onyx:0> $AF_INET $SOCK_DGRAM socket
onyx:1> dup ‘localhost’ 7777 bindsocket
onyx:1> dup true setnonblocking
onyx:1> dup 10 string recv
onyx:2> 1 sprint
‘hello’
onyx:1>
```

string flags **regex** ***regex*****:**

string **regex** ***regex*****:**

Input(s):

string: A string that specifies a regular expression. See Section 2.9 for syntax.

flags: A dictionary of optional flags:

- **$c:** Continue where previous match ended. Don’t update the offset to start the next match from unless this match is successful. Defaults to false.
- **$g:** Continue where previous match ended. If the match is unsuccessful, update the offset to start the next match from to the beginning of *input*. Defaults to false.
- **$i:** Case insensitive. Defaults to false.
- **$m:** Treat input as a multi-line string. Defaults to false.
- **$s:** Treat input as a single line, so that the dot metacharacter matches any character, including a newline. Defaults to false.

Output(s):

regex: A regex object.

Errors(s):

regexerror.

stackunderflow.

typecheck.

Description: Create a regex object, according to *string* and *flags*.

Example(s):

```
onyx:0> ‘pattern’ regex 1 sprint
-regex-
onyx:0> ‘pattern’ <$g true> regex 1 sprint
-regex-
onyx:0>
```

pattern template flags **regsub** ***regsub*****:**

pattern template **regsub** ***regsub*****:**

Input(s):

pattern: A string that specifies a regular expression. See Section 2.9 for syntax.

template: A string that specifies a substitution template. See Section 2.9 for syntax.

flags: A dictionary of optional flags:

$g: Substitute all matches, if true, rather than just the first match. Defaults to false.

$i: Case insensitive. Defaults to false.

$m: Treat input as a multi-line string. Defaults to false.

$s: Treat input as a single line, so that the dot metacharacter matches any character, including a newline. Defaults to false.

Output(s):

regsub: A regsub object.

Errors(s):

regexerror.

stackunderflow.

typecheck.

Description: Create a regsub object, according to *pattern*, *template*, and *flags*.

Example(s):

```
onyx:0> ‘([a-z]+)’ ‘<\1>’ <$g true> regsub
onyx:1> 1 sprint
-regsub-
onyx:0>
```

***old new* rename –:**

Input(s):

old: A string object that represents a file path.

new: A string object that represents a file path.

Output(s): None.

Errors(s):

invalidfileaccess.

ioerror.

limitcheck.

stackunderflow.

typecheck.

undefinedfilename.

Description: Rename a file or directory from *old* to *new*.

Example(s):

```
onyx:0> ‘/tmp/tdir’ 8@755 mkdir
onyx:0> ‘/tmp/tdir’ ‘/tmp/ndir’ rename
onyx:0> ‘/tmp/ndir’ {1 sprint} dirforeach
‘.’
‘..’
onyx:0>
```

***count proc* repeat –:**

Input(s):

count: Number of times to evaluate *proc* (non-negative).

proc: An object to evaluate.

Output(s): None.

Errors(s):

rangecheck.
stackunderflow.
typecheck.

Description: Evaluate *proc count* times. This operator supports the **exit** operator.

Example(s):

```
onyx:0> 3 {'hi' 1 sprint} repeat
'hi'
'hi'
'hi'
onyx:0>
```

***file* require –:**

Input(s):

file: A string that represents a module filename.

Output(s): None.

Errors(s):

invalidfileaccess.
stackunderflow.
typecheck.
undefined.
undefinedfilename.

Description: Search for and evaluate an Onyx source file. The file is searched for by catenating a prefix, a "/", and *file* to form a file path. Prefixes are tried in the following order:

1. The ordered elements of the rpath_pre array, which is defined in onyxdict.
2. If defined, the ordered elements of the ONYX_RPATH environment variable, which is a colon-separated list.
3. The ordered elements of the rpath_post array, which is defined in onyxdict.

Example(s):

```
onyx:0> 'modgtk/modgtk_defs.nx' require
onyx:0>
```

***path* rmdir –:**

Input(s):

path: A string object that represents a directory path.

Output(s): None.

Errors(s):

invalidfileaccess.
ioerror.
stackunderflow.
typecheck.
unregistered.

Description: Remove an empty directory.

Example(s):

```
onyx:0> '/tmp/tdir' 8@755 mkdir
onyx:0> '/tmp/tdir' rmdir
onyx:0>
```

***region count amount* roll *rolled*:**

Input(s):

region: 0 or more objects to be rolled.

count: Number of objects in *region*.

amount: Amount by which to roll. If positive, roll upward. If negative, roll downward.

Output(s):

rolled: Rolled version of *region*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Roll the top *count* objects on ostack (not counting *count* and *amount*) by *amount* positions. A positive *amount* indicates an upward roll, whereas a negative *amount* indicates a downward roll.

Example(s):

```
onyx:0> 3 2 1 0
onyx:4> pstack
0
1
2
3
onyx:4> 3 1 roll
onyx:4> pstack
1
2
0
3
onyx:4> 3 -2 roll
onyx:4> pstack
2
0
1
3
onyx:4> 4 0 roll
onyx:4> pstack
2
0
1
3
onyx:4>
```

***a* round *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Integer round of *a*.

Errors(s):

stackunderflow.

typecheck.

Description: Round a to the nearest integer and return the result.

Example(s):

```
onyx:0> -1.51 round 1 sprint
-2
onyx:0> -1.49 round 1 sprint
-1
onyx:0> 0 round 1 sprint
0
onyx:0> 1.49 round 1 sprint
1
onyx:0> 1.51 round 1 sprint
2
onyx:0>
```

. . . ***amount*** **rot . . . :**

Input(s):

. . . : One or more objects.

amount: Number of positions to rotate the stack upward. A negative value causes downward rotation.

Output(s):

. . . : One or more objects.

Errors(s):

stackunderflow.

typecheck.

Description: Rotate the stack contents up *amount* positions.

Example(s):

```
onyx:0> 1 2 3 4 5 2 rot pstack clear
3
2
1
5
4
onyx:0> 1 2 3 4 5 -2 rot pstack clear
2
1
5
4
3
onyx:0>
```

stack **sadn –:**

Input(s):

stack: A stack object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Rotate the contents of *stack* down one position.

Example(s):

```
onyx:0> (1 2 3 4) dup sadn 1 sprint
(2 3 4 1)
onyx:0>
```

***stack* saup –:**

Input(s):

stack: A stack object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Rotate the contents of *stack* up one position.

Example(s):

```
onyx:0> (1 2 3 4) dup saup 1 sprint
(4 1 2 3)
onyx:0>
```

***stack* sbdup –:**

Input(s):

stack: A stack object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Duplicate the bottom object on *stack* and push it onto *stack*.

Example(s):

```
onyx:0> (2 1 0) dup sbdup pstack
(2 1 0 2)
onyx:1>
```

***stack* sbpop *obj*:**

Input(s):

stack: A stack object.

Output(s):

obj: An object.

Errors(s):

stackunderflow.

typecheck.

Description: Pop *obj* off the bottom of *stack*.

Example(s):

```
onyx:0> (1 2 3) dup sbpop pstack
1
(2 3)
onyx:2>
```

***stack obj* sbpush –:**

Input(s):

stack: A stack object.

obj: An object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Push *obj* onto the bottom of *stack*.

Example(s):

```
onyx:0> (0) dup 1 sbpush
onyx:1> pstack
(1 0)
onyx:1>
```

***stack* sclear –:**

Input(s):

stack: A stack object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Remove all objects on *stack*.

Example(s):

```
onyx:0> (1 2 3 4) dup sclear pstack
()
onyx:1>
```

***stack* scleartomark –:**

Input(s):

stack: A stack object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

unmatchedmark.

Description: Remove objects from *stack* down to and including the topmost mark.

Example(s):

```
onyx:0> (3 mark 1 0) dup scleartomark pstack
(3)
onyx:1>
```

***stack* scount *count*:**

Input(s):

stack: A stack object.

Output(s):

count: The number of objects on *stack*.

Errors(s):

stackunderflow.

typecheck.

Description: Get the number of objects on *stack*.

Example(s):

```
onyx:0> (1 2) scount 1 sprint
2
onyx:0>
```

stack **scounttomark** ***count*:**

Input(s):

stack: A stack object.

Output(s):

count: The depth of the topmost mark on *stack*.

Errors(s):

stackunderflow.

typecheck.

unmatchedmark.

Description: Get the depth of the topmost mark on *stack*.

Example(s):

```
onyx:0> (3 mark 1 0) scounttomark 1 sprint
2
onyx:0>
```

stack **sdn –:**

Input(s):

stack: A stack object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Rotate the top three objects on *stack* down one position.

Example(s):

```
onyx:0> (3 2 1 0) dup sdn pstack
(3 1 0 2)
onyx:1>
```

stack **sdup –:**

Input(s):

stack: A stack object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Duplicate the top object on *stack* and push it onto *stack*.

Example(s):

```
onyx:0> (1) dup sdup 1 sprint
(1 1)
onyx:0>
```

***string pattern* search *post pattern pre true*:**

***string pattern* search *string false*:**

Input(s):

string: A string object.

pattern: A string that represents a substring to search for in *string*.

Output(s):

post: The substring of *string* that follows the match.

pattern: The substring of *string* that matches the input *pattern*.

pre: The substring of *string* that precedes the match.

true: Success.

string: The same object as the input *string*.

false: Failure.

Errors(s):

stackunderflow.

typecheck.

Description: Search for the first instance of *pattern* in *string*, and if found, return substrings that partition *string* into *pre*, *pattern*, and *post*.

Example(s):

```
onyx:0> 'abcabc' 'ab' search pstack clear
true
''
'ab'
'cabc'
onyx:0> 'abcabc' 'ca' search pstack clear
true
'ab'
'ca'
'bc'
onyx:0> 'abcabc' 'cb' search pstack clear
false
'abcabc'
onyx:0>
```

***file offset* seek –:**

Input(s):

file: A file object.

offset: Offset in bytes from the beginning of *file* to move the file position pointer to.

Output(s): None.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Move the file position pointer for *file* to *offset*.

Example(s):

```
onyx:0> '/tmp/foo' 'w+' open
onyx:1> dup 'Hello\n' write
onyx:1> dup 0 seek
onyx:1> readline pstack
false
'Hello'
onyx:2>
```

– self *thread*:

Input(s): None.

Output(s):

thread: A thread object that corresponds to the running thread.

Errors(s): None.

Description: Get a thread object for the running thread.

Example(s):

```
onyx:0> self 1 sprint
-thread-
onyx:0>
```

***sock mesg flags* send *nsend*:**

***sock mesg* send *nsend*:**

Input(s):

sock: A socket.

mesg: A message string.

flags: An array of flag names. The following flags are supported:

$MSG_OOB

$MSG_PEEK

$MSG_WAITALL

Output(s):

nsend: Number of bytes of *mesg* actually sent.

Errors(s):

argcheck.

neterror.

stackunderflow.

typecheck.

unregistered.

Description: Send a message.

Example(s):

```
onyx:0> $AF_INET $SOCK_DGRAM socket
onyx:1> dup 'localhost' 7777 connect
```

```
onyx:1> dup 'hello' send
onyx:2> 1 sprint
5
onyx:1>
```

service **serviceport** ***port*****:**

Input(s):

service: A string that represents a network service name.

Output(s):

port: The port number corresponding to *service*, or 0 if the service is unknown.

Errors(s):

stackunderflow.

typecheck.

Description:

Example(s):

```
onyx:0> 'ftp' serviceport 1 sprint
21
onyx:0>
```

gid **setegid** ***boolean*****:**

Input(s):

gid: A group ID.

Output(s):

boolean: If false, success, otherwise failure.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Set the process's effective group ID to *gid*.

Example(s):

```
onyx:0> 1001 setegid 1 sprint
false
onyx:0> 0 setegid 1 sprint
true
onyx:0>
```

key val **setenv –:**

Input(s):

key: A name object.

val: A value to associate with *key*.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Set an environment variable named *key* and associate *val* with it. If *val* is not a string, it is converted to a string using the **cvs** operator before the environment variable is set. A corresponding entry is also created in the envdict dictionary.

Example(s):

```
onyx:0> $foo ‘foo’ setenv
onyx:0> envdict $foo known 1 sprint
true
onyx:0> envdict $foo get 1 sprint
‘foo’
onyx:0> $foo unsetenv
onyx:0> envdict $foo known 1 sprint
false
onyx:0>
```

***uid* seteuid *boolean*:**

Input(s):

uid: A user ID.

Output(s):

boolean: If false, success, otherwise failure.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Set the process's effective user ID to *uid*.

Example(s):

```
onyx:0> 1001 seteuid 1 sprint
false
onyx:0> 0 seteuid 1 sprint
true
onyx:0>
```

***gid* setgid *boolean*:**

Input(s):

gid: A group ID.

Output(s):

boolean: If false, success, otherwise failure.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Set the process's group ID to *gid*.

Example(s):

```
onyx:0> 1001 setgid 1 sprint
false
onyx:0> 0 setgid 1 sprint
true
onyx:0>
```

***file* setgstderr –:**

Input(s):

file: A file to set the global stderr to.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Set the global stderr to *file*. See Section 2.4 for standard I/O details.

Example(s):

```
onyx:0> '/tmp/stderr' 'w' open dup 0 setiobuf setgstderr
onyx:0> () {stderr 'Some text\n' write} thread join
onyx:0> '/tmp/stderr' 'r' open readline pop 1 sprint
'Some text'
onyx:0>
```

***file* setgstdin –:**

Input(s):

file: A file to set the global stdin to.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Set the global stdin to *file*. See Section 2.4 for standard I/O details.

Example(s): Under normal interactive operation, stdin is pushed onto estack during interpreter initialization and evaluated until EOF is reached. Therefore, changing stdin has no effect on the file descriptor already on estack. The following example recursively evaluates stdin after redefining it.

```
lawine:~> cat /tmp/stdin
1 2 3 pstack
lawine:~> onyx
Canonware Onyx, version 3.1.0.
onyx:0> '/tmp/stdin' 'r' open cvx setgstdin
onyx:0> () {stdin eval} thread join
3
2
1
onyx:0>
```

***file* setgstdout –:**

Input(s):

file: A file to set the global stdout to.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Set the global stdout to *file*. See Section 2.4 for standard I/O details.

Example(s): In the following example, the prompt continues to be printed, even though stdout has been redefined, because the prompt module was initialized to print to file descriptor 1. This demonstrates the only known exception in the stock Onyx interpreter where redefining stdout will not redirect output.

```
onyx:0> '/tmp/stdout' 'w' open dup 0 setiobuf setgstdout
onyx:0> () {'Some text\n' print} thread join
onyx:0> '/tmp/stdout' 'r' open readline pop 1 sprint
'Some text'
onyx:0>
```

file count **setiobuf –:**

Input(s):

file: A file object.

count: The size in bytes to set the I/O buffer associated with *file* to.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Set the size of the I/O buffer associated with *file*.

Example(s):

```
onyx:0> stdout iobuf 1 sprint
512
onyx:0> stdout 0 setiobuf
onyx:0> stdout iobuf 1 sprint
0
onyx:0>
```

boolean **setlocking –:**

Input(s):

boolean: A boolean to set the implicit locking mode to.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Set the current implicit locking mode. See Section 2.7.1 for implicit synchronization details.

Example(s):

```
onyx:0> currentlocking 1 sprint
false
onyx:0> true setlocking
onyx:0> currentlocking 1 sprint
true
onyx:0>
```

file boolean **setnonblocking –:**

Input(s):

file: A file object.

boolean: Non-blocking mode to set *file* to.

Output(s): None.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Set non-blocking mode for *file* to *boolean*.

Example(s):

```
onyx:0> '/tmp/foo' 'w' open
onyx:1> dup nonblocking 1 sprint
false
onyx:1> dup true setnonblocking
onyx:1> dup nonblocking 1 sprint
true
onyx:1>
```

***sock level optname optval* setsockopt –:**

***sock optname optval* setsockopt –:**

Input(s):

sock: A socket.

level: Level at which to set the socket option. If not specified, $SOL_SOCKET is used.

optname: Name of option to set the value of. The following option names are supported:

$SO_DEBUG

$SO_REUSEADDR

$SO_REUSEPORT

$SO_KEEPALIVE

$SO_DONTROUTE

$SO_BROADCAST

$SO_OOBINLINE

$SO_SNDBUF

$SO_RCVBUF

$SO_SNDLOWAT

$SO_RCVLOWAT

$SO_TYPE

$SO_ERROR: *optval* is an integer.

$SO_LINGER: *optval* is a dictionary, and the following entries are defined:

$on: Boolean.

$time: Linger time in seconds.

$SO_SNDTIMEO

$SO_RCVTIMEO: *optval* is an integer, in nanoseconds.

optval: Value to associate with *optname*.

Output(s): None.

Errors(s):

argcheck.

stackunderflow.

typecheck.

unregistered.

Description: Set a socket option.

Example(s):

```
onyx:0> $AF_INET $SOCK_STREAM socket
onyx:1> dup $SO_OOBINLINE sockopt 1 sprint
0
onyx:1> dup $SO_OOBINLINE 1 setsockopt
onyx:1> dup $SO_OOBINLINE sockopt 1 sprint
1
onyx:1>
```

file **setstderr –:**

Input(s):

file: A file to set the calling thread's stderr to.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Set the thread's stderr to *file*. See Section 2.4 for standard I/O details.

Example(s):

```
onyx:0> '/tmp/stderr' 'w' open dup 0 setiobuf setstderr
onyx:0> stderr 'Some text\n' write
onyx:0> '/tmp/stderr' 'r' open readline pop 1 sprint
'Some text'
onyx:0>
```

file **setstdin –:**

Input(s):

file: A file to set the calling thread's stdin to.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Set the thread's stdin to *file*. See Section 2.4 for standard I/O details.

Example(s): Under normal interactive operation, stdin is pushed onto estack during interpreter initialization and evaluated until EOF is reached. Therefore, changing stdin has no effect on the file descriptor already on estack. The following example recursively evaluates stdin after redefining it.

```
lawine:~> cat /tmp/stdin
1 2 3 pstack
lawine:~> onyx
Canonware Onyx, version 3.1.0.
onyx:0> '/tmp/stdin' 'r' open cvx setstdin
```

```
onyx:0> stdin eval
3
2
1
onyx:3>
```

***file* setstdout –:**

Input(s):

file: A file to set the calling thread's stdout to.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Set the thread's stdout to *file*. See Section 2.4 for standard I/O details.

Example(s): In the following example, the prompt continues to be printed, even though stdout has been redefined, because the prompt module was initialized to print to file descriptor 1. This demonstrates the only known exception in the stock Onyx interpreter where redefining stdout will not redirect output.

```
onyx:0> ‘/tmp/stdout’ ‘w’ open dup 0 setiobuf setstdout
onyx:0> ‘Some text\n’ print
onyx:0> gstdout setstdout
onyx:0> ‘/tmp/stdout’ ‘r’ open readline pop 1 sprint
‘Some text’
onyx:0>
```

***uid* setuid *boolean*:**

Input(s):

uid: A user ID.

Output(s):

boolean: If false, success, otherwise failure.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Set the process's user ID to *uid*.

Example(s):

```
onyx:0> 1001 setuid 1 sprint
false
onyx:0> 0 setuid 1 sprint
true
onyx:0>
```

***stack* sexch –:**

Input(s):

stack: A stack object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Exchange the top two objects on *stack*.

Example(s):

```
onyx:0> (1 2 3) dup sexch pstack
(1 3 2)
onyx:1>
```

– shift –:

Input(s):

a: An integer.

shift: An integer that represents a bitwise shift amount. Negative means right shift, and positive means left shift.

Output(s):

r: *a* shifted by *shift* bits.

Errors(s):

stackunderflow.

typecheck.

Description: Shift an integer bitwise.

Example(s):

```
onyx:0> 4 1 shift 1 sprint
8
onyx:0> 4 -1 shift 1 sprint
2
onyx:0>
```

***stack index* sibdup –:**

Input(s):

stack: A stack object.

index: Offset from bottom of *stack*, counting from 0, of the object to duplicate.

Output(s): None.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Create a duplicate of the object on *stack* that is at offset *index* from the bottom of *stack* and push it onto *stack*.

Example(s):

```
onyx:0> (3 2 1 0) dup 2 sibdup pstack
(3 2 1 0 1)
onyx:1>
```

***stack index* sibpop *obj*:**

Input(s):

stack: A stack object.

index: Offset from bottom of *stack*, counting from 0, of the object to remove from *stack*.

Output(s):

obj: An object removed from *stack*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Remove the *obj* from *stack* that is at offset *index* from the bottom of *stack*.

Example(s):

```
onyx:0> (0 1 2 3) dup 2 sibpop pstack
2
(0 1 3)
onyx:2>
```

stack index **sidup –:**

Input(s):

stack: A stack object.

index: Depth (count starts at 0) of the object to duplicate in *stack*.

Output(s): None.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Create a duplicate of the object on *stack* at depth *index* and push it onto *stack*.

Example(s):

```
onyx:0> (3 2 1 0) dup 2 sidup
onyx:1> 1 sprint
(3 2 1 0 2)
onyx:0>
```

condition **signal –:**

Input(s):

condition: A condition object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Signal a thread that is waiting on *condition*. If there are no waiters, this operator has no effect.

Example(s):

```
onyx:0> condition mutex dup lock ostack
onyx:3> {dup lock exch signal unlock}
onyx:4> thread 3 1 roll
onyx:3> dup 3 1 roll
onyx:4> wait unlock join
onyx:0>
```

a **sin** ***r*:**

Input(s):

a: An integer or real.

Output(s):

r: Sine of a in radians.

Errors(s):

stackunderflow.

typecheck.

Description: Return the sine of a in radians.

Example(s):

```
onyx:0> 0 sin 1 sprint
0.000000e+00
onyx:0> 1.570796 sin 1 sprint
1.000000e+00
onyx:0> 0.7853982 sin 1 sprint
7.071068e-01
onyx:0>
```

a **sinh** ***r*:**

Input(s):

a: An integer or real.

Output(s):

r: Hyperbolic sine of a.

Errors(s):

stackunderflow.

typecheck.

Description: Return the hyperbolic sine of a.

Example(s):

```
onyx:0> 3 sinh 1 sprint
1.001787e+01
onyx:0>
```

stack index **sipop** ***obj*:**

Input(s):

stack: A stack object.

index: Offset from top of *stack*, counting from 0, of the object to remove from *stack*.

Output(s):

obj: An object removed from *stack*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Remove the *obj* at *index* from *stack*.

Example(s):
```
onyx:0> (3 2 1 0) dup 2 sipop pstack
2
(3 1 0)
onyx:2>
```

***stack count* snbpop *array*:**

Input(s):

stack: A stack object.

count: Number of objects to pop off the bottom of *stack*.

Output(s):

array: An array of objects popped off the bottom of *stack*, with the same object ordering as when on *stack*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Pop *count* objects off the bottom of *stack* and put them into an array.

Example(s):
```
onyx:0> (1 2 3 4) dup 2 snbpop pstack
[1 2]
(3 4)
onyx:2>
```

***stack count* sndn –:**

Input(s):

stack: A stack object.

count: Number of objects on *stack* to rotate down one position.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Rotate *count* objects on *stack* down one position.

Example(s):
```
onyx:0> (5 4 3 2 1 0) dup 4 sndn pstack
(5 4 2 1 0 3)
onyx:1>
```

***stack count* sndup –:**

Input(s):

stack: A stack object.

count: Number of objects on *stack* to duplicate.

Output(s): None.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Create duplicates of the top *count* objects on *stack*.

Example(s):
```
onyx:0> (3 2 1 0) dup 2 sndup pstack
(3 2 1 0 1 0)
onyx:1>
```

***stack* snip *obj*:**

Input(s):

stack: A stack object.

Output(s):

obj: The object that was the second to top object on *stack*.

Errors(s):

stackunderflow.

typecheck.

Description: Remove the second to top object from *stack*.

Example(s):
```
onyx:0> (2 1 0) dup snip pstack
1
(2 0)
onyx:2>
```

***stack count* snpop *array*:**

Input(s):

stack: A stack object.

count: Number of objects to pop off of *stack*.

Output(s):

array: An array of objects popped off of *stack*, with the same object ordering as when on *stack*.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Pop *count* objects off of *stack* and put them into an array.

Example(s):
```
onyx:0> (1 2 3 4) dup 2 snpop pstack
[3 4]
(1 2)
onyx:2>
```

***stack count* snup –:**

Input(s):

stack: A stack object.

count: Number of objects on *stack* to rotate up one position.

Output(s): None.

Errors(s):

rangecheck.
stackunderflow.
typecheck.

Description: Rotate *count* objects on *stack* up one position.

Example(s):

```
onyx:0> (5 4 3 2 1 0) dup 4 snup pstack
(5 4 0 3 2 1)
onyx:1>
```

family type proto **socket** ***sock*****:**

family type **socket** ***sock*****:**

Input(s):

family: The name of a socket address family, either $AF_INET or $AF_LOCAL.

type: The name of a socket type, either $SOCK_STREAM or $SOCK_DGRAM.

proto: The name of a socket protocol. This argument is not useful, given the current limited choice of address families.

Output(s):

sock: A socket.

Errors(s):

argcheck.
invalidaccess.
stackunderflow.
typecheck.
unregistered.

Description: Create a socket.

Example(s):

```
onyx:0> $AF_INET $SOCK_STREAM socket
onyx:1> $AF_LOCAL $SOCK_DGRAM socket
onyx:2>
```

family type proto **socketpair** ***sock sock*****:**

family type **socketpair** ***sock sock*****:**

Input(s):

family: The name of a socket address family, either $AF_INET or $AF_LOCAL.

type: The name of a socket type, either $SOCK_STREAM or $SOCK_DGRAM.

proto: The name of a socket protocol. This argument is not useful, given the current limited choice of address families.

Output(s):

sock: A connected socket. There are no functional differences between the two sockets that are returned.

Errors(s):

argcheck.
invalidaccess.
stackunderflow.
typecheck.

unregistered.

Description: Create a pair of sockets that are connected to each other.

Example(s):

```
onyx:0> $AF_LOCAL $SOCK_STREAM socketpair
onyx:2> pstack
-file-
-file-
onyx:2>
```

***sock* sockname *dict*:**

Input(s):

sock: A socket.

Output(s):

dict: A dictionary of information about *sock*. Depending on the socket family, the following entries may exist:

family: Socket family.

address: IPv4 address.

port: IPv4 port.

path: Unix-domain socket path.

Errors(s):

argcheck.

ioerror.

neterror.

stackunderflow.

typecheck.

unregistered.

Description: Get information about *sock*.

Example(s):

```
onyx:0> $AF_INET $SOCK_STREAM socket
onyx:1> dup ‘localhost’ bindsocket
onyx:1> dup sockname 1 sprint
<$family $AF_INET $address 2130706433 $port 33745>
onyx:1> close
onyx:0> $AF_LOCAL $SOCK_STREAM socket
onyx:1> dup ‘/tmp/socket’ bindsocket
onyx:1> dup sockname 1 sprint
<$family $AF_LOCAL $path ‘/tmp/socket’>
onyx:1>
```

***sock level optname* sockopt *optval*:**

***sock optname* sockopt *optval*:**

Input(s):

sock: A socket.

level: Level at which to get the socket option. If not specified, $SOL_SOCKET is used.

optname: Name of option to get the value of. The following option names are supported:

$SO_DEBUG

$SO_REUSEADDR
$SO_REUSEPORT
$SO_KEEPALIVE
$SO_DONTROUTE
$SO_BROADCAST
$SO_OOBINLINE
$SO_SNDBUF
$SO_RCVBUF
$SO_SNDLOWAT
$SO_RCVLOWAT
$SO_TYPE
$SO_ERROR: *optval* is an integer.
$SO_LINGER: *optval* is a dictionary, and the following entries are defined:
- **$on:** Boolean.
- **$time:** Linger time in seconds.

$SO_SNDTIMEO
$SO_RCVTIMEO: *optval* is an integer, in nanoseconds.

Output(s):
- **optval:** Value associated with *optname*.

Errors(s):
- **argcheck.**
- **stackunderflow.**
- **typecheck.**
- **unregistered.**

Description: Get a socket option.

Example(s):
```
onyx:0> $AF_INET $SOCK_STREAM socket
onyx:1> dup $SO_SNDBUF sockopt 1 sprint
16384
onyx:1>
```

***stack* sover –:**

Input(s):
- **stack:** A stack object.

Output(s): None.

Errors(s):
- **stackunderflow.**
- **typecheck.**

Description: Create a duplicate of the second object on *stack* and push it onto *stack*.

Example(s):
```
onyx:0> (2 1 0) dup sover pstack
(2 1 0 1)
onyx:1>
```

***input pattern flags limit* split *array*:**

***input pattern flags* split *array*:**

***input pattern limit* split *array*:**

***input pattern* split *array*:**

***input regex limit* split *array*:**

***input regex* split *array*:**

Input(s):

input: An input string to find matches in.

pattern: A string that specifies a regular expression. See Section 2.9 for syntax.

flags: A dictionary of optional flags:

$i: Case insensitive. Defaults to false.

$m: Treat input as a multi-line string. Defaults to false.

$s: Treat input as a single line, so that the dot metacharacter matches any character, including a newline. Defaults to false.

regex: A regex object.

limit: Split *input* into no more than *limit* substrings. 0 is treated as infinity. Defaults to 0.

Output(s):

array: An array of substrings containing the text between pattern matches.

Errors(s):

rangecheck.

regexerror.

stackunderflow.

typecheck.

Description: Create an array of substrings from *input* that are separated by portions of *input* that match a regular expression.

If there are capturing subpatterns in the regular expression, also create substrings for those capturing subpatterns and insert them into the substring array.

As a special case, if the regular expression matches the empty string, split a single character. This avoids an infinite loop.

Example(s):

```
onyx:0> ‘a:b:c’ ‘:’ split 1 sprint
[‘a’ ‘b’ ‘c’]
onyx:0> ‘a:b:c’ ‘:’ 2 split 1 sprint
[‘a’ ‘b:c’]
onyx:0> ‘a:b:c’ ‘(:)’ split 1 sprint
[‘a’ ‘:’ ‘b’ ‘:’ ‘c’]
onyx:0> ‘a:b:c’ ‘’ split 1 sprint
[‘a’ ‘:’ ‘b’ ‘:’ ‘c’]
onyx:0>
```

***stack* spop *obj*:**

Input(s):

stack: A stack object.

Output(s):

obj: The object that was popped off of *stack*.

Errors(s):

stackunderflow.

typecheck.

Description: Pop an object off of *stack* and push it onto ostack.

Example(s):

```
onyx:0> (1 2) dup spop
onyx:2> pstack
2
(1)
onyx:2>
```

obj depth **sprint –:**

Input(s):

obj: An object to print syntactically.

depth: Maximum recursion depth.

Output(s): None.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Syntactically print *obj*. See Section 2.10.8 for printing details.

Example(s):

```
onyx:0> [1 [2 3] 4]
onyx:1> dup 0 sprint
-array-
onyx:1> dup 1 sprint
[1 -array- 4]
onyx:1> dup 2 sprint
[1 [2 3] 4]
onyx:1>
```

obj depth **sprints *string*:**

Input(s):

obj: An object to print syntactically.

depth: Maximum recursion depth.

Output(s):

string: A syntactical string representation of *obj*. See Section 2.10.8 for printing details.

Errors(s):

stackunderflow.

typecheck.

Description: Create a syntactical string representation of *obj*.

Example(s):

```
onyx:0> [1 [2 3] 4]
onyx:1> dup 0 sprints print ‘\n’ print flush
-array-
onyx:1> dup 1 sprints print ‘\n’ print flush
```

```
[1 -array- 4]
onyx:1> dup 2 sprints print '\n' print flush
[1 [2 3] 4]
onyx:1>
```

– sprintsdict *dict*:

Input(s): None.

Output(s):

dict: A dictionary.

Errors(s): None.

Description: Get sprintsdict. See Section 2.10.8 for details on sprintsdict.

Example(s):

```
onyx:0> sprintsdict 0 sprint
-dict-
onyx:0>
```

***stack obj* spush –:**

Input(s):

stack: A stack object.

obj: An object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Push *obj* onto *stack*.

Example(s):

```
onyx:0> (0) dup 1 spush
onyx:1> pstack
(0 1)
onyx:1>
```

***a* sqrt *r*:**

Input(s):

a: A non-negative integer or real.

Output(s):

r: Square root of a.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Return the square root of a.

Example(s):

```
onyx:0> 4 sqrt 1 sprint
2.000000e+00
onyx:0> 2.0 sqrt 1 sprint
1.414214e+00
onyx:0>
```

***seed* srand –:**

Input(s):

seed: A non-negative integer.

Output(s): None.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Seed the pseudo-random number generator with *seed*.

Example(s):

```
onyx:0> 5 srand
onyx:0>
```

***stack count amount* sroll –:**

Input(s):

stack: A stack object.

count: Number of objects to roll in *stack*.

amount: Amount by which to roll. If positive, roll upward. If negative, roll downward.

Output(s): None.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Roll the top *count* objects on *stack* by *amount* positions. A positive *amount* indicates an upward roll, whereas a negative *amount* indicates a downward roll.

Example(s):

```
onyx:0> (3 2 1 0)
onyx:1> dup 3 1 sroll pstack
(3 0 2 1)
onyx:1> dup 3 -2 sroll pstack
(3 1 0 2)
onyx:1> dup 4 0 sroll pstack
(3 1 0 2)
onyx:1>
```

***stack amount* srot –:**

Input(s):

stack: One or more objects.

amount: Number of positions to rotate *stack* upward. A negative value causes downward rotation.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Rotate *stack* up *count* positions.

Example(s):

```
onyx:0> (1 2 3 4 5) dup 2 srot 1 sprint
(4 5 1 2 3)
onyx:0> (1 2 3 4 5) dup -2 srot 1 sprint
(3 4 5 1 2)
onyx:0>
```

– stack *stack*:

Input(s): None.

Output(s):

stack: An empty stack object.

Errors(s): None.

Description: Create a new stack object and push it onto ostack.

Example(s):

```
onyx:0> stack
onyx:1> pstack
()
```

***obj* start –:**

Input(s):

obj: An object.

Output(s): None.

Errors(s):

stackunderflow.

Description: Evaluate *obj*. This operator provides a context that silently terminates execution stack unwinding due to the **exit**, **quit**, and **stop** operators.

Example(s):

```
onyx:0> stdin cvx start
onyx:0> quit
onyx:0>
```

***file/filename* status *dict*:**

Input(s):

file: A file object.

filename: A string that represents a filename.

Output(s):

dict: A dictionary that contains the following entries:

dev: Inode's device.

ino: Inode's number.

mode: Inode permissions.

nlink: Number of hard links.

uid: User ID of the file owner.

gid: Group ID of the file owner.

rdev: Device type.

size: File size in bytes.

atime: Time of last access, in nanoseconds since the epoch.

mtime: Time of last modification, in nanoseconds since the epoch.

ctime: Time of last file status change, in nanoseconds since the epoch.

blksize: Optimal block size for I/O.

blocks: Number of blocks allocated.

Errors(s):

invalidfileaccess.

ioerror.

stackunderflow.

typecheck.

unregistered.

Description: Get status information about a file.

Example(s):

```
onyx:0> '/tmp' status 1 sprint
<$dev 134405 $ino 2 $mode 17407 $nlink 5 $uid 0 $gid 0 $rdev 952 $size 3584
$atime 994883041000000000 $mtime 994883041000000000 $ctime 994883041000000000
$blksize 0 $blocks 8>
onyx:0>
```

– stderr *file*:

Input(s): None.

Output(s):

file: A file object corresponding to the calling thread's stderr.

Errors(s): None.

Description: Get the thread's stderr. See Section 2.4 for standard I/O details.

Example(s):

```
onyx:0> stderr pstack
-file-
onyx:1>
```

– stdin *file*:

Input(s): None.

Output(s):

file: A file object corresponding to the calling thread's stdin.

Errors(s): None.

Description: Get the thread's stdin. See Section 2.4 for standard I/O details.

Example(s):

```
onyx:0> stdin pstack
-file-
onyx:1>
```

– stdout *file*:

Input(s): None.

Output(s):

file: A file object corresponding to the calling thread's stdout.

Errors(s): None.

Description: Get the thread's stdout. See Section 2.4 for standard I/O details.

Example(s):
```
onyx:0> stdout pstack
-file-
onyx:1>
```

– stop –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Unwind the execution stack to the innermost **stopped** or **start** context.

Example(s):
```
onyx:0> {stop} stopped 1 sprint
true
onyx:0>
```

***obj* stopped *boolean*:**

Input(s):

obj: An object to evaluate.

Output(s):

boolean: True if stop operator was executed, false otherwise.

Errors(s):

invalidexit.

stackunderflow.

Description: Evaluate *obj*. This operator provides a context that terminates execution stack unwinding due to the **stop**. It will also terminate execution stack unwinding due to the **exit** operator, but will throw an invalidexit error, then do the equivalent of calling **quit**.

Example(s):
```
onyx:0> {stop} stopped 1 sprint
true
onyx:0> {} stopped 1 sprint
false
onyx:0>
```

***length* string *string*:**

Input(s):

length: Non-negative number of bytes.

Output(s):

string: A string of *length* bytes.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Create a string of *length* bytes. The bytes are initialized to 0.

Example(s):

```
onyx:0> 3 string 1 sprint
'\x00\x00\x00'
onyx:0>
onyx:0> 0 string 1 sprint
''
onyx:0>
```

stack **stuck –:**

Input(s):

stack: A stack object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Tuck duplicate of top object on *stack* under second object on *stack*.

Example(s):

```
onyx:0> (2 1 0) dup stuck pstack
(2 0 1 0)
onyx:1>
```

a b **sub** ***r*****:**

Input(s):

a: An integer or real.

b: An integer or real.

Output(s):

r: The value of b subtracted from a.

Errors(s):

stackunderflow.

typecheck.

Description: Subtract b from a and return the result.

Example(s):

```
onyx:0> 5 3 sub 1 sprint
2
onyx:0> -3 4 sub 1 sprint
-7
onyx:0> 5.1 1.1 sub 1 sprint
4.000000e+00
onyx:0> 5 1.0 sub 1 sprint
4.000000e+00
onyx:0> -3.0 4.1 sub 1 sprint
-7.100000e+00
onyx:0>
```

integer **submatch** ***substring*****:**

Input(s):

integer:

0: Get substring of text that matched the regular expression.

>**0:** Get substring of text that matched the specified capturing subpattern.

Output(s):

substring: A substring of the string that was most recently matched by the **match**, **split**, or **subst** operators.

Errors(s):

stackunderflow.

typecheck.

Description: Get a substring of the input string that was most recently matched against.

Example(s):

```
onyx:0> ‘input’ ‘n(p)u’ match {0 submatch 1 sprint 1 submatch 1 sprint} if
‘npu’
‘p’
onyx:0>
```

input pattern template flags **subst** ***output count*:**

input pattern template **subst** ***output count*:**

input regsub **subst** ***output count*:**

Input(s):

input: An input string.

pattern: A string that specifies a regular expression. See Section 2.9 for syntax.

template: A string that specifies a substitution template. See Section 2.9 for syntax.

flags: A dictionary of optional flags:

$g: Substitute all matches, if true, rather than just the first match. Defaults to false.

$i: Case insensitive. Defaults to false.

$m: Treat input as a multi-line string. Defaults to false.

$s: Treat input as a single line, so that the dot metacharacter matches any character, including a newline. Defaults to false.

regsub: A regsub object.

Output(s):

output: A string that is created by substituting substrings within *input* that match a regular expression.

count: Number of substitutions made. If 0 substitutions were made, then *output* is a duplicate of *input*, rather than a copy.

Errors(s):

regexerror.

stackunderflow.

typecheck.

Description: Create a string by substituting according to a template for each substring within *input* that matches a regular expression.

Example(s):

```
onyx:0> ‘Input String’ ‘([a-r])’ ‘[\1]’ <$g true> subst pstack
6
‘I[n][p]ut St[r][i][n][g]’
onyx:2>
```

***stack* sunder –:**

Input(s):

stack: A stack object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Create a duplicate of the second object on *stack* and put it under the top object on *stack*.

Example(s):

```
onyx:0> (2 1 0) dup sunder pstack
(2 1 1 0)
onyx:1>
```

***stack* sup –:**

Input(s):

stack: A stack object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Rotate the top three objects on *stack* up one position.

Example(s):

```
onyx:0> (3 2 1 0) dup sup pstack
(3 0 2 1)
onyx:1>
```

***filename linkname* symlink –:**

Input(s):

filename: A string that represents a filename.

linkname: A string that represents a filename.

Output(s): None.

Errors(s):

invalidfileaccess.

ioerror.

stackunderflow.

typecheck.

undefinedfilename.

unregistered.

Description: Create a symbolic link from *linkname* to *filename*.

Example(s):

```
onyx:0> ‘/tmp/foo’ ‘w’ open
onyx:1> dup ‘Hello\n’ write
onyx:1> dup flushfile
onyx:1> close
```

```
onyx:0> '/tmp/foo' '/tmp/bar' symlink
onyx:0> '/tmp/bar' 'r' open
onyx:1> readline
onyx:2> pstack
false
'Hello'
onyx:2>
```

***args* system *status*:**

Input(s):

args: An array of strings. The first string in *args* is the path of the program to invoke, and any additional array elements are passed as command line arguments to the invoked program.

Output(s):

status: Exit code of terminated process. A negative value indicates that the process was terminated by a signal (use the **neg** operator to get the signal number), and a non-negative value is the exit code of a program that terminated normally.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Execute a program as a child process and wait for it to terminate.

Example(s):

```
onyx:0> ['/usr/bin/which' 'onyx'] system
/usr/local/bin/onyx
onyx:1> 1 sprint
0
onyx:0>
```

– systemdict *dict*:

Input(s): None.

Output(s):

dict: A dictionary.

Errors(s): None.

Description: Get systemdict. See Section 2.10.9 for details on systemdict.

Example(s):

```
onyx:0> systemdict 0 sprint
-dict-
onyx:0>
```

***a* sinh *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Tangent of a in radians.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Return the tangent of a in radians.

Example(s):

```
onyx:0> 0.785 tan 1 sprint
9.992040e-01
onyx:0>
```

***a* tanh *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Hyperbolic tangent of a.

Errors(s):

stackunderflow.

typecheck.

Description: Return the hyperbolic tangent of a.

Example(s):

```
onyx:0> 3 tanh 1 sprint
9.950548e-01
onyx:0>
```

***file* tell *offset*:**

Input(s):

fil: A file object.

Output(s):

offset: Offset of the file position pointer for *file*.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Get the file position pointer offset for *file*.

Example(s):

```
onyx:0> ‘/tmp/foo’ ‘w+’ open
onyx:1> dup tell 1 sprint
0
onyx:1> dup ‘Hello\n’ write
onyx:1> dup tell 1 sprint
6
onyx:1>
```

***file*/*filename* *flag* test *boolean*:**

Input(s):

file: A file object.

filename: A string that represents a filename.

flag: A single-character string that represents the test to do on *file* or *filename*:

'b': Block special device?
'c': Character special device?
'd': Directory?
'e': Exists?
'f': Regular file?
'g': Setgid?
'k': Sticky?
'p': Named pipe?
'r': Readable?
's': Size greater than 0?
't': tty?
'u': Setuid?
'w': Write bit set?
'x': Executable bit set?
'L': Symbolic link?
'O': Owner matches effective uid?
'G': Group matches effective gid?
'S': Socket?

Output(s):

boolean: If true, the test evaluated to true; false otherwise.

Errors(s):

invalidfileaccess.
ioerror.
rangecheck.
stackunderflow.
typecheck.
unregistered.

Description: Test a file for an attribute.

Example(s):

```
onyx:0> '/blah' 'e' test 1 sprint
false
onyx:0> '/tmp' 'e' test 1 sprint
true
onyx:0>
```

***stack entry* thread *thread*:**

Input(s):

stack: A stack that contains the contents for the new thread's ostack.
entry: An initial object to execute in the new thread.

Output(s):

thread: A thread object that corresponds to the new thread.

Errors(s):

stackunderflow.
typecheck.

Description: Create and run a new thread.

Example(s):

```
onyx:0> (1 2) {add 1 sprint} thread join 'Done\n' print flush
3
Done
onyx:0>
```

***thread* threaddstack *stack*:**

Input(s):

thread: A thread object.

Output(s):

stack: The dictionary stack belonging to *thread*.

Errors(s):

stackunderflow.

typecheck.

Description: Get a reference to the dictionary stack belonging to *thread*.

Example(s):

```
onyx:0> self threaddstack 1 sprint
(-dict- -dict- -dict- -dict-)
onyx:0>
```

***thread* threadestack *stack*:**

Input(s):

thread: A thread object.

Output(s):

stack: The execution stack belonging to *thread*.

Errors(s):

stackunderflow.

typecheck.

Description: Get a reference to the execution stack belonging to *thread*.

Example(s):

```
onyx:0> self threadestack 1 sprint
(-file- -array- --eval-- --ifelse-- -array- --for-- -array-)
onyx:0>
```

***thread* threadistack *stack*:**

Input(s):

thread: A thread object.

Output(s):

stack: The index stack belonging to *thread*.

Errors(s):

stackunderflow.

typecheck.

Description: Get a reference to the index stack belonging to *thread*.

Example(s):

```
onyx:0> self threadistack 1 sprint
(0 0 0 0 7 0 3)
onyx:0>
```

***thread* threadostack *stack*:**

Input(s):

thread: A thread object.

Output(s):

stack: The operand stack belonging to *thread*.

Errors(s):

stackunderflow.

typecheck.

Description: Get a reference to the operand stack belonging to *thread*.

Example(s):

```
onyx:0> self threadostack 1 sprint
(-stack- -stack- -stack- -stack-)
onyx:0>
```

– threadsdict *dict*:

Input(s): None.

Output(s):

dict: A dictionary. Each key is a thread reference. By default, each value is null, but this need not be so, and the value can be redefined for debugging purposes.

Errors(s): None.

Description: Get a dictionary containing references to all threads.

Example(s):

```
onyx:0> threadsdict 1 sprint
<-thread- null>
onyx:0>
```

***name* throw *obj*:**

Input(s):

name: The name of an error.

Output(s):

obj: The object that was being executed when the error was thrown.

Errors(s):

stackunderflow.

typecheck.

undefined.

Description: Throw an error, using the following steps:

1. Set newerror in the currenterror dictionary to true.
2. Set errorname in the currenterror dictionary to *name*.
3. Set ostack, dstack, estack, and istack in the currenterror dictionary to be current stack snapshots.
4. Push the object that was being executed before throw was called onto ostack.
5. If there is an error handler in the errordict dictionary that corresponds to *name*, evaluate it. Otherwise, evaluate errordict's **handleerror** and **stop** operators.

Example(s):

```
onyx:0> $unregistered throw
Error $unregistered
ostack: ()
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..1):
0:      -file-
1:      --start--
onyx:1> pstack
-file-
onyx:1>
```

***condition mutex timeout* timedwait *boolean*:**

Input(s):

condition: A condition object.

mutex: A mutex object that this thread currently owns.

timeout: Minimum number of nanoseconds to wait for *condition*.

Output(s):

boolean: If false, success, otherwise timeout.

Errors(s):

stackunderflow.

typecheck.

Description: Wait on *condition* for at least *timeout* nanoseconds. *mutex* is atomically released when the current thread blocks, then acquired again before the current thread runs again. Using a mutex that the current thread does not own will result in undefined behavior (likely crash).

Example(s):

```
onyx:0> condition mutex dup lock ostack
onyx:3> {dup lock exch signal unlock}
onyx:4> thread 3 1 roll
onyx:3> dup 3 1 roll
onyx:4> 1000000000 timedwait 1 sprint unlock join
false
onyx:0> mutex condition 1 idup dup lock 1000000000 timedwait 1 sprint unlock
true
onyx:0>
```

***file/string* token *false*:**

***file/string* token *file/substring obj true*:**

Input(s):

file: A file that is used as onyx source code to scan a token from.

string: A string that is used as onyx source code to scan a token from.

Output(s):

file: The same file object that was passed in.

substring: The remainder of *string* after scanning a token.

obj: An object that was constructed by scanning a token.

false/true: If true, a token was successfully scanned, false otherwise.

Errors(s):

stackunderflow.

syntaxerror.

typecheck.

undefined.

Description: Scan a token from a file or string, using onyx syntax rules. If a token is followed by whitespace, one character of whitespace is consumed when the token is scanned.

Example(s):

```
onyx:0> ‘1 2’ token pstack clear
true
1
‘2’
onyx:0> ‘foo’ token pstack clear
true
foo
‘’
onyx:0> ‘foo ’ token pstack clear
true
foo
‘’
onyx:0> ‘foo  ’ token pstack clear
true
foo
‘ ’
onyx:0> ‘foo$bar’ token pstack clear
true
foo
‘$bar’
onyx:0> ‘foo{}’ token pstack clear
true
foo
‘{}’
onyx:0> ‘ ’ token pstack clear
false
onyx:0>
```

***a* trunc *r*:**

Input(s):

a: An integer or real.

Output(s):

r: Integer created from a by discarding the fractional portion.

Errors(s):

stackunderflow.

typecheck.

Description: Discard the fractional portion of a to create an integer, and return the result.

Example(s):

```
onyx:0> -1.51 trunc 1 sprint
-1
```

```
onyx:0> -1.49 trunc 1 sprint
-1
onyx:0> 0 trunc 1 sprint
0
onyx:0> 1.49 trunc 1 sprint
1
onyx:0> 1.51 trunc 1 sprint
1
onyx:0>
```

***file length* truncate –:**

Input(s):

file: A file object.

length: New length for *file*.

Output(s): None.

Errors(s):

ioerror.

rangecheck.

stackunderflow.

typecheck.

Description: Set the length of *file* to *length*. If this causes the file to grow, the appended bytes will have the value zero.

Example(s):

```
onyx:0> '/tmp/foo' 'w+' open
onyx:1> dup 'Hello\n' write
onyx:1> dup flushfile
onyx:1> dup 0 seek
onyx:1> dup 10 string read
onyx:3> pop 1 sprint
'Hello\n'
onyx:1> dup 3 truncate
onyx:1> dup 0 seek
onyx:1> dup 10 string read
onyx:3> pop 1 sprint
'Hel'
onyx:1>
```

– true *true*:

Input(s): None.

Output(s):

true: The boolean value true.

Errors(s): None.

Description: Return true.

Example(s):

```
onyx:0> true 1 sprint
true
onyx:0>
```

***mutex* trylock *boolean*:**

Input(s):

mutex: A mutex object.

Output(s):

boolean: If false, *mutex* was successfully acquired. Otherwise the mutex acquisition failed.

Errors(s):

stackunderflow.

typecheck.

Description: Try to acquire *mutex*, but return a failure immediately if *mutex* cannot be acquired, rather than blocking.

Example(s):

```
onyx:0> mutex dup
onyx:2> trylock 1 sprint
false
onyx:1> trylock 1 sprint
true
onyx:0>
```

***a b* tuck *b a b*:**

Input(s):

a: An object.

b: An object.

Output(s):

:

Errors(s):

stackunderflow.

typecheck.

Description: Create a duplicate of the top object on ostack and put it under the second object on ostack.

Example(s):

```
onyx:0> ‘a’ ‘b’ ‘c’
oonyx:3> tuck pstack
‘c’
‘b’
‘c’
‘a’
onyx:4>
```

***obj* type *name*:**

Input(s):

obj: An object.

Output(s):

name: An executable name that corresponds to the type of *obj*:

array: arraytype.

boolean: booleantype.

condition: conditiontype.

dict: dicttype.
file: filetype.
fino: finotype.
hook: hooktype.
integer: integertype.
mark: marktype.
mutex: mutextype.
name: nametype.
null: nulltype.
operator: operatortype.
pmark: pmarktype.
stack: stacktype.
string: stringtype.
thread: threadtype.

Errors(s):

stackunderflow.

Description: Get a name that represent the type of *obj*.

Example(s):

```
onyx:0> true type 1 sprint
booleantype
onyx:0>
```

– uid *uid*:

Input(s): None.

Output(s):

uid: Process's user ID.

Errors(s): None.

Description: Get the process's user ID.

Example(s):

```
onyx:0> uid 1 sprint
1001
onyx:0>
```

***nmask* umask *omask*:**

Input(s):

nmask: Value to set umask to.

Output(s):

omask: Old umask.

Errors(s):

stackunderflow.
typecheck.

Description: Set the process's umask to *nmask* and return the old umask.

Example(s):

```
onyx:0> 8@777 umask <$b 8 $w 3 $p '0'> output '\n' print flush
022
onyx:0>
```

dict key **undef –:**

Input(s):

dict: A dictionary.

val: A key in *dict* to undefine.

Output(s): None

Errors(s):

stackunderflow.

typecheck.

Description: If *key* is defined in *dict*, undefine it.

Example(s):

```
onyx:0> $foo 'foo' def
onyx:0> currentdict $foo undef
onyx:0> currentdict $foo undef
onyx:0>
```

a b **under** ***a a b*****:**

Input(s):

a: An object.

b: An object.

Output(s):

a: An object.

b: An object.

Errors(s):

stackunderflow.

Description: Create a duplicate of the second object on ostack and put it under the top object on ostack.

Example(s):

```
onyx:0> 0 1 2 under pstack
2
1
1
0
onyx:4>
```

boolean obj **unless –:**

Input(s):

boolean: A boolean.

obj: An object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Evaluate *obj* unless *boolean* is true.

Example(s):

```
onyx:0> false {‘yes’ 1 sprint} unless
‘yes’
onyx:0> true {‘yes’ 1 sprint} unless
onyx:0>
```

***filename* unlink –:**

Input(s):

filename: A string that represents a filename.

Output(s): None.

Errors(s):

invalidfileaccess.

ioerror.

stackunderflow.

typecheck.

undefinedfilename.

unregistered.

Description: Unlink *filename*.

Example(s):

```
onyx:0> ‘/tmp/foo’ ‘w’ open
onyx:1> dup ‘Hello\n’ write
onyx:1> dup flushfile
onyx:1> close
onyx:0> ‘/tmp/foo’ unlink
onyx:0> ‘/tmp/foo’ ‘r’ open
Error $invalidfileaccess
ostack: (‘/tmp/foo’ ‘r’)
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      --open--
1:      -file-
2:      --start--
onyx:3>
```

***mutex* unlock –:**

Input(s):

mutex: A mutex object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Unlock *mutex*. Unlocking a mutex that the running thread does not own will result in undefined behavior (likely crash).

Example(s):

```
onyx:0> mutex dup lock unlock
onyx:0>
```

***key* unsetenv –:**

Input(s):

key: A name object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Unset *key* in the environment and in the envdict dictionary, if *key* is defined.

Example(s):

```
onyx:0> $foo 'foo' setenv
onyx:0> envdict $foo known 1 sprint
true
onyx:0> envdict $foo get 1 sprint
'foo'
onyx:0> $foo unsetenv
onyx:0> envdict $foo known 1 sprint
false
onyx:0>
```

***proc cond* until –:**

Input(s):

proc: An object to be repeatedly evaluated.

cond: An object that, when evaluated, places a boolean on ostack.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Repeatedly evaluate *proc* and *cond*, terminating the first time that *cond* places false on ostack. This operator supports the **exit** operator.

Example(s):

```
onyx:0> 0 {inc dup 1 sprint}{dup 3 lt} until pop
1
2
3
onyx:0> 0 {inc dup 1 sprint exit}{dup 3 lt} until pop
1
onyx:0>
```

***a b c* up *c a b*:**

Input(s):

a: An object.

b: An object.

c: An object.

Output(s):

c: An object.

a: An object.

b: An object.

Errors(s):

stackunderflow.

Description: Rotate the top three objects on ostack up one position.

Example(s):

```
onyx:0> 'a' 'b' 'c' 'd' up pstack
'c'
'b'
'd'
'a'
onyx:4>
```

– userdict *dict*:

Input(s): None.

Output(s):

dict: A dictionary.

Errors(s): None.

Description: Get userdict. See Section 2.10.12 for details on userdict.

Example(s):

```
onyx:0> userdict 1 sprint
<>
onyx:0>
```

– version *string*:

Input(s): None.

Output(s):

string: A string that contains the version name.

Errors(s): None.

Description: Get the version string. The string returned is a reference to the original version string.

Example(s):

```
onyx:0> version pstack
'1.0.0'
onyx:1>
```

***condition mutex* wait –:**

Input(s):

condition: A condition object.

mutex: A mutex object that this thread currently owns.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Wait on *condition*. *mutex* is atomically released when the current thread blocks, then acquired again before the current thread runs again. Using a mutex that the current thread does not own will result in undefined behavior (likely crash).

Example(s):

```
onyx:0> condition mutex dup lock ostack
onyx:3> {dup lock exch signal unlock}
onyx:4> thread 3 1 roll
onyx:3> dup 3 1 roll
onyx:4> wait unlock join
onyx:0>
```

***pid* waitpid *status*:**

Input(s):

pid: Process identifier.

Output(s):

status: Exit code of terminated process. A negative value indicates that the process was terminated by a signal (use the **neg** operator to get the signal number), and a non-negative value is the exit code of a program that terminated normally.

Errors(s):

stackunderflow.

typecheck.

Description: Wait for the process with process ID *pid* to exit.

Example(s):

```
onyx:0> ['/bin/date'] forkexec dup 1 sprint waitpid 1 sprint
6516
Sat Jul 13 20:47:54 PDT 2002
0
onyx:0>
```

***key* where *false*:**

***key* where *dict true*:**

Input(s):

key: A key to search for in dstack.

Output(s):

dict: The topmost dictionary in dstack that contains a definition for *key*.

false/true: If false, no definition of *key* was found in dstack. Otherwise *dict* is the topmost dictionary in dstack that contains a definition for *key*.

Errors(s):

stackunderflow.

Description: Get the topmost dictionary in dstack that defines *key*.

Example(s):

```
onyx:0> $foo where pstack clear
false
onyx:0> $threaddict where pstack clear
true
<$threaddict -dict- $userdict -dict- $currenterror -dict- $errordict -dict-
$resume --stop-->
onyx:0>
```

***cond proc* while –:**

Input(s):

cond: An object that, when evaluated, places a boolean on ostack.

proc: An object to be repeatedly evaluated.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Repeatedly evaluate *cond* and *proc*, terminating the first time that *cond* places false on ostack. This operator supports the **exit** operator.

Example(s):

```
onyx:0> 0 {dup 3 lt}{inc dup 1 sprint} while pop
1
2
3
onyx:0> 0 {dup 3 lt}{inc dup 1 sprint exit} while pop
1
onyx:0>
```

***file integer/string* write *false*:**

***file integer/string* write *integer/substring true*:**

Input(s):

file: A file object.

integer: An integer that represents an ascii character value.

string: A string object.

Output(s):

false: Successful complete write.

integer: The *integer* that was passed in.

substring: The substring of *string* that was not written.

true: Successful partial write.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Write *integer* or *string* to *file*. Partial writes can only happen for non-blocking files.

Example(s):

```
onyx:0> '/tmp/foo' 'w+' open
onyx:1> dup 'Hello\n' write pop
onyx:1> dup 0 seek
onyx:1> dup readline 1 sprint 1 sprint
false
'Hello'
onyx:1>
```

***obj* xcheck *boolean*:**

Input(s):

obj: An object.

Output(s):

boolean: True if *obj* has the executable or evaluatable attribute, false otherwise.

Errors(s):

stackunderflow.

Description: Check *obj* for executable or evaluatable attribute.

Example(s):

```
onyx:0> {1 2 3} xcheck 1 sprint
true
onyx:0> [1 2 3] xcheck 1 sprint
false
onyx:0>
```

***a b* xor *r*:**

Input(s):

a: An integer or boolean.

b: The same type as *a*.

Output(s):

r: If *a* and *b* are integers, their bitwise exclusive or, otherwise their logical exclusive or.

Errors(s):

stackunderflow.

typecheck.

Description: Return the bitwise exclusive or of two integers, or the logical exclusive or of two booleans.

Example(s):

```
onyx:0> true false xor 1 sprint
true
onyx:0> true true xor 1 sprint
false
onyx:0> 5 3 xor 1 sprint
6
onyx:0>
```

– yield –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Vuluntarily yield the processor, so that another thread or process may be run.

Example(s):

```
onyx:0> 0 100000 {1 add yield} repeat 1 sprint
100000
onyx:0>
```

2.10.10 threaddict

Each thread has its own threaddict, which is not shared with any other threads. threaddict is meant to be used for thread-specific definitions that would otherwise go in systemdict.

Table 2.12: threaddict summary

Input(s) Op/Proc/Var Output(s)	Description
– **threaddict** dict	Get threaddict.
– **userdict** dict	Get userdict.
– **currenterror** dict	Get currenterror.
– **errordict** dict	Get errordict.

– currenterror *dict*:

Input(s): None.

Output(s):

dict: The currenterror dictionary. See Section 2.10.1 for details on currenterror.

Errors(s): None.

Description: Get currenterror.

Example(s):

```
onyx:0> currenterror 0 sprint
-dict-
onyx:0>
```

– errordict *dict*:

Input(s): None.

Output(s):

dict: The errordict dictionary. See Section 2.10.3 for details on errordict.

Errors(s): None.

Description: Get errordict.

Example(s):

```
onyx:0> errordict 0 sprint
-dict-
onyx:0>
```

– threaddict *dict*:

Input(s): None.

Output(s):

dict: The threaddict dictionary.

Errors(s): None.

Description: Get threaddict.

Example(s):

```
onyx:0> threaddict 0 sprint
-dict-
onyx:0>
```

– userdict *dict*:

Input(s): None.

Output(s):

dict: The userdict dictionary. See Section 2.10.12 for details on userdict.

Errors(s): None.

Description: Get userdict.

Example(s):

```
onyx:0> userdict 1 sprint
<>
onyx:0>
```

2.10.11 threadsdict

There is one entry in threadsdict for each thread. The key is the thread, and the value is null by default. The value can be safely changed, which can be useful when debugging. However, the key must not be changed, or garbage collection will trigger a horrible death, since the root set will not be complete.

2.10.12 userdict

Each thread has its own userdict, which is not shared with any other threads. userdict is meant to be used for general storage of definitions that do not need to be shared among threads. userdict starts out empty when a thread is created.

Chapter 3

The onyx program

onyx is a stand-alone Onyx interpreter, with an integrated command line editor. The Onyx language is documented in a separate chapter, so this chapter documents only the differences from the main Onyx language documentation.

3.1 Usage

onyx -h

onyx -V

onyx -e <expr>

onyx [-i <expr>]* [-f <file>]* [-s <expr>]

onyx <file> [<args>]

3.1.1 Options

-h: Display usage information and exit.

-V: Display the version number and exit.

-e <expr>: Evaluate <expr> as Onyx code.

-i <expr>: Evaluate initialization <expr>.

-f <file>: Evaluate initialization <file>.

-s <expr>: Call **start** with <expr>, rather than with the stdin file.

3.2 Environment variables

ONYX_EDITOR: By default, the command line editor uses emacs key bindings. Use this variable to explicitly set the key bindings to either “emacs” or “vi”.

ONYXRC: If this variable is set to the name of a file, that file will be evaluated as part of the initialization of interactive onyx sessions. A typical setting for this would be "~/.onyxrc".

3.3 Initialization for interactive sessions

When *onyx* is run interactively, there are several stages of initialization, some of which can be controlled directly by the user.

1. The interpreter is booted.
2. Standard I/O is set up.
3. The initial thread is created and used by all subsequent steps.
4. The initial thread's stdin is modified to support interactive command line editing.
5. Language changes specific to interactive invocation are made (see Section 3.4).
6. Initialization expressions and scripts (specified by the `-i` and `-s` flags) are evaluated in the order they are specified on the command line.
7. If the `ONYXRC` environment variable is set, then the contents of the file it specifies are evaluated.
8. stdin is evaluated.

3.4 Language differences

If *onyx* is being run interactively:

- The name "stop" is redefined in the initial thread's errordict to recursively evaluate the stdin file in a stopped context in order to keep the interpreter from exiting on error. It is possible (though generally unlikely, since the user must type a very long line of code) for buffering of stdin to cause strange things to occur; any additional program execution after an error is a result of this.
- The name "resume" is defined in the initial thread's threaddict as an alias to the stop operator. Thus, when an error occurs, when the user is ready to continue running after addressing any issues the error caused, resume can be called as a more intuitive name for resuming.
- The name "promptstring" is defined in systemdict; it takes no arguments and returns a string. The return string is used as the interactive prompt. For the duration of the call to promptstring, a temporary definition called "promptdict" is stored in the top dictionary on dstack, as a side effect of the machinery that makes the interpreter somewhat resilient to errors during the call to promptstring.

If *onyx* is being run non-interactively:

- The name "stop" in errordict is redefined to call the die operator with an argument of 1.

Chapter 4

The libonyx library

The *libonyx* library implements an embeddable Onyx interpreter. *libonyx* is designed to allow multiple interpreter instances in the same program, though since Onyx is a multi-threaded language, in most cases it makes more sense to use a single interpreter instance with multiple threads.

The Onyx language is described elsewhere in this manual, so this chapter documents the C API with as little information about the Onyx language as possible.

A minimal program that runs the Onyx interpreter interactively looks like:

```
#include <libonyx/libonyx.h>

int
main(int argc, char **argv, char **envp)
{
    cw_nx_t nx;
    cw_nxo_t thread, *nxo;

    /* Initialize libonyx and the Onyx interpreter. */
    libonyx_init();
    nx_new(&nx, NULL, argc, argv, envp);

    /* Create a thread. */
    nxo_thread_new(&thread, &nx);

    /* Set up stdin for evaluation. */
    nxo = nxo_stack_push(nxo_thread_ostack_get(&thread));
    nxo_dup(nxo, nxo_thread_stdin_get(&thread));
    nxo_attr_set(nxo, NXOA_EXECUTABLE);

    /* Start the thread. */
    nxo_thread_start(&thread);

    /* Clean up. */
    nx_delete(&nx);
    libonyx_shutdown();
```

```
    return 0;
}
```

In most cases, an application will need to implement additional Onyx operators (and make them accessible from within the Onyx interpreter) in order to make the application accessible/controllable from the Onyx interpreter. If the application user interface is to be interaction with the Onyx interpreter, then little else needs to be done.

4.1 Compilation

Use the following compiler command line to compile applications with *libonyx*.

```
cc 'onyx_config --cppflags' <file> 'onyx_config --ldflags --libs'
```

4.2 Types

libonyx is careful to use the following data types rather than the built-in types (other than when using system library functions and string pointers (char *)) to allow easy porting and explicit knowledge of variable sizes:

cw_bool_t: Boolean, either FALSE or TRUE.

cw_sint8_t: Signed 8 bit variable.

cw_uint8_t: Unsigned 8 bit variable.

cw_sint16_t: Signed 16 bit variable.

cw_uint16_t: Unsigned 16 bit variable.

cw_sint32_t: Signed 32 bit variable.

cw_uint32_t: Unsigned 32 bit variable.

cw_sint64_t: Signed 64 bit variable.

cw_uint64_t: Unsigned 64 bit variable.

cw_fp64_t: 64 bit IEEE floating point variable.

4.3 Global variables

libonyx defines the following global variables, which can be used by the application:

cw_g_mem: *mem* instance, default memory allocator.

4.4 Threads

libonyx encapsulates each interpreter instance in an *nx* object. An *nx* object supports running multiple concurrent threads. Each thread context is encapsulated by an *nxo* thread object.

In general, each process thread should execute in its own *nxo* thread object context, though the only explicit restriction placed on *nxo* thread object operations is that only one thread can be executing in an *nxo* thread object context at a time. In other words, the *nxo* thread class does not synchronize access to its internals, since there is normally no reason for multiple threads to execute in the same *nxo* thread object context.

4.5 Garbage collection

Since there can be arbitrary threads executing in the interpreter concurrently, there are two ways to implement safe garbage collection: concurrent or atomic. *libonyx* uses atomic garbage collection, which means that the thread doing garbage collection suspends all other threads that are created via *thd_new*(..., TRUE) during the mark phase. In order for this to work, the garbage collector must not do any locking while the other threads are suspended, or else there is a high probability of eventual deadlock. *libonyx* itself meets these criteria, as must any C extensions to the interpreter that are executed by the garbage collector during the mark phase (reference iteration).

4.6 Exceptions

libonyx reserves *xep* exception numbers 0 to 127 and defines the following exceptions:

***CW_ONYXX_OOM*:** Memory allocation error.

***CW_ONYXX_EXIT*:** Internal use, for the exit operator.

***CW_ONYXX_STOP*:** Internal use, for the stop operator.

***CW_ONYXX_QUIT*:** Internal use, for the quit operator.

4.7 Integration issues

4.7.1 Thread creation

libonyx's garbage collector uses the *thd* class to suspend and resume all other threads during the mark phase of atomic collection. For this to work, all threads that have any contact with *libonyx* must be created as suspendible threads using the *thd* class.

This can cause integration headaches for existing threaded applications, but there is no other portable way to suspend and resume threads. The only alternative is to assure that only one thread is executing in the interpreter and to disable timeout-based (asynchronous) collection.

4.7.2 Restarted interrupted system calls

As mentioned above, *libonyx* uses thread suspension and resumption to implement garbage collection. This has the side-effect of making restarted interrupted system calls a real possibility. However, the operating system will return with a partial result if the system call was partially complete when it was interrupted. In practice, what this means is that short reads and writes are possible where they otherwise wouldn't happen, so the application should not make any assumptions about interruptible system calls always completing with a full result. See the *thd* class documentation for more details.

4.7.3 Signals

Depending on how *libonyx* is built, *SIGUSR1* and *SIGUSR2* may be reserved by the *thd* class for thread suspension and resumption. Additionally, the *SIGPIPE* signal is ignored by default, since socket operations can cause *SIGPIPE* signals, for which the library has no use.

4.8 Guidelines for writing extensions

When embedding *libonyx* in an application, it is usually desireable to add some operators so that the interpreter can interact with the rest of the application. The *libonyx* source code contains hundreds of operators that can be used as examples when writing new operators. However, there are some very important rules that operators must follow, some of which may not be obvious when reading the code.

- Manually managed (*malloc*()/*free*()) memory should not be allocated unless the code is very careful. If a function recurses into the interpreter (this includes calls to functions such as *nxo_thread_nerror*()), there is the very real possibility that control will never return to the operator due to an exception. Code must either catch all exceptions and clean up allocations, or not recurse into the interpreter.
- Composite objects should never be allocated on the C stack. The garbage collector has no knowledge of such objects, so if the only reference to an object is on the C stack, the object may be collected, which will lead to unpredictable program behavior. Instead of allocating objects on the C stack, use tstack, available via *nxo_thread_tstack_get*(), which is a per-thread stack that the garbage collector scans.
- For an object to be safe from garbage collection, there must always be at least one reference to it inside the interpreter. So, if C code obtains a pointer to a composite object, then destroys the last known internal Onyx reference (pops it off a stack, redefines it in a dict, replaces an element of an array, etc.), the pointer is no longer safe to use. The *libonyx* API is structured such that it is invalid to do such a thing, for this reason.
- tstack must be cleaned up before returning from a function. This constraint is placed on the code in order to avoid leaking space on tstack. In debug versions of *libonyx*, this is enforced by assertions. The one exception to this rule has to do with *xep* exceptions, in which case the catchers of the exceptions are responsible for cleaning up tstack. Therefore, it is not necessary to catch exceptions merely to avoid tstack leakage.

Since Onyx type checking is dynamic, it is the responsibility of the operators to assure objects are the correct type before calling any of the type-specific *nxo_**() functions. Failure to do so will result in unpredictable behavior and likely crashes.

4.9 API

void *libonyx_init*(void):

Input(s): None.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Initialize various global variables. In particular, initialize *cw_g_mem*.

void *libonyx_shutdown*(void):

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Clean up the global variables that are initialized by *libonyx_init*().

void * *cw_opaque_alloc_t*(void *a_arg, size_t a_size, const char *a_filename, cw_uint32_t a_line_num):

Input(s):

a_arg: Opaque pointer.
a_size: Size of memory range to allocate.
a_filename: Should be __FILE__.
a_line_num: Should be __LINE__.

Output(s):

retval: Pointer to a memory range.

Exception(s):

CW_ONYXX_OOM.

Description: Allocate *a_size* of space and return a pointer to it.

void * *cw_opaque_calloc_t*(void *a_arg, size_t a_number, size_t a_size, const char *a_filename, cw_uint32_t a_line_num):

Input(s):

a_arg: Opaque pointer.
a_number: Number of elements to allocate.
a_size: Size of each element to allocate.
a_filename: Should be __FILE__.
a_line_num: Should be __LINE__.

Output(s):

retval: Pointer to a zeroed memory range.

Exception(s):

CW_ONYXX_OOM.

Description: Allocate a zeroed array of *a_number* objects, each *a_size* bytes long, and return a pointer to the array.

void * *cw_opaque_realloc_t*(void *a_arg, void *a_ptr, size_t a_size, size_t a_old_size, const char *a_filename, cw_uint32_t a_line_num):

Input(s):

a_arg: Opaque pointer.

a_ptr: Pointer to memory range to be reallocated.

a_size: Size of memory range to allocate.

a_old_size: Size of memory range previously pointed to by *a_ptr*.

a_filename: Should be __FILE__.

a_line_num: Should be __LINE__.

Output(s):

retval: Pointer to a memory range.

Exception(s):

CW_ONYXX_OOM.

Description: Reallocate *a_size* of space and return a pointer to it.

void *cw_opaque_dealloc_t*(void *a_mem, void *a_ptr, size_t a_size, const char *a_filename, cw_uint32_t a_line_num):

Input(s):

a_arg: Opaque pointer.

a_ptr: Pointer to to memory range to be freed.

a_size: Sizef of memory range pointed to by *a_ptr*.

a_filename: Should be __FILE__.

a_line_num: Should be __LINE__.

Output(s): None.

Exception(s): None.

Description: Deallocate the memory pointed to by *a_ptr*.

void * *cw_opaque_alloc*(cw_opaque_alloc_t *a_func, void *a_arg, size_t a_size):

Input(s):

a_func: Opaque allocator function pointer.

a_arg: Opaque pointer.

a_size: Size of memory range to allocate.

Output(s):

retval: Pointer to a memory range.

Exception(s):

CW_ONYXX_OOM.

Description: Allocate *a_size* of space and return a pointer to it.

void * *cw_opaque_calloc*(cw_opaque_calloc_t *a_func, void *a_arg, size_t a_number, size_t a_size):

Input(s):

a_func: Opaque allocator function pointer.

a_arg: Opaque pointer.

a_number: Number of elements to allocate.

a_size: Size of each element to allocate.

Output(s):

retval: Pointer to a zeroed memory range.

Exception(s):

CW_ONYXX_OOM.

Description: Allocate a zeroed array of *a_number* objects, each *a_size* bytes long, and return a pointer to the array.

void * *cw_opaque_realloc*(cw_opaque_realloc_t *a_func, void *a_arg, void *a_ptr, size_t a_size, size_t a_old_size):

Input(s):

a_func: Opaque allocator function pointer.

a_arg: Opaque pointer.

a_ptr: Pointer to memory range to be reallocated.

a_size: Size of memory range to allocate.

a_old_size: Size of memory range previously pointed to by *a_ptr*.

Output(s):

retval: Pointer to a memory range.

Exception(s):

CW_ONYXX_OOM.

Description: Reallocate *a_size* of space and return a pointer to it.

void *cw_opaque_dealloc*(cw_opaque_dealloc_t *a_func, void *a_mem, void *a_ptr, size_t a_size):

Input(s):

a_func: Opaque allocator function pointer.

a_arg: Opaque pointer.

a_ptr: Pointer to to memory range to be freed.

a_size: Sizef of memory range pointed to by *a_ptr*.

Output(s): None.

Exception(s): None.

Description: Deallocate the memory pointed to by *a_ptr*.

void *cw_onyx_code*(cw_nxo_t *a_thread, const char *a_code):

Input(s):

a_thread: Pointer to a thread *nxo*.

a_code: A ”-delimited string constant.

Output(s): None.

Exception(s): Depends on actions of a_code.

Description: Convenience macro for static embedded Onyx code.

void *cw_assert*(expression):

Input(s):

expression: C expression that evaluates to zero or non-zero.

Output(s): Possible error printed to *stderr*.

Exception(s): None.

Description: If the expression evaluates to zero, print an error message to *stderr* and *abort*().

Note: This macro is only active if the *CW_ASSERT* cpp macro is defined.

void *cw_dassert*(expression):

Input(s):

expression: C expression that evaluates to zero or non-zero.

Output(s): Possible error printed to *stderr*.

Exception(s): None.

Description: If the expression evaluates to zero, print an error message to *stderr* and *abort*().

Note: This macro is only active if the *CW_ASSERT* and *CW_DBG* cpp macros are defined.

void *cw_not_reached*(void):

Input(s): None.

Output(s): Error printed to *stderr*.

Exception(s): None.

Description: Abort with an error message.

Note: This macro is only active if the *CW_ASSERT* cpp macro is defined.

void *cw_check_ptr*(a_pointer):

Input(s):

a_pointer: A pointer.

Output(s): Possible error printed to *stderr*.

Exception(s): None.

Description: If *a_pointer* is NULL, print an error message to *stderr* and *abort*().

Note: This macro is only active if the *CW_ASSERT* cpp macro is defined.

void *cw_error*(const char *a_str):

Input(s):

a_str: Pointer to a NULL-terminated character array.

Output(s): Contents of *a_str*, followed by a carriage return, printed to *stderr*.

Exception(s): None.

Description: Print the contents of *a_str*, followed by a carriage return, to *stderr*.

cw_uint64_t *cw_ntohq*(cw_uint64_t a_val):

Input(s):

a_val: 64 bit integer.

Output(s):

retval: 64 bit integer.

Exception(s): None.

Description: Convert *a_val* from network byte order to host byte order and return the result.

cw_uint64_t *cw_htonq*(cw_uint64_t a_val):

Input(s):

a_val: 64 bit integer.

Output(s):

retval: 64 bit integer.

Exception(s): None.

Description: Convert *a_val* from host byte order to network byte order and return the result.

cw_uint32_t *cw_offsetof*(<type> a_type, <field_name> a_field):

Input(s):

a_type: C structure type name.

a_field: Name of a field within *a_type*.

Output(s):

retval: Offset of *a_field* into *a_type*.

Exception(s): None.

Description: Calculate the offset of *a_field* into *a_type* and return the result.

4.10 Classes

4.10.1 ch

The *ch* class implements chained hashing. It uses a simple bucket chaining hash table implementation. Table size is set at creation time, and cannot be changed, so performance will suffer if a *ch* object is over-filled. The main cw_ch_t data structure and the table are contiguously allocated, which means that care must be taken when manually pre-allocating space for the structure. Each item that is inserted into the *ch* object is encapsulated by a *chi* object, for which space can optionally be passed in as a parameter to *ch_insert*(). If no space for the *chi* object is passed in, an opaque allocator function is used internally for allocation.

Multiple entries with the same key are allowed and are stored in LIFO order.

Calling *ch_remove_iterate*() and *ch_get_iterate*() are guaranteed to operate on the oldest item in the hash table, which means that the hash code has an integrated FIFO queue.

The *ch* class is meant to be small and simple without compromising performance. Note that it is not well suited for situations where the number of items can vary wildly; the *dch* class is designed for such situations.

API

cw_uint32_t *CW_CH_TABLE2SIZEOF*(cw_uint32_t a_table_size):

Input(s):

a_table_size: Number of slots in the hash table.

Output(s):

retval: Size of a *ch* object with *a_table_size* slots.

Exception(s): None.

Description: Calculate the size of a *ch* object with *a_table_size* slots.

***ch_new*(cw_ch_t *a_ch, cw_mema_t *a_mema, cw_uint32_t a_table_size, cw_ch_hash_t *a_hash, cw_ch_key_comp_t *a_key_comp):**

Input(s):

a_ch: Pointer to space for a *ch* with *a_table_size* slots, or NULL. Use the *CW_CH_TABLE2SIZEOF*() macro to calculate the total space needed for a given table size.

a_mema: Pointer to a memory allocator.

a_table_size: Number of slots in the hash table.

a_hash: Pointer to a hashing function.

a_key_comp: Pointer to a key comparison function.

Output(s):

retval: Pointer to a *ch*.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *ch_delete*(cw_ch_t *a_ch):

Input(s):

a_ch: Pointer to a *ch*.

Output(s): None.

Exception(s): None.

Description: Destructor.

cw_uint32_t *ch_count*(cw_ch_t *a_ch):

Input(s):

a_ch: Pointer to a *ch*.

Output(s):

retval: Number of items in *a_ch*.

Exception(s): None.

Description: Return the number of items in *a_ch*.

void *ch_insert*(cw_ch_t *a_ch, const void *a_key, const void *a_data, cw_chi_t *a_chi):

Input(s):

a_ch: Pointer to a *ch*.

a_key: Pointer to a key.

a_data: Pointer to data associated with *a_key*.

a_chi: Pointer to space for a *chi*, or NULL.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Insert *a_data* into *a_ch*, using key *a_key*. Use *a_chi* for the internal *chi* container if non-NULL.

cw_bool_t *ch_remove*(cw_ch_t *a_ch, const void *a_search_key, void **r_key, void **r_data, cw_chi_t **r_chi):

Input(s):

a_ch: Pointer to a *ch*.

a_search_key: Pointer to the key to search with.

r_key: Pointer to a key pointer, or NULL.

r_data: Pointer to a data pointer, or NULL.

r_chi: Pointer to a *chi* pointer, or NULL.

Output(s):

retval:

FALSE: Success.

TRUE: Item with key *a_search_key* not found.

***r_key:** If (*r_key* != NULL) and (retval == FALSE), pointer to a key. Otherwise, undefined.

***r_data:** If (*r_data* != NULL) and (retval == FALSE), pointer to data. Otherwise, undefined.

***r_chi:** If (*r_chi* != NULL) and (retval == FALSE), pointer to space for a *chi*, or NULL. Otherwise, undefined.

Exception(s): None.

Description: Remove the item from *a_ch* that was most recently inserted with key *a_search_key*. If successful, set **r_key* and **r_data* to point to the key, data, and externally allocated *chi*, respectively.

cw_bool_t *ch_search*(cw_ch_t *a_ch, const void *a_key, void **r_data):

Input(s):

a_ch: Pointer to a *ch*.

a_key: Pointer to a key.

r_data: Pointer to a data pointer, or NULL.

Output(s):

retval:

FALSE: Success.

TRUE: Item with key *a_key* not found in *a_ch*.

***r_data:** If (*r_data* != NULL) and (retval == FALSE), pointer to data.

Exception(s): None.

Description: Search for the most recently inserted item with key *a_key*. If found, **r_data* to point to the associated data.

cw_bool_t *ch_get_iterate*(cw_ch_t *a_ch, void **r_key, void **r_data):

Input(s):

a_ch: Pointer to a *ch*.

r_key: Pointer to a key pointer, or NULL.

r_data: Pointer to a data pointer, or NULL.

Output(s):

retval:

FALSE: Success.

TRUE: *a_ch* is empty.

***r_key:** If (*r_key* != NULL) and (retval == FALSE), pointer to a key. Otherwise, undefined.

***r_data:** If (*r_data* != NULL) and (retval == FALSE), pointer to data. Otherwise, undefined.

Exception(s): None.

Description: Set **r_key* and **r_data* to point to the oldest item in *a_ch*. Promote the item so that it is the newest item in *a_ch*.

cw_bool_t *ch_remove_iterate*(cw_ch_t *a_ch, void **r_key, void **r_data, cw_chi_t **r_chi):

Input(s):

a_ch: Pointer to a *ch*.

r_key: Pointer to a key pointer, or NULL.

r_data: Pointer to a data pointer, or NULL.

r_chi: Pointer to a *chi* pointer, or NULL.

Output(s):

retval:

FALSE: Success.

TRUE: *a_ch* is empty.

***r_key:** If (*r_key* != NULL) and (retval == FALSE), pointer to a key. Otherwise, undefined.

***r_data:** If (*r_data* != NULL) and (retval == FALSE), pointer to data. Otherwise, undefined.

***r_chi:** If (*r_chi* != NULL) and (retval == FALSE), pointer to a *chi*, or NULL. Otherwise, undefined.

Exception(s): None.

Description: Set **r_key* and **r_data* to point to the oldest item in *a_ch*, set **r_chi* to point to the item's container, if externally allocated, and remove the item from *a_ch*.

cw_uint32_t *ch_string_hash*(const void *a_key):

Input(s):

a_key: Pointer to a key.

Output(s):

retval: Hash result.

Exception(s): None.

Description: NULL-terminated string hashing function.

cw_uint32_t *ch_direct_hash*(const void *a_key):

Input(s):

a_key: Pointer to a key.

Output(s):

retval: Hash result.

Exception(s): None.

Description: Direct (pointer) hashing function.

cw_bool_t *ch_string_key_comp*(const void *a_k1, const void *a_k2):

Input(s):

a_k1: Pointer to a key.

a_k2: Pointer to a key.

Output(s):

retval:

FALSE: Not equal.

TRUE: Equal.

Exception(s): None.

Description: Test two keys (NULL-terminated strings) for equality.

cw_bool_t *ch_direct_key_comp*(const void *a_k1, const void *a_k2):

Input(s):

a_k1: Pointer to a key.

a_k2: Pointer to a key.

Output(s):

retval:

FALSE: Not equal.

TRUE: Equal.

Exception(s): None.

Description: Test two keys (pointers) for equality.

4.10.2 cnd

The *cnd* class implements condition variables, which can be used in conjunction with the *mtx* class to wait for a condition to occur.

API

void *cnd_new*(cw_cnd_t *a_cnd):

Input(s):

a_cnd: Pointer to space for a *cnd*.

Output(s): None.

Exception(s): None.

Description: Constructor.

void *cnd_delete*(cw_cnd_t *a_cnd):

Input(s):

a_cnd: Pointer to a *cnd*.

Output(s): None.

Exception(s): None.

Description: Destructor.

void *cnd_signal*(cw_cnd_t *a_cnd):

Input(s):

a_cnd: Pointer to a *cnd*.

Output(s): None.

Exception(s): None.

Description: Signal one thread waiting on *a_cnd*, if there are any waiters.

void *cnd_broadcast*(cw_cnd_t *a_cnd):

Input(s):

a_cnd: Pointer to a *cnd*.

Output(s): None.

Exception(s): None.

Description: Signal all threads waiting on *a_cnd*.

cw_bool_t *cnd_timedwait*(cw_cnd_t *a_cnd, cw_mtx_t *a_mtx, const struct timespec *a_timeout):

Input(s):

a_cnd: Pointer to a *cnd*.

a_mtx: Pointer to a *mtx*.

a_timeout: Timeout, specified as an absolute time interval.

Output(s):

retval:

FALSE: Success.

TRUE: Timeout.

Exception(s): None.

Description: Wait for *a_cnd* for at least *a_time*.

void *cnd_wait*(cw_cnd_t *a_cnd, cw_mtx_t *a_mtx):

Input(s):

a_cnd: Pointer to a *cnd*.

a_mtx: Pointer to a *mtx*.

Output(s): None.

Exception(s): None.

Description: Wait for *a_cnd*.

4.10.3 dch

The *dch* class implements dynamic chained hashing. The *dch* class is a wrapper around the *ch* class that enforces fullness/emptiness constraints and rebuilds the hash table when necessary. Other than this added functionality, the *dch* class behaves almost exactly like the *ch* class. See the *ch* class documentation for additional information.

API

***dch_new*(cw_dch_t *a_dch, cw_mema_t *a_mema, cw_uint32_t a_base_table, cw_uint32_t a_base_grow, cw_uint32_t a_base_shrink, cw_ch_hash_t *a_hash, cw_ch_key_comp_t *a_key_comp):**

Input(s):

a_dch: Pointer to space for a *dch*, or NULL.

a_mema: Pointer to a memory allocator.

a_base_table: Number of slots in the initial hash table.

a_base_grow: Maximum number of items to allow in *a_dch* before doubling the hash table size. The same proportions (in relation to *a_base_table*) are used to decide when to double the table additional times.

a_base_shrink: Minimum proportional (with respect to *a_base_table*) emptiness to allow in the hash table before cutting the hash table size in half.

a_hash: Pointer to a hashing function.

a_key_comp: Pointer to a key comparison function.

Output(s):

retval: Pointer to a *dch*.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *dch_delete*(cw_dch_t *a_dch):

Input(s):

a_dch: Pointer to a *dch*.

Output(s): None.

Exception(s): None.

Description: Destructor.

cw_uint32_t *dch_count*(cw_dch_t *a_dch):

Input(s):

a_dch: Pointer to a *dch*.

Output(s):

retval: Number of items in *a_dch*.

Exception(s): None.

Description: Return the number of items in *a_dch*.

cw_bool_t *dch_shrinkable_get*(cw_dch_t *a_dch):

Input(s):

a_dch: Pointer to a *dch*.

Output(s):

retval:

TRUE: *a_dch* is currently shrinkable (initial default).

FALSE: *a_dch* is not currently shrinkable, so no attempt will be made to shrink the hash table in *dch_remove*() or *dch_remove_iterate*().

Exception(s): None.

Description: Return whether *a_dch* is currently shrinkable.

void *dch_shrinkable_set*(cw_dch_t *a_dch, cw_bool_t a_shrinkable):

Input(s):

a_dch: Pointer to a *dch*.

a_shrinkable:

TRUE: Set *a_dch* to be shrinkable.

FALSE: Set *a_dch* to not be shrinkable. No attempt will be made to shrink the hash table in *dch_remove*() or *dch_remove_iterate*() while in this state.

Output(s): None.

Exception(s): None.

Description: Set whether *a_dch* should try to shrink the hash table in *dch_remove*() and *dch_remove_iterate*().

void *dch_insert*(cw_dch_t *a_dch, const void *a_key, const void *a_data, cw_chi_t *a_chi):

Input(s):

a_dch: Pointer to a *dch*.

a_key: Pointer to a key.

a_data: Pointer to data associated with *a_key*.

a_chi: Pointer to space for a *chi*, or NULL.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Insert *a_data* into *a_dch*, using key *a_key*. Use *a_chi* for the internal *chi* container if non-NULL.

cw_bool_t *dch_remove*(cw_dch_t *a_dch, const void *a_search_key, void **r_key, void **r_data, cw_chi_t **r_chi):

Input(s):

a_dch: Pointer to a *dch*.

a_search_key: Pointer to the key to search with.

r_key: Pointer to a key pointer, or NULL.

r_data: Pointer to a data pointer, or NULL.

r_chi: Pointer to a *chi* pointer, or NULL.

Output(s):

retval:

FALSE: Success.

TRUE: Item with key *a_search_key* not found.

***r_key:** If (*r_key* != NULL) and (retval == FALSE), pointer to a key. Otherwise, undefined.

***r_data:** If (*r_data* != NULL) and (retval == FALSE), pointer to data. Otherwise, undefined.

***r_chi:** If (*r_chi* != NULL) and (retval == FALSE), pointer to space for a *chi*, or NULL. Otherwise, undefined.

Exception(s): None.

Description: Remove the item from *a_dch* that was most recently inserted with key *a_search_key*. If successful, set **r_key* and **r_data* to point to the key, data, and externally allocated *chi*, respectively.

cw_bool_t *dch_search*(cw_dch_t *a_dch, const void *a_key, void **r_data):

Input(s):

a_dch: Pointer to a *dch*.

a_key: Pointer to a key.

r_data: Pointer to a data pointer, or NULL.

Output(s):

retval:

FALSE: Success.

TRUE: Item with key *a_key* not found in *a_dch*.

***r_data:** If (*r_data* != NULL) and (retval == FALSE), pointer to data.

Exception(s): None.

Description: Search for the most recently inserted item with key *a_key*. If found, **r_data* to point to the associated data.

cw_bool_t *dch_get_iterate*(cw_dch_t *a_dch, void **r_key, void **r_data):

Input(s):

a_dch: Pointer to a *dch*.

r_key: Pointer to a key pointer, or NULL.

r_data: Pointer to a data pointer, or NULL.

Output(s):

retval:

FALSE: Success.

TRUE: *a_dch* is empty.

***r_key:** If (*r_key* != NULL) and (retval == FALSE), pointer to a key. Otherwise, undefined.

***r_data:** If (*r_data* != NULL) and (retval == FALSE), pointer to data. Otherwise, undefined.

Exception(s): None.

Description: Set **r_key* and **r_data* to point to the oldest item in *a_dch*. Promote the item so that it is the newest item in *a_dch*.

cw_bool_t *dch_remove_iterate*(cw_dch_t *a_dch, void **r_key, void **r_data, cw_chi_t **r_chi):

Input(s):

a_dch: Pointer to a *dch*.

r_key: Pointer to a key pointer, or NULL.

r_data: Pointer to a data pointer, or NULL.

r_chi: Pointer to a *chi* pointer, or NULL.

Output(s):

retval:

FALSE: Success.

TRUE: *a_dch* is empty.

***r_key:** If (*r_key* != NULL) and (retval == FALSE), pointer to a key. Otherwise, undefined.

***r_data:** If (*r_data* != NULL) and (retval == FALSE), pointer to data. Otherwise, undefined.

***r_chi:** If (*r_chi* != NULL) and (retval == FALSE), pointer to a *chi*, or NULL. Otherwise, undefined.

Exception(s): None.

Description: Set **r_key* and **r_data* to point to the oldest item in *a_dch*, set **r_chi* to point to the item's container, if externally allocated, and remove the item from *a_dch*.

4.10.4 mb

The *mb* class implements memory barriers. A memory barrier is a low level construct that is sometimes useful for guaranteeing the order in which memory operations take place, even when multiple microprocessors are involved. In most cases, mutexes are the best choice for synchronizing data access, but sometimes it is convenient (and critical to performance) to use memory barriers where weaker access constraints are adequate.

API

void *mb_write*(void):

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Create a write barrier, so that any memory writes done before the memory barrier are guaranteed to be visible by the time any memory writes after the memory barrier become visible.

4.10.5 mem

The *mem* class implements a memory allocation (malloc) wrapper. For the debug version of *libonyx*, extra information is hashed for each memory allocation that allows tracking of the following:

- File/line number of allocation.
- Double allocation/deallocation of the same address.
- Memory leaks (memory left allocated at mem destruction time).

If any memory leaks are detected, diagnostic output is printed to *stderr*.

Also, the debug version of *libonyx* sets all newly allocated bytes to `0xa5`, and all deallocated bytes to `0x5a` (except in the case of *mem_calloc*()). This tends to cause things to break sooner when uninitialized or deallocated memory is referenced.

In general, the *mem* class doesn't need to be used directly. Instead, there are several preprocessor macros that can be used: *cw_malloc*(), *cw_calloc*(), *cw_realloc*(), and *cw_free*().

The *mema* class encapsulates a set of pointers to allocation functions. It is used by the *ch* and *dch* classes.

API

cw_mema_t * *mema_new*(cw_mema_t *a_mema, cw_opaque_alloc_t *a_alloc, cw_opaque_calloc_t *a_calloc, cw_opaque_realloc_t *a_realloc, cw_opaque_dealloc_t *a_dealloc, void *a_arg):

Input(s):

a_mema: Pointer to space for a *mema*, or NULL.
a_alloc: Pointer to an allocation function.
a_alloc: Pointer to a zero-ing allocation function.
a_alloc: Pointer to a reallocation function.
a_dealloc: Pointer to a deallocation function.
a_arg: Opaque pointer to pass to *a_alloc*(), *a_calloc*(), *a_realloc*(), and *a_dealloc*().

Output(s):

retval: Pointer to a *mema*.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *mema_delete*(cw_mema_t *a_mema):

Input(s):

a_mema: Pointer to a *mema*.

Output(s): None.

Exception(s): None.

Description: Destructor.

cw_opaque_alloc_t * *mema_alloc_get*(cw_mema_t *a_mema):

Input(s):

a_mema: Pointer to a *mema*.

Output(s):

retval: Pointer to an allocation function.

Exception(s): None.

Description: Return a pointer to an allocation function.

cw_opaque_calloc_t * *mema_calloc_get*(cw_mema_t *a_mema):

Input(s):

a_mema: Pointer to a *mema*.

Output(s):

retval: Pointer to a zero-ing allocation function.

Exception(s): None.

Description: Return a pointer to a zero-ing allocation function.

cw_opaque_realloc_t * *mema_realloc_get*(cw_mema_t *a_mema):

Input(s):

a_mema: Pointer to a *mema*.

Output(s):

retval: Pointer to a reallocation function.

Exception(s): None.

Description: Return a pointer to a reallocation function.

cw_opaque_dealloc_t * *mema_dealloc_get*(cw_mema_t *a_mema):

Input(s):

a_mema: Pointer to a *mema*.

Output(s):

retval: Pointer to a deallocation function.

Exception(s): None.

Description: Return a pointer to a deallocation function.

cw_opaque_arg_t * *mema_arg_get*(cw_mema_t *a_mema):

Input(s):

a_mema: Pointer to a *mema*.

Output(s):

retval: Opaque pointer to pass to *a_alloc*(), *a_calloc*(), *a_realloc*(), and *a_dealloc*().

Exception(s): None.

Description: Return an opaque pointer to pass to the allocation functions returned by *mema_alloc_get*(*a_mema*), *mema_calloc_get*(*a_mema*), *mema_realloc_get*(*a_mema*), and *mema_dealloc_get*(*a_mema*).

cw_mem_t * *mem_new*(cw_mem_t *a_mem, cw_mem_t *a_internal):

Input(s):

a_mem: Pointer to space for a *mem*, or NULL.

a_internal: Pointer to a *mem* to use for internal memory allocation, or NULL.

Output(s):

retval: Pointer to a *mem*.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *mem_delete*(cw_mem_t *a_mem):

Input(s):

a_mem: Pointer to a *mem*.

Output(s): None.

Exception(s): None.

Description: Destructor.

void * *mem_malloc_e*(cw_mem_t *a_mem, size_t a_size, const char *a_filename, cw_uint32_t a_line_num):
void * *mem_malloc*(cw_mem_t *a_mem, size_t a_size):
void * *cw_malloc*(size_t a_size):

Input(s):

a_mem: Pointer to a *mem*.

a_size: Size of memory range to allocate.

a_filename: Should be __FILE__.

a_line_num: Should be __LINE__.

Output(s):

retval: Pointer to a memory range.

Exception(s):

CW_ONYXX_OOM.

Description: *malloc*() wrapper.

void * *mem_calloc_e*(cw_mem_t *a_mem, size_t a_number, size_t a_size, const char *a_filename, cw_uint32_t a_line_num):
void * *mem_calloc*(cw_mem_t *a_mem, size_t a_number, size_t a_size):
void * *cw_calloc*(size_t a_number, size_t a_size):

Input(s):

a_mem: Pointer to a *mem*.

a_number: Number of elements to allocate.

a_size: Size of each element to allocate.

a_filename: Should be __FILE__.

a_line_num: Should be __LINE__.

Output(s):

retval: Pointer to a zeroed memory range.

Exception(s):

CW_ONYXX_OOM.

Description: *calloc*() wrapper.

void * *mem_realloc_e*(cw_mem_t *a_mem, void *a_ptr, size_t a_size, size_t a_old_size, const char *a_filename, cw_uint32_t a_line_num):
void * *mem_realloc*(cw_mem_t *a_mem, void *a_ptr, size_t a_size):
void * *cw_realloc*(void *a_ptr, size_t a_size):

Input(s):

a_mem: Pointer to a *mem*.

a_ptr: Pointer to memory range to be reallocated.

a_size: Size of memory range to allocate.

a_old_size: Size of memory range previously pointed to by *a_ptr*.

a_filename: Should be __FILE__.

a_line_num: Should be __LINE__.

Output(s):

retval: Pointer to a memory range.

Exception(s):

CW_ONYXX_OOM.

Description: *realloc*() wrapper.

void *mem_free_e*(cw_mem_t *a_mem, void *a_ptr, size_t a_size, const char *a_filename, cw_uint32_t a_line_num):
void *mem_free*(cw_mem_t *a_mem, void *a_ptr, size_t a_size):
void *cw_free*(void *a_ptr):

Input(s):

a_mem: Pointer to a *mem*.

a_ptr: Pointer to to memory range to be freed.

a_size: Sizef of memory range pointed to by *a_ptr*.

a_filename: Should be __FILE__.

a_line_num: Should be __LINE__.

Output(s): None.

Exception(s): None.

Description: *free*() wrapper.

4.10.6 mq

The *mq* class implements a simple unidirectional message queue. In addition to putting and getting messages, there are methods that control the ability to get or put. This provides a simple out of band state transition capability.

API

void *mq_new*(cw_mq_t *a_mq, cw_mem_t *a_mem, cw_uint32_t a_msg_size):

Input(s):

a_mq: Pointer to space for a *mq*.

a_mem: Pointer to the allocator to use internally.

a_msg_size: Size (in bytes) of messages used for all subsequent calls to *mq_*get*() and *mq_put*().

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *mq_delete*(cw_mq_t *a_mq):

Input(s):

a_mq: Pointer to a *mq*.

Output(s): None.

Exception(s): None.

Description: Destructor.

cw_bool_t *mq_tryget*(cw_mq_t *a_mq, ...):

Input(s):

a_mq: Pointer to a *mq*.

...: Pointer to space to store a message.

Output(s):

retval:

FALSE: Success.

TRUE: No messages in the queue, or get is in the stop state.

***...:** If *retval* is FALSE, a message. Otherwise, undefined.

Exception(s): None.

Description: Try to get a message, but return TRUE if none are available.

cw_bool_t *mq_timedget*(cw_mq_t *a_mq, const struct timespec *a_timeout, ...):

Input(s):

a_mq: Pointer to a *mq*.

a_timeout: Timeout, specified as an absolute time interval.

...: Pointer to space to store a message.

Output(s):

retval:

FALSE: Success.

TRUE: No messages in the queue, or get is in the stop state.

***...:** If *retval* is FALSE, a message. Otherwise, undefined.

Exception(s): None.

Description: Get a message. If none are available, block until a message is available, or until timeout.

cw_bol_t *mq_get*(cw_mq_t *a_mq, ...):

Input(s):

a_mq: Pointer to a *mq*.

...: Pointer to space to store a message.

Output(s):

retval:

FALSE: Success.

TRUE: Get is in the stop state.

***...:** If *retval* is FALSE, a message. Otherwise, undefined.

Exception(s): None.

Description: Get a message. If none are available, block until a message is available.

cw_bool_t *mq_put*(cw_mq_t *a_mq, ...):

Input(s):

a_mq: Pointer to a *mq*.

...: A message.

Output(s):

retval:

FALSE: Success.

TRUE: Failure due to put being in the stop state.

Exception(s):

CW_ONYXX_OOM.

Description: Put a message in *a_mq*.

cw_bool_t *mq_get_start*(cw_mq_t *a_mq):

Input(s):

a_mq: Pointer to a *mq*.

Output(s):

retval:

FALSE: Success.

TRUE: Error (already in start state).

Exception(s): None.

Description: Change the get operation to the start state (*mq_get*() will not return TRUE).

cw_bool_t *mq_get_stop*(cw_mq_t *a_mq):

Input(s):

a_mq: Pointer to a *mq*.

Output(s):

retval:

FALSE: Success.

TRUE: Error (already in stop state).

Exception(s): None.

Description: Change the get operation to the stop state (*mq_get*() will return TRUE).

cw_bool_t *mq_put_start*(cw_mq_t *a_mq):

Input(s):

a_mq: Pointer to a *mq*.

Output(s):

retval:

FALSE: Success.

TRUE: Error (already in start state).

Exception(s): None.

Description: Change the put operation to the start state (*mq_put*() will not return TRUE).

cw_bool_t *mq_put_stop*(cw_mq_t *a_mq):

Input(s):

a_mq: Pointer to a *mq*.

Output(s):

retval:

FALSE: Success.
TRUE: Error (already in stop state).

Exception(s): None.

Description: Change the put operation to the stop state (*mq_put*() will return TRUE).

4.10.7 mtx

The *mtx* class implements typical mutual exclusion locks. Only one thread can hold a lock at a time, and attempting to attain the lock while already owning it has undefined results.

API

void *mtx_new*(cw_mtx_t *a_mtx):

Input(s):
a_mtx: Pointer to space for a *mtx*.

Output(s): None.

Exception(s): None.

Description: Constructor.

void *mtx_delete*(cw_mtx_t *a_mtx):

Input(s):
a_mtx: Pointer to a *mtx*.

Output(s): None.

Exception(s): None.

Description: Destructor.

void *mtx_lock*(cw_mtx_t *a_mtx):

Input(s):
a_mtx: Pointer to a *mtx*.

Output(s): None.

Exception(s): None.

Description: Lock *a_mtx*.

cw_bool_t *mtx_trylock*(cw_mtx_t *a_mtx):

Input(s):
a_mtx: Pointer to a *mtx*.

Output(s):
retval:
FALSE: Success.
TRUE: Failure.

Exception(s): None.

Description: Try to lock *a_mtx*, but return immediately instead of blocking if *a_mtx* is already locked.

void *mtx_unlock*(cw_mtx_t *a_mtx):

Input(s):

a_mtx: Pointer to a *mtx*.

Output(s): None.

Exception(s): None.

Description: Unlock *a_mtx*.

4.10.8 nx

The *nx* class encapsulates an Onyx interpreter instance. It contains a number of interpreter-global objects, as well as the garbage collector. Reclamation of all objects associated with an *nx* instance is managed by a garbage collector, so when an *nx* is destroyed, all associated objects are deallocated.

API

cw_nx_t * *nx_new*(cw_nx_t *a_nx, cw_op_t *a_thread_init, int a_argc, char **a_argv, char **a_envp):

Input(s):

a_nx: Pointer to space for an *nx*, or NULL.

a_thread_init: Pointer to an initialization function to be called during thread initialization, or NULL.

a_argc: Number of command line arguments.

a_argv: Pointer to an array of command line argument strings.

a_envp: Pointer to an array of environment variable strings.

Output(s):

retval: Pointer to an *nx*.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *nx_delete*(cw_nx_t *a_nx):

Input(s): Pointer to an *nx*.

Output(s): None.

Exception(s): None.

Description: Destructor.

cw_nxa_t * *nx_nxa_get*(cw_nx_t *a_nx):

Input(s): Pointer to an *nx*.

Output(s):

retval: Pointer to an *nxa*.

Exception(s): None.

Description: Return a pointer to the garbage collector.

cw_mema_t * *nx_mema_get*(cw_nx_t *a_nx):

Input(s): Pointer to an *nx*.

Output(s):

retval: Pointer to an *mema*.

Exception(s): None.

Description: Return a pointer to a memory allocator that encapsulates the garbage collector allocator functions.

cw_nxo_t * *nx_systemdict_get*(cw_nx_t *a_nx):

Input(s):

a_nx: Pointer to an *nx*.

Output(s):

retval: Pointer to the *nxo* corresponding to *systemdict* .

Exception(s): None.

Description: Return a pointer to the *nxo* corresponding to *systemdict* .

cw_nxo_t * *nx_globaldict_get*(cw_nx_t *a_nx):

Input(s):

a_nx: Pointer to an *nx*.

Output(s):

retval: Pointer to the *nxo* corresponding to *globaldict* .

Exception(s): None.

Description: Return a pointer to the *nxo* corresponding to *globaldict* .

cw_nxo_t * *nx_envdict_get*(cw_nx_t *a_nx):

Input(s):

a_nx: Pointer to an *nx*.

Output(s):

retval: Pointer to the *nxo* corresponding to *envdict* .

Exception(s): None.

Description: Return a pointer to the *nxo* corresponding to *envdict* .

cw_nxo_t * *nx_stdin_get*(cw_nx_t *a_nx):

Input(s):

a_nx: Pointer to an *nx*.

Output(s):

retval: Pointer to the *nxo* corresponding to *stdin* .

Exception(s): None.

Description: Return a pointer to the *nxo* corresponding to *stdin* .

void *nx_stdin_set*(cw_nx_t *a_nx, cw_nxo_t *a_stdin):

Input(s):

a_nx: Pointer to an *nx*.

a_stdin: Pointer to a file *nxo*.

Output(s): None.

Exception(s): None.

Description: Set *a_nx*'s stdin to *a_stdin*.

cw_nxo_t * *nx_stdout_get*(cw_nx_t *a_nx):

Input(s):

a_nx: Pointer to an *nx*.

Output(s):

retval: Pointer to the *nxo* corresponding to *stdout* .

Exception(s): None.

Description: Return a pointer to the *nxo* corresponding to *stdout* .

void *nx_stdout_set*(cw_nx_t *a_nx, cw_nxo_t *a_stdout):

Input(s):

a_nx: Pointer to an *nx*.

a_stdout: Pointer to a file *nxo*.

Output(s): None.

Exception(s): None.

Description: Set *a_nx*'s stdout to *a_stdout*.

cw_nxo_t * *nx_stderr_get*(cw_nx_t *a_nx):

Input(s):

a_nx: Pointer to an *nx*.

Output(s):

retval: Pointer to the *nxo* corresponding to *stderr* .

Exception(s): None.

Description: Return a pointer to the *nxo* corresponding to *stderr* .

void *nx_stderr_set*(cw_nx_t *a_nx, cw_nxo_t *a_stderr):

Input(s):

a_nx: Pointer to an *nx*.

a_stderr: Pointer to a file *nxo*.

Output(s): None.

Exception(s): None.

Description: Set *a_nx*'s stderr to *a_stderr*.

4.10.9 nxa

The *nxa* class implements garbage collection. The garbage collector runs a separate thread that is controlled via an asynchronous message queue.

API

void * *nxa_malloc_e*(cw_nxa_t *a_nxa, size_t a_size, const char *a_filename, cw_uint32_t a_line_num):
void * *nxa_malloc*(cw_nxa_t *a_nxa, size_t a_size):

Input(s):

a_nxa: Pointer to a *nxa*.

a_size: Size of memory range to allocate.

a_filename: Should be __FILE__.

a_line_num: Should be __LINE__.

Output(s):

retval: Pointer to a memory range.

Exception(s):

CW_ONYXX_OOM.

Description: *malloc*() wrapper.

void * *nxa_calloc_e*(cw_nxa_t *a_nxa, size_t a_number, size_t a_size, const char *a_filename, cw_uint32_t a_line_num):
void * *nxa_calloc*(cw_nxa_t *a_nxa, size_t a_number, size_t a_size):

Input(s):

a_nxa: Pointer to a *nxa*.

a_number: Number of elements to allocate.

a_size: Size of each element to allocate.

a_filename: Should be __FILE__.

a_line_num: Should be __LINE__.

Output(s):

retval: Pointer to a zeroed memory range.

Exception(s):

CW_ONYXX_OOM.

Description: *calloc*() wrapper.

void * *nxa_realloc_e*(cw_nxa_t *a_nxa, void *a_ptr, size_t a_size, size_t a_old_size, const char *a_filename, cw_uint32_t a_line_num):
void * *nxa_realloc*(cw_nxa_t *a_nxa, void *a_ptr, size_t a_size, size_t a_old_size):

Input(s):

a_nxa: Pointer to a *nxa*.

a_ptr: Pointer to memory range to be reallocated.

a_size: Size of memory range to allocate.

a_old_size: Size of memory range previously pointed to by *a_ptr*.

a_filename: Should be __FILE__.

a_line_num: Should be __LINE__.

Output(s):

retval: Pointer to a memory range.

Exception(s):

CW_ONYXX_OOM.

Description: *realloc*() wrapper.

void * *nxa_free_e*(cw_nxa_t *a_nxa, void *a_ptr, size_t a_size, const char *a_filename, cw_uint32_t a_line_num):
void * *nxa_free*(cw_nxa_t *a_nxa, void *a_ptr, size_t a_size):

Input(s):

a_nxa: Pointer to a *nxa*.

a_ptr: Pointer to to memory range to be freed.

a_size: Sizef of memory range pointed to by *a_ptr*.

a_filename: Should be __FILE__.

a_line_num: Should be __LINE__.

Output(s): None.

Exception(s): None.

Description: *free*() wrapper.

void *nxa_collect*(cw_nxa_t *a_nxa):

Input(s):

a_nxa: Pointer to a *nxa*.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Force an asynchronous garbage collection.

void *nxa_dump*(cw_nxa_t *a_nxa, cw_nxo_t *a_thread):

Input(s):

a_nxa: Pointer to a *nxa*.

a_thread: Pointer to a thread *nxo*.

Output(s): Output printed to *stdout* .

Exception(s):

CW_ONYXX_OOM.

Description: Print the internal state of gcdict to *stdout* .

cw_bool_t *nxa_active_get*(cw_nxa_t *a_nxa):

Input(s):

a_nxa: Pointer to a *nxa*.

Output(s):

retval:

FALSE: Garbage collector deactivated.

TRUE: Garbage collector active.

Exception(s): None.

Description: Return whether the garbage collector is active (runnable).

void *nxa_active_set*(cw_nxa_t *a_nxa, cw_bool_t a_active):

Input(s):

a_nxa: Pointer to a *nxa*.

a_active:

FALSE: Deactivate garbage collector.

TRUE: Activate garbage collector.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Send a message to the garbage collector to activate or deactivate. The asynchronous nature of the message means that it is possible for the garbage collector to run after this function returns, even if a deactivation message has been sent.

cw_nxoi_t *nxa_period_get*(cw_nxa_t *a_nxa):

Input(s):

a_nxa: Pointer to a *nxa*.

Output(s):

retval: Current inactivity period in seconds that the garbage collector waits before doing a collection.

Exception(s): None.

Description: Return the current inactivity period in seconds that the garbage collector waits before doing a collection.

void *nxa_period_set*(cw_nxa_t *a_nxa, cw_nxoi_t a_period):

Input(s):

a_nxa: Pointer to a *nxa*.

a_period: Inactivity period in seconds that the garbage collector should wait before doing a collection. If 0, the garbage collector will never run due to inactivity.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Set the inactivity period in seconds that the garbage collector should wait before doing a collection.

cw_nxoi_t *nxa_threshold_get*(cw_nxa_t *a_nxa):

Input(s):

a_nxa: Pointer to a *nxa*.

Output(s):

retval: Number of bytes of memory allocated since the last garbage collection that will trigger the garbage collector to run.

Exception(s): None.

Description: Return the number of bytes of memory allocated since the last garbage collection that will trigger the garbage collector to run.

void *nxa_threshold_set*(cw_nxa_t *a_nxa, cw_nxoi_t a_threshold):

Input(s):

a_nxa: Pointer to a *nxa*.

a_threshold: The number of bytes of memory allocated since the last garbage collection that will trigger the garbage collector to run.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Set the number of bytes of memory allocated since the last garbage collection that will trigger the garbage collector to run.

void *nxa_stats_get*(cw_nxa_t *a_nxa, cw_nxoi_t *r_collections, cw_nxoi_t *r_count, cw_nxoi_t *r_ccount, cw_nxoi_t *r_cmark, cw_nxoi_t *r_csweep, cw_nxoi_t *r_mcount, cw_nxoi_t *r_mmark, cw_nxoi_t *r_msweep, cw_nxoi_t *r_scount, cw_nxoi_t *r_smark, cw_nxoi_t *r_ssweep):

Input(s):

a_nxa: Pointer to a *nxa*.

r_collections: Pointer to an integer.

r_count: Pointer to an integer.

r_ccount: Pointer to an integer.

r_cmark: Pointer to an integer.

r_csweep: Pointer to an integer.

r_mcount: Pointer to an integer.

r_mmark: Pointer to an integer.

r_msweep: Pointer to an integer.

r_scount: Pointer to an integer.

r_smark: Pointer to an integer.

r_ssweep: Pointer to an integer.

Output(s):

***r_collections:** Number of times the garbage collector has run.

***r_count:** Current number of bytes of memory allocated.

***r_ccount:** Number of bytes of memory allocated as of the end of the most recent garbage collection.

***r_cmark:** Number of microseconds spent in the mark phase of the most recent garbage collection.

***r_csweep:** Number of microseconts spent in the sweep phase of the most recent garbage collection.

***r_mcount:** Largest number of bytes of memory ever allocated at any point in time.

***r_mmark:** Largest number of microseconds ever spent in the mark phase of a garbage collection.

***r_msweep:** Largest number of microseconts spent in the sweep phase of a garbage collection.

***r_scount:** Total number of bytes of memory ever allocated.

***r_smark:** Total number of microseconds spent in the mark phase of all garbage collections.

***r_ssweep:** Total number of microseconts spent in the sweep phase of all garbage collections.

Exception(s): None.

Description: Return garbage collector statistics.

cw_nx_t * *nxa_nx_get*(cw_nxa_t *a_nxa):

Input(s):

a_nxa: Pointer to a *nxa*.

Output(s):

retval: Pointer to a *nx*.

Exception(s): None.

Description: Return a pointer to the *nx* associated with *a_nxa*.

cw_nxo_t * *nxa_gcdict_get*(cw_nxa_t *a_nxa):

Input(s):

a_nxa: Pointer to a *nxa*.

Output(s):

retval: Pointer to a dict *nxo*.

Exception(s): None.

Description: Return a pointer to the dict *nxo* corresponding to *gcdict* .

4.10.10 nxn

The *nxn* class provides access to a table of string constants. The main reason for this class's existence is that multiple C files often use identical string constants, and this saves memory by allowing all to refer to a single string.

API

const cw_uint8_t * *nxn_str*(cw_nxn_t a_nxn):

Input(s):

a_nxn: A number that corresponds to an entry in the string table.

Output(s):

retval: Pointer to a string constant.

Exception(s): None.

Description: Return a pointer to the string constant associated with *a_nxn*.

cw_uint32_t *nxn_len*(cw_nxn_t a_nxn):

Input(s):

a_nxn: A number that corresponds to an entry in the string table.

Output(s):

retval: String length of a string constant.

Exception(s): None.

Description: Return the string length of the string constant associated with *a_nxn*.

4.10.11 nxo

The *nxo* class is the basis for the Onyx type system. *nxo* objects can be any of the following types, as determined by the cw_nxot_t type:

NXOT_NO: *nxo_no*

NXOT_ARRAY: *nxo_array*

NXOT_BOOLEAN: *nxo_boolean*

NXOT_CONDITION: *nxo_condition*

NXOT_DICT: *nxo_dict*

NXOT_FILE: *nxo_file*

NXOT_FINO: *nxo_fino*

NXOT_HOOK: *nxo_hook*

NXOT_INTEGER: *nxo_integer*

NXOT_MARK: *nxo_mark*

NXOT_MUTEX: *nxo_mutex*

NXOT_NAME: *nxo_name*

NXOT_NULL: *nxo_null*

NXOT_OPERATOR: *nxo_operator*

NXOT_PMARK: *nxo_pmark*

NXOT_REAL: *nxo_real*

NXOT_STACK: *nxo_stack*

NXOT_STRING: *nxo_string*

NXOT_THREAD: *nxo_thread*

Due to limitations of the C programming language, it is the responsibility of the application to do type checking to assure that an incompatible *nxo* object is not passed to a type-specific function. For example, passing a file *nxo* to *nxo_string_get*() is prohibited, and will result in undefined behaviour (including crashes).

Composite objects contain a reference to an *nxoe* object. For the most part, the application does not need to be aware of this. The only exception is when writing extensions with the hook type. Hook objects need to be able to iterate over the objects they reference internally, and return *nxoe* references to the garbage collector.

The following functions are applicable to all types of *nxo* objects.

API

cw_sint32_t *nxo_compare*(const cw_nxo_t *a_a, const cw_nxo_t *a_b):

Input(s):

a_a: Pointer to an *nxo*.

a_b: Pointer to an *nxo*.

Output(s):

retval:

-1: For types which it is meaningful (integer, string), *a_a* is less than *a_b*.

0: *a_a* and *a_b* are equal.

1: For types which it is meaningful (integer, string), *a_a* is greater than *a_b*.

2: Incompatible types, or not the same composite object.

Exception(s): None.

Description: Compare *a_a* and *a_b*.

void *nxo_dup*(cw_nxo_t *a_to, cw_nxo_t *a_from):

Input(s):

a_to: Pointer to an *nxo*.

a_from: Pointer to an *nxo*.

Output(s): None.

Exception(s): None.

Description: Duplicate *a_from* to *a_to*. This does not do a copy of composite objects; rather it creates a new reference to the value of a composite object.

cw_nxot_t *nxo_type_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to an *nxo*.

Output(s):

retval:

NXOT_NO: *nxo_no*

NXOT_ARRAY: *nxo_array*

NXOT_BOOLEAN: *nxo_boolean*

NXOT_CONDITION: *nxo_condition*

NXOT_DICT: *nxo_dict*

NXOT_FILE: *nxo_file*

NXOT_FINO: *nxo_fino*

NXOT_HOOK: *nxo_hook*

NXOT_INTEGER: *nxo_integer*

NXOT_MARK: *nxo_mark*

NXOT_MUTEX: *nxo_mutex*

NXOT_NAME: *nxo_name*

NXOT_NULL: *nxo_null*

NXOT_OPERATOR: *nxo_operator*

NXOT_PMARK: *nxo_pmark*

NXOT_REAL: *nxo_real*
NXOT_STACK: *nxo_stack*
NXOT_STRING: *nxo_string*
NXOT_THREAD: *nxo_thread*

Exception(s): None.

Description: Return the type of *a_nxo*.

cw_nxoe_t * *nxo_nxoe_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to an *nxo*.

Output(s):

retval: Pointer to the *nxoe* associated with *a_nxo*, or NULL if *a_nxo* is not composite.

Exception(s): None.

Description: Return a pointer to the *nxoe* associated with *a_nxo*.

cw_bool_t *nxo_lcheck*():

Input(s):

a_nxo: Pointer to an array, dict, file, stack, or string *nxo*.

Output(s):

retval:

FALSE: *a_nxo* is not implicitly locked.
TRUE: *a_nxo* is implicitly locked.

Exception(s): None.

Description: For array, dict, file, stack, or string *nxo*s, return whether *a_nxo* is implicitly locked.

cw_nxoa_t *nxo_attr_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to an *nxo*.

Output(s):

retval:

NXOA_LITERAL: *a_nxo* is literal.
NXOA_EXECUTABLE: *a_nxo* is executable.

Exception(s): None.

Description: Return the attribute for *a_nxo*.

void *nxo_attr_set*(cw_nxo_t *a_nxo, cw_nxoa_t a_attr):

Input(s):

a_nxo: Pointer to an *nxo*.
a_attr: Value of attribute to set for *a_nxo*.

Output(s): None.

Exception(s): None.

Description: Set the attribute for *a_nxo* to *a_attr*.

4.10.12 nxo_array

The *nxo_array* class is a subclass of the *nxo* class.

API

void *nxo_array_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, cw_bool_t a_locking, cw_uint32_t a_len):

Input(s):

- **a_nxo:** Pointer to an array *nxo*.
- **a_nx:** Pointer to an *nx*.
- **a_locking:** Implicit locking mode.
- **a_len:** Number of array elements.

Output(s): None.

Exception(s):

- **CW_ONYXX_OOM.**

Description: Constructor.

void *nxo_array_subarray_new*(cw_nxo_t *a_nxo, cw_nxo_t *a_array, cw_nx_t *a_nx, cw_uint32_t a_offset, cw_uint32_t a_len):

Input(s):

- **a_nxo:** Pointer to an array *nxo*.
- **a_array:** Pointer to an array *nxo* to create a subarray of.
- **a_nx:** Pointer to an *nx*.
- **a_offset:** Offset into *a_array*.
- **a_len:** Number of array elements.

Output(s): None.

Exception(s):

- **CW_ONYXX_OOM.**

Description: Subarray constructor.

void *nxo_array_copy*(cw_nxo_t *a_to, cw_nxo_t *a_from):

Input(s):

- **a_to:** Pointer to an array *nxo*.
- **a_from:** Pointer to an array *nxo*.

Output(s): None.

Exception(s): None.

Description: Copy the contents of *a_from* to *a_to*. The length of *a_to* must be at least that of *a_from*.

cw_uint32_t *nxo_array_len_get*(const cw_nxo_t *a_nxo):

Input(s):

- **a_nxo:** Pointer to an array *nxo*.

Output(s):

retval: Number of elements in *a_nxo*.

Exception(s): None.

Description: Return the number of elements in *a_nxo*.

void *nxo_array_el_get*(const cw_nxo_t *a_nxo, cw_nxoi_t a_offset, cw_nxo_t *r_el):

Input(s):

a_nxo: Pointer to an array *nxo*.

a_offset: Offset of element to get.

r_el: Pointer to space to dup an object to.

Output(s):

***r_el:** A dup of the element of *a_nxo* at offset *a_offset*.

Exception(s): None.

Description: Get a dup of the element of *a_nxo* at offset *a_offset*.

void *nxo_array_el_set*(cw_nxo_t *a_nxo, cw_nxo_t *a_el, cw_nxoi_t a_offset):

Input(s):

a_nxo: Pointer to an array *nxo*.

a_el: Pointer to an *nxo*.

a_offset: Offset of element in *a_nxo* to replace with *a_el*.

Output(s): None.

Exception(s): None.

Description: Dup *a_el* into the element of *a_nxo* at offset *a_offset*.

4.10.13 nxo_boolean

The *nxo_boolean* class is a subclass of the *nxo* class.

API

void *nxo_boolean_new*(cw_nxo_t *a_nxo, cw_bool_t a_val):

Input(s):

a_nxo: Pointer to a boolean *nxo*.

a_val: Initial value.

Output(s): None.

Exception(s): None.

Description: Constructor.

cw_bool_t *nxo_boolean_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a boolean *nxo*.

Output(s):

retval: Value of *a_nxo*.

Exception(s): None.

Description: Return the value of *a_nxo*.

void *nxo_boolean_set*(cw_nxo_t *a_nxo, cw_bool_t a_val):

Input(s):

a_nxo: Pointer to a boolean *nxo*.

a_val: Value to set *a_nxo* to.

Output(s): None.

Exception(s): None.

Description: Set the value of *a_nxo* to *a_val*.

4.10.14 nxo_condition

The *nxo_condition* class is a subclass of the *nxo* class.

API

void *nxo_condition_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx):

Input(s):

a_nxo: Pointer to a condition *nxo*.

a_nx: Pointer to an *nx*.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *nxo_condition_signal*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a condition *nxo*.

Output(s): None.

Exception(s): None.

Description: Signal one thread waiting on *a_nxo*, if there are any waiters.

void *nxo_condition_broadcast*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a condition *nxo*.

Output(s): None.

Exception(s): None.

Description: Signal all threads waiting on *a_nxo*.

void *nxo_condition_wait*(cw_nxo_t *a_nxo, cw_nxo_t *a_mutex):

Input(s):

a_nxo: Pointer to a condition *nxo*.

a_mutex: Pointer to a mutex *nxo*.

Output(s): None.

Exception(s): None.

Description: Wait for *a_nxo*.

cw_bool_t *nxo_condition_timedwait*(cw_nxo_t *a_nxo, cw_nxo_t *a_mutex, const struct timespec *a_timeout):

Input(s):

a_nxo: Pointer to a condition *nxo*.

a_mutex: Pointer to a mutex *nxo*.

a_timeout: Timeout, specified as an absolute time interval.

Output(s):

retval:

FALSE: Success.

TRUE: Timeout.

Exception(s): None.

Description: Wait for *a_nxo* for at least *a_timeout*.

4.10.15 nxo_dict

The *nxo_dict* class is a subclass of the *nxo* class.

API

void *nxo_dict_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, cw_bool_t a_locking, cw_uint32_t a_dict_size):

Input(s):

a_nxo: Pointer to a dict *nxo*.

a_nx: Pointer to an *nx*.

a_locking: Implicit locking mode.

a_dict_size: Initial number of slots. Dictionaries dynamically grow and shrink as needed, but if the maximum size of *a_nxo* is known, it should be specified here to save space.

Output(s): None

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

***nxo_dict_copy*(cw_nxo_t *a_to, cw_nxo_t *a_from, cw_nx_t *a_nx):**

Input(s):

a_to: Pointer to a dict *nxo*.

a_from: Pointer to a dict *nxo*.

a_nx: Pointer to an *nx*.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Do a deep copy (actual contents are copied) of *a_from* to *a_to*.

void *nxo_dict_def*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, cw_nxo_t *a_key, cw_nxo_t *a_val):

Input(s):

a_nxo: Pointer to a dict *nxo*.

a_nx: Pointer to an *nx*.

a_key: Pointer to an *nxo*.

a_val: Pointer to an *nxo*.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Define *a_key* with value *a_val* in *a_nxo*.

void *nxo_dict_undef*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, cw_nxo_t *a_key):

Input(s):

a_nxo: Pointer to a dict *nxo*.

a_nx: Pointer to an *nx*.

a_key: Pointer to an *nxo*.

Output(s): None.

Exception(s): None.

Description: Undefine *a_key* in *a_nxo*, if defined.

cw_bool_t *nxo_dict_lookup*(const cw_nxo_t *a_nxo, const cw_nxo_t *a_key, cw_nxo_t *r_nxo):

Input(s):

a_nxo: Pointer to a dict *nxo*.

a_key: Pointer to an *nxo*.

r_nxo: Pointer to an *nxo*.

Output(s):

retval:

FALSE: Success.

TRUE: *a_key* not found.

r_nxo: If *retval* is FALSE, value associated with *a_key* in *a_nxo*, otherwise unmodified.

Exception(s): None.

Description: Find *a_key* in *a_nxo* and dup its associated value to *r_nxo*.

cw_uint32_t *nxo_dict_count*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a dict *nxo*.

Output(s):

retval: The number of key/value pairs in *a_nxo*.

Exception(s): None.

Description: Return the number of key/value pairs in *a_nxo*.

void *nxo_dict_iterate*(cw_nxo_t *a_nxo, cw_nxo_t *r_nxo):

Input(s):

a_nxo: Pointer to a dict *nxo*.

r_nxo: Pointer to an *nxo*.

Output(s):

FALSE: Success.

TRUE: *a_nxo* is empty.

r_nxo: If *retval* is FALSE, A key in *a_nxo*, otherwise unmodified.

Exception(s): None.

Description: Iteratively get a key in *a_nxo*. Each successive call to this function will get the next key, and wrap back around to the first key when all keys have been returned.

4.10.16 nxo_file

The *nxo_file* class is a subclass of the *nxo* class.

API

cw_sint32_t *cw_nxo_file_read_t*(void *a_arg, cw_nxo_t *a_file, cw_uint32_t a_len, cw_uint8_t *r_str):

Input(s):

a_arg: Opaque data pointer.

a_file: Pointer to a file *nxo*.

a_len: Length of *r_str*.

r_str: Pointer to space to put read data.

Output(s):

retval:

-1: Read error.

>= 0: Number of bytes stored in *r_str*.

r_str: If *retval* is non-negative, *retval* bytes of read data, otherwise undefined.

Exception(s): Application specific.

Description: Read up to *a_len* bytes of data from *a_file* and store the result in *r_str*.

cw_bool_t *cw_nxo_file_write_t*(void *a_arg, cw_nxo_t *a_file, const cw_uint8_t *a_str, cw_uint32_t a_len):

Input(s):

a_arg: Opaque data pointer.

a_file: Pointer to a file *nxo*.

a_str: Pointer to data to write.

a_len: Length of *a_str*.

Output(s):

retval:

FALSE: Success.

TRUE: Write error.

Exception(s): Application specific.

Description: Write *a_len* bytes of data from *a_str* to *a_file*.

cw_nxoe_t * *cw_nxo_file_ref_iter_t*(void *a_arg, cw_bool_t a_reset):

Input(s):

a_arg: Opaque data pointer.

a_reset:

FALSE: At least one iteration has already occurred.

TRUE: First iteration.

Output(s):

retval:

non-NULL: Pointer to an *nxoe*.

NULL: No more references.

Exception(s): None.

Description: Reference iterator function typedef.

void *cw_nxo_file_delete_t*(void *a_arg, cw_nx_t *a_nx):

Input(s):

a_arg: Opaque data pointer.

a_nx: Pointer to an *nx*.

Output(s): None.

Exception(s): None.

Description: Destructor function typedef.

void *nxo_file_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, cw_bool_t a_locking):

Input(s):

a_nxo: Pointer to a file *nxo*.

a_nx: Pointer to an *nx*.

a_locking: Implicit locking mode.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *nxo_file_fd_wrap*(cw_nxo_t *a_nxo, cw_uint32_t a_fd, cw_bool_t a_close):

Input(s):

a_nxo: Pointer to a file *nxo*.

a_fd: File descriptor number.

a_close: If TRUE, close *a_fd* when *a_nxo* is finalized by the garbage collector, otherwise, do not automatically close *a_fd*. This should typically be set to TRUE.

Output(s): None.

Exception(s): None.

Description: Wrap file descriptor *a_fd* so that operations on *a_nxo* will be backed by the file descriptor.

void *nxo_file_synthetic*(cw_nxo_t *a_nxo, cw_nxo_file_read_t *a_read, cw_nxo_file_write_t *a_write, cw_nxo_file_ref_iter_t *a_ref_iter, cw_nxo_file_delet_t *a_delete, void *a_arg):

Input(s):

- **a_nxo:** Pointer to a file *nxo*.
- **a_read:** Pointer to a read function.
- **a_write:** Pointer to a write function.
- **a_ref_iter:** Pointer to a reference iterator function.
- **a_delete:** Pointer to a destructor function.
- **a_arg:** Opaque pointer to be passed to the read and write functions.

Output(s): None.

Exception(s): None.

Description: Set up *a_nxo* to call the specified read and write functions to satisfy file operations.

cw_nxn_t *nxo_file_open*(cw_nxo_t *a_nxo, const cw_uint8_t *a_filename, cw_uint32_t a_nlen, const cw_uint8_t *a_flags, cw_uint32_t a_flen):

Input(s):

- **a_nxo:** Pointer to a file *nxo*.
- **a_filename:** Pointer to a string (not required to be '\0' terminated) that represents a filename.
- **a_nlen:** Length in bytes of *a_filename*.
- **a_flags:** Pointer to a string (not required to be '\0' terminated) that represents a file mode:
  - **"r":** Read only.
  - **"r+":** Read/write, starting at offset 0.
  - **"w":** Write only. Create file if necessary. Truncate file if non-zero length.
  - **"w+":** Read/write, starting at offset 0. Create file if necessary.
  - **"a":** Write only, starting at end of file.
  - **"a+":** Read/write, starting at end of file.
- **a_flen:** Length in bytes of *a_flags*.

Output(s):

- **retval:**
  - **NXN_ZERO.**
  - **NXN_ioerror.**
  - **NXN_invalidfileaccess.**
  - **NXN_limitcheck.**

Exception(s): None.

Description: Open a file.

cw_nxn_t *nxo_file_close*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a file *nxo*.

Output(s):

retval:

NXN_ZERO.

NXN_ioerror.

Exception(s): None.

Description: Close a file.

cw_sint32_t *nxo_file_fd_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a file *nxo*.

Output(s):

retval:

-1: Invalid or synthetic file.

>= 0: File descriptor number.

Exception(s): None.

Description: Return the file descriptor associated with *a_nxo*.

cw_bool_t *nxo_file_nonblocking_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a file *nxo*.

Output(s):

retval:

FALSE: Blocking file.

TRUE: Non-blocking file.

Exception(s): None.

Description: Return the non-blocking mode for *a_nxo*.

cw_bool_t *nxo_file_nonblocking_set*(cw_nxo_t *a_nxo, cw_bool_t a_nonblocking):

Input(s):

a_nxo: Pointer to a file *nxo*.

a_nonblocking: Non-blocking mode to set *nxo* to.

Output(s):

retval:

FALSE: Success.

TRUE: I/O error or non-POSIX file.

Exception(s): None.

Description: Set the non-blocking mode for *a_nxo* to *a_nonblocking*.

cw_sint32_t *nxo_file_read*(cw_nxo_t *a_nxo, cw_uint32_t a_len, cw_uint8_t *r_str):

Input(s):

a_nxo: Pointer to a file *nxo*.

a_len: Length in bytes of *r_str*.

r_str: Pointer to a string to store read data into.

Output(s):

retval:

-1: NXN_ioerror.

>= 0: Number of bytes of data read into *r_str*.

r_str: If *retval* is non-negative, *retval* bytes of read data.

Exception(s): None.

Description: Read data.

cw_nxn_t *nxo_file_readline*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, cw_bool_t a_locking, cw_nxo_t *r_string, cw_bool_t *r_eof):

Input(s):

a_nxo: Pointer to a file *nxo*.

a_nx: Pointer to an *nx*.

a_locking: Implicit locking mode.

r_string: Pointer to an *nxo*.

r_eof: Pointer to a cw_bool_t.

Output(s):

retval:

NXN_ZERO.

NXN_ioerror.

r_string: If *retval* is NXN_ZERO, a string object, otherwise unmodified.

***r_eof:**

FALSE: End of file not reached.

TRUE: End of file reached.

Exception(s):

CW_ONYXX_OOM.

Description: Read a line, terminated by "\r", "\r\n", or EOF.

cw_nxn_t *nxo_file_write*(cw_nxo_t *a_nxo, const cw_uint8_t *a_str, cw_uint32_t a_len, cw_uint32_t *r_count):

Input(s):

a_nxo: Pointer to a file *nxo*.

a_str: Pointer to data to write.

a_len: Length of *a_str*.

r_count: Pointer to a location to store the number of bytes written, or NULL.

Output(s):

retval:

NXN_ZERO.

NXN_ioerror.

***r_count:** If *r_count* is non-NULL, and retval is NXN_ZERO, number of bytes written, otherwise undefined.

Exception(s): None.

Description: Write the *a_len* bytes of data pointed to *a_str*. If the file is in non-blocking mode, it is possible for **r_count* to be as small as 0 for a successful write.

cw_nxn_t *nxo_file_truncate*(cw_nxo_t *a_nxo, off_t a_length):

Input(s):

a_nxo: Pointer to a file *nxo*.

a_length: Length to set file to.

Output(s):

retval:

NXN_ZERO.

NXN_ioerror.

Exception(s): None.

Description: Truncate or extend the file associated with *a_nxo* so that it is *a_length* bytes long.

cw_nxoi_t *nxo_file_position_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a file *nxo*.

Output(s):

retval:

-1: NXN_ioerror.

>= 0: Current file position.

Exception(s): None.

Description: Get the current file position.

cw_nxn_t *nxo_file_position_set*(cw_nxo_t *a_nxo, cw_nxoi_t a_position):

Input(s):

a_nxo: Pointer to a file *nxo*.

a_position: File position.

Output(s):

retval:

NXN_ZERO.

NXN_ioerror.

Exception(s): None.

Description: Move the current file position to *a_position*.

cw_uint32_t *nxo_file_buffer_size_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a file *nxo*.

Output(s):

retval: Size in bytes of the internal data buffer.

Exception(s): None.

Description: Return the size of the internal data buffer.

void *nxo_file_buffer_size_set*(cw_nxo_t *a_nxo, cw_uint32_t a_size):

Input(s):

a_nxo: Pointer to a file *nxo*.

a_size: Size in bytes of internal buffer to use.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Use an internal buffer of *a_size* bytes.

cw_nxoi_t *nxo_file_buffer_count*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a file *nxo*.

Output(s):

retval: Current number of buffered bytes available for reading.

Exception(s): None.

Description: Return the current number of buffered bytes available for reading.

cw_nxn_t *nxo_file_buffer_flush*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a file *nxo*.

Output(s):

retval:

NXN_ZERO.

NXN_ioerror.

Exception(s): None.

Description: Flush any buffered write data to disk, and discard any buffered read data.

4.10.17 nxo_fino

The *nxo_fino* class is a subclass of the *nxo* class.

API

void *nxo_fino_new*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to an *nxo*.

Output(s): None.

Exception(s): None.

Description: Constructor.

4.10.18 nxo_hook

The *nxo_hook* class is a subclass of the *nxo* class.

API

void *cw_nxo_hook_eval_t*(void *a_data, cw_nxo_t *a_thread):

- **Input(s):**
  - **a_data:** Opaque data pointer.
  - **a_thread:** Pointer to a thread *nxo*.
- **Output(s):** None.
- **Exception(s):** Hook-dependent.
- **Description:** Evaluation function typedef.

cw_nxoe_t * *cw_nxo_hook_ref_iter_t*(void *a_data, cw_bool_t a_reset):

- **Input(s):**
  - **a_data:** Opaque data pointer.
  - **a_reset:**
    - **FALSE:** At least one iteration has already occurred.
    - **TRUE:** First iteration.
- **Output(s):**
  - **retval:**
    - **non-NULL:** Pointer to an *nxoe*.
    - **NULL:** No more references.
- **Exception(s):** None.
- **Description:** Reference iterator function typedef.

cw_bool_t *cw_nxo_hook_delete_t*(void *a_data, cw_nx_t *a_nx, cw_uint32_t a_iter):

- **Input(s):**
  - **a_data:** Opaque data pointer.
  - **a_nx:** Pointer to an *nx*.
  - **a_iter:** Garbage collector sweep iteration count (starts at 0). This value can be used to impose ordering of dependent object deletions.
- **Output(s):**
  - **retval:**
    - **FALSE:** Success.
    - **TRUE:** Defer deletion until a later garbage collector sweep iteration.
- **Exception(s):** None.
- **Description:** Destructor function typedef.

void *nxo_hook_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, void *a_data, cw_nxo_hook_eval_t *a_eval_f, cw_nxo_hook_ref_iter_t *a_ref_iter_f, cw_nxo_hook_delete_t *a_delete_f):

- **Input(s):**

a_nxo: Pointer to a hook *nxo*.

a_nx: Pointer to an *nx*.

a_data: Opaque data pointer to be passed to *a_eval_f*, *a_ref_iter_f*, and *a_delete_f*.

a_eval_f: Pointer to an evaluation function.

a_ref_iter_f: Pointer to a reference iterator function.

a_delete_f: Pointer to a destructor function.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

cw_nxo_t * *nxo_hook_tag_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a hook *nxo*.

Output(s):

retval: Pointer to the tag object associated with *a_nxo*.

Exception(s): None.

Description: Return a pointer to the tag object associated with *a_nxo*. This object pointer can safely be used for modifying the tag object.

void * *nxo_hook_data_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a hook *nxo*.

Output(s):

retval: Opaque data pointer.

Exception(s): None.

Description: Return the opaque data pointer associated with *a_nxo*.

void *nxo_hook_data_set*(cw_nxo_t *a_nxo, void *a_data):

Input(s):

a_nxo: Pointer to a hook *nxo*.

a_data: Opaque data pointer.

Output(s): None.

Exception(s): None.

Description: Set the opaque data pointer associated with *a_nxo*.

void *nxo_hook_eval*(cw_nxo_t *a_nxo, cw_nxo_t *a_thread):

Input(s):

a_nxo: Pointer to a hook *nxo*.

a_thread: Pointer to a thread *nxo*.

Output(s): None.

Exception(s): Hook-specific.

Description: Evaluate the *a_nxo*. If there is no evaluation function associated with *a_nxo*, it is pushed onto ostack.

4.10.19 nxo_integer

The *nxo_integer* class is a subclass of the *nxo* class.

API

void *nxo_integer_new*(cw_nxo_t *a_nxo, cw_nxoi_t a_val):

- **Input(s):**
  - **a_nxo:** Pointer to an integer *nxo*.
  - **a_val:** Initial value.
- **Output(s):** None.
- **Exception(s):** None.
- **Description:** Constructor.

cw_nxoi_t *nxo_integer_get*(const cw_nxo_t *a_nxo):

- **Input(s):**
  - **a_nxo:** Pointer to an integer *nxo*.
- **Output(s):**
  - **retval:** Value of *a_nxo*.
- **Exception(s):** None.
- **Description:** Return the value of *a_nxo*.

void *nxo_integer_set*(cw_nxo_t *a_nxo, cw_nxoi_t a_val):

- **Input(s):**
  - **a_nxo:** Pointer to an integer *nxo*.
  - **a_val:** Integer value.
- **Output(s):** None.
- **Exception(s):** None.
- **Description:** Set the value of *a_nxo* to *a_val*.

4.10.20 nxo_mark

The *nxo_mark* class is a subclass of the *nxo* class.

API

void *nxo_mark_new*(cw_nxo_t *a_nxo):

- **Input(s):**
  - **a_nxo:** Pointer to an *nxo*.
- **Output(s):** None.
- **Exception(s):** None.
- **Description:** Constructor.

4.10.21 nxo_mutex

The *nxo_mutex* class is a subclass of the *nxo* class.

API

void *nxo_mutex_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx):

Input(s):

a_nxo: Pointer to a mutex *nxo*.

a_nx: Pointer to an *nx*.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *nxo_mutex_lock*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a mutex *nxo*.

Output(s): None.

Exception(s): None.

Description: Lock *a_nxo*.

cw_bool_t *nxo_mutex_trylock*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a mutex *nxo*.

Output(s):

retval:

FALSE: Success.

TRUE: Failure.

Exception(s): None.

Description: Try to lock *a_nxo*, but return immediately with an error if unable to do so.

void *nxo_mutex_unlock*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a mutex *nxo*.

Output(s): None.

Exception(s): None.

Description: Unlock *a_nxo*.

4.10.22 nxo_name

The *nxo_name* class is a subclass of the *nxo* class.

API

void *nxo_name_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, const cw_uint8_t *a_str, cw_uint32_t a_len, cw_bool_t a_is_static):

Input(s):

a_nxo: Pointer to a name *nxo*.

a_nx: Pointer to an *nx*.

a_str: Pointer to a character string (not required to be '\0' terminated).

a_len: Length in bytes of *a_str*.

a_is_static:

FALSE: *a_str* may be modified or deallocated during the lifetime of the program.

TRUE: *a_str* will not be modified for the lifetime of the program.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

const cw_uint8_t * *nxo_name_str_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a name *nxo*.

Output(s):

retval: Pointer to a string that represents *a_nxo*.

Exception(s): None.

Description: Return a pointer to a string that represents *a_nxo*.

cw_uint32_t *nxo_name_len_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a name *nxo*.

Output(s):

retval: Length in bytes of the name associated with *a_nxo*.

Exception(s): None.

Description: Return the length in bytes of the name associated with *a_nxo*.

4.10.23 nxo_no

The *nxo_no* class is a subclass of the *nxo* class.

API

void *nxo_no_new*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to an *nxo*.

Output(s): None.

Exception(s): None.

Description: Constructor.

4.10.24 nxo_null

The *nxo_null* class is a subclass of the *nxo* class.

API

void *nxo_null_new*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to an *nxo*.

Output(s): None.

Exception(s): None.

Description: Constructor.

4.10.25 nxo_operator

The *nxo_operator* class is a subclass of the *nxo* class.

API

void *nxo_operator_new*(cw_nxo_t *a_nxo, cw_op_t *a_op, cw_nxn_t a_nxn):

Input(s):

a_nxo: Pointer to an operator *nxo*.

a_op: Pointer to an operator function.

a_nxn: NXN_ZERO, or an *nxn*.

Output(s): None.

Exception(s): None.

Description: Constructor.

cw_op_t * *nxo_operator_f*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to an operator *nxo*.

Output(s):

retval: Pointer to an operator function.

Exception(s): None.

Description: Return the operator function associated with *a_nxo*.

4.10.26 nxo_pmark

The *nxo_pmark* class is a subclass of the *nxo* class.

API

void *nxo_pmark_new*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to an *nxo*.

Output(s): None.

Exception(s): None.

Description: Constructor.

4.10.27 nxo_real

The *nxo_real* class is a subclass of the *nxo* class.

API

void *nxo_real_new*(cw_nxo_t *a_nxo, cw_nxor_t a_val):

Input(s):

a_nxo: Pointer to a real *nxo*.

a_val: Initial value.

Output(s): None.

Exception(s): None.

Description: Constructor.

cw_nxor_t *nxo_real_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a real *nxo*.

Output(s):

retval: Value of *a_nxo*.

Exception(s): None.

Description: Return the value of *a_nxo*.

void *nxo_real_set*(cw_nxo_t *a_nxo, cw_nxor_t a_val):

Input(s):

a_nxo: Pointer to a real *nxo*.

a_val: Real value.

Output(s): None.

Exception(s): None.

Description: Set the value of *a_nxo* to *a_val*.

4.10.28 nxo_regex

The *nxo_regex* class is a subclass of the *nxo* class.

API

cw_nxn_t *nxo_regex_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, const cw_uint8_t *a_pattern, cw_uint32_t a_len, cw_bool_t a_cont, cw_bool_t a_global, cw_bool_t a_insensitive, cw_bool_t a_multiline, cw_bool_t a_singleline):

Input(s):

- **a_nxo:** Pointer to a regex *nxo*.
- **a_nx:** Pointer to an *nx*.
- **a_pattern:** Pointer to a string that specifies a regular expression.
- **a_len:** Length of *a_pattern*.
- **a_cont:** Continue where last successful match ended if TRUE.
- **a_global:** Continue where last match ended if TRUE.
- **a_insensitive:** Match with case insensitivity if TRUE.
- **a_multiline:** Treat input as a multi-line string if TRUE.
- **a_singleline:** Treat input as a single line, so that the dot metacharacter matches any character, including a newline.

Output(s):

- **retval:**
  - **NXN_ZERO:** Success.
  - **NXN_regexerror:** Regular expression error.

Exception(s):

- **CW_ONYXX_OOM.**

Description: Constructor.

void *nxo_regex_match*(cw_nxo_t *a_nxo, cw_nxo_t *a_thread, cw_nxo_t *a_input, cw_bool_t *r_match):

Input(s):

- **a_nxo:** Pointer to a regex *nxo*.
- **a_thread:** Pointer to a thread *nxo*.
- **a_input:** Pointer to a string *nxo*.
- **r_match:** Pointer to a cw_bool_t.

Output(s):

- ***r_match:**
  - **TRUE:** Match successful.
  - **FALSE:** No match found.

Exception(s):

- **CW_ONYXX_OOM.**

Description: Look in *a_input* for a match to the regex pointed to by *a_nxo*. As a side effect, set the thread's match cache, which can be queried via *nxo_regex_submatch*().

cw_nxn_t *nxo_regex_nonew_match*(cw_nxo_t *a_thread, const cw_uint8_t *a_pattern, cw_uint32_t a_len, cw_bool_t a_cont, cw_bool_t a_global, cw_bool_t a_insensitive, cw_bool_t a_multiline, cw_bool_t a_singleline, cw_nxo_t *a_input, cw_bool_t *r_match):

Input(s):

a_thread: Pointer to a thread *nxo*.

a_pattern: Pointer to a string that specifies a regular expression.

a_len: Length of *a_pattern*.

a_cont: Continue where last successful match ended if TRUE.

a_global: Continue where last match ended if TRUE.

a_insensitive: Match with case insensitivity if TRUE.

a_multiline: Treat input as a multi-line string if TRUE.

a_singleline: Treat input as a single line, so that the dot metacharacter matches any character, including a newline.

a_input: Pointer to a string *nxo*.

r_match: Pointer to a cw_bool_t.

Output(s):

retval:

NXN_ZERO: Success.

NXN_regexerror: Regular expression error.

***r_match:**

TRUE: Match successful.

FALSE: No match found.

Exception(s):

CW_ONYXX_OOM.

Description: Look in *a_input* for a match to the regular expression specified by *a_pattern*, *a_len*, *a_cont*, *a_global*, *a_insensitive*, *a_multiline*, and *a_singleline*. As a side effect, set the thread's match cache, which can be queried via *nxo_regex_submatch*().

This function combines *nxo_regex_new*() and *nxo_regex_match*() in such a way that no Onyx regex object is created, thus providing a more efficient way of doing a one-off match.

void *nxo_regex_split*(cw_nxo_t *a_nxo, cw_nxo_t *a_thread, cw_uint32_t a_limit, cw_nxo_t *a_input, cw_nxo_t *r_array):

Input(s):

a_nxo: Pointer to a regex *nxo*.

a_thread: Pointer to a thread *nxo*.

a_limit: Maximum number of substrings to split *a_input* into. 0 is treated as infinity.

a_input: Pointer to a string *nxo*.

r_array: Pointer to an *nxo* to dup an array of substrings to.

Output(s):

***r_array:** An array of substrings.

Exception(s):

CW_ONYXX_OOM.

Description: Use the regex pointed to by *a_nxo* to find matches in *a_input* and create an array of substrings that contain the data between those matches.

If there are capturing subpatterns in the regular expression, also create substrings for those capturing subpatterns and insert them into the substring array.

As a special case, if the regular expression matches the empty string, split a single character. This avoids an infinite loop.

As a side effect, set the thread's match cache, which can be queried via *nxo_regex_submatch*(). Keep in mind that this function can match multiple times in a single invocation, so only the last match is available in this way.

cw_nxn_t *nxo_regex_nonew_split*(cw_nxo_t *a_thread, const cw_uint8_t *a_pattern, cw_uint32_t a_len, cw_bool_t a_insensitive, cw_bool_t a_multiline, cw_bool_t a_singleline, cw_uint32_t a_limit, cw_nxo_t *a_input, cw_nxo_t *r_array):

Input(s):

a_thread: Pointer to a thread *nxo*.

a_pattern: Pointer to a string that specifies a regular expression.

a_len: Length of *a_pattern*.

a_insensitive: Match with case insensitivity if TRUE.

a_multiline: Treat input as a multi-line string if TRUE.

a_singleline: Treat input as a single line, so that the dot metacharacter matches any character, including a newline.

a_limit: Maximum number of substrings to split *a_input* into. 0 is treated as infinity.

a_input: Pointer to a string *nxo*.

r_array: Pointer to an *nxo* to dup an array of substrings to.

Output(s):

retval:

NXN_ZERO: Success.

NXN_regexerror: Regular expression error.

***r_array:** An array of substrings.

Exception(s):

CW_ONYXX_OOM.

Description: Use the regex specified by *a_pattern*, *a_len*, *a_insensitive*, *a_multiline*, and *a_singleline* to find matches in *a_input* and create an array of substrings that contain the data between those matches.

If there are capturing subpatterns in the regular expression, also create substrings for those capturing subpatterns and insert them into the substring array.

As a special case, if the regular expression matches the empty string, split a single character. This avoids an infinite loop.

As a side effect, set the thread's match cache, which can be queried via *nxo_regex_submatch*(). Keep in mind that this function can match multiple times in a single invocation, so only the last match is available in this way.

This function combines *nxo_regex_nex*() and *nxo_regex_split*() in such a way that no Onyx regex object is created, thus providing a more efficient way of doing a one-off split.

void *nxo_regex_submatch*(cw_nxo_t *a_thread, cw_uint32_t a_capture, cw_nxo_t *r_match):

Input(s):

a_thread: Pointer to a thread *nxo*.

a_capture: Index of captured subpattern to create a substring for:

0: Get substring of input text that matched the regular expression.

>0: Get substring of input text that matched the specified capturing subpattern.

r_match: Pointer to an *nxo* to dup a substring reference to.

Output(s):

***r_match:** An *nxo*:

null: Subpattern not matched.

string: A substring of text that corresponds to the captured subpattern specified by *a_capture*.

Exception(s):

CW_ONYXX_OOM.

Description: Create a substring using the calling thread's match cache that corresponds to capturing subpattern *a_capture*.

Each thread has a match cache that is used by various *regex* and *regsub* functions. That cache stores a reference to the string that was most recently matched against, as well as offsets and lengths of the match and capturing subpatterns. Since creating substrings puts pressure on the garbage collector, substring creation is done lazily (i.e. when this function is called). Normally, a program has little need to ask for the same substring twice, so the created substrings are not cached. That means that if this function is called twice in succession with the same arguments, two different (but equivalent) substrings will be returned.

4.10.29 nxo_regsub

The *nxo_regsub* class is a subclass of the *nxo* class.

API

cw_nxn_t *nxo_regsub_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, const cw_uint8_t *a_pattern, cw_uint32_t a_plen, cw_bool_t a_global, cw_bool_t a_insensitive, cw_bool_t a_multiline, cw_bool_t a_singleline, const cw_uint8_t *a_template, cw_uint32_t a_tlen):

Input(s):

a_nxo: Pointer to a regsub *nxo*.

a_nx: Pointer to an *nx*.

a_pattern: Pointer to a string that specifies a regular expression.

a_plen: Length of *a_pattern*.

a_global: Substitute as many times as possible if TRUE.

a_insensitive: Match with case insensitivity if TRUE.

a_multiline: Treat input as a multi-line string if TRUE.

a_singleline: Treat input as a single line, so that the dot metacharacter matches any character, including a newline.

a_template: Pointer to a string that specifies a substitution template.

a_tlen: Length of *a_template*.

Output(s):

retval:

NXN_ZERO: Success.

NXN_regexerror: Regular expression error.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *nxo_regsub_subst*(cw_nxo_t *a_nxo, cw_nxo_t *a_thread, cw_nxo_t *a_input, cw_nxo_t *r_output, cw_uint32_t *r_count):

Input(s):

a_nxo: Pointer to a regsub *nxo*.

a_thread: Pointer to a thread *nxo*.

a_input: Pointer to a string *nxo*.

r_output: Pointer to an *nxo* to dup a string reference to.

r_count: Pointer to a cw_uint32_t.

Output(s):

***r_output:** A string that was created by substituting regular expression matches according to a substitution template.

***r_count:** Number of substitutions made. If 0 substitutions were made, **r_output* is a duplicate of *a_input*, rather than a copy.

Exception(s):

CW_ONYXX_OOM.

Description: Create a string by substituting according to *a_template* for each substring within *input* that matches a regular expression.

As a side effect, set the thread's match cache, which can be queried via *nxo_regex_submatch*().

cw_nxn_t *nxo_regsub_nonew_subst*(cw_nxo_t *a_thread, const cw_uint8_t *a_pattern, cw_uint32_t a_plen, cw_bool_t a_global, cw_bool_t a_insensitive, cw_bool_t a_multiline, cw_bool_t a_singleline, const cw_uint8_t *a_template, cw_uint32_t a_tlen, cw_nxo_t *a_input, cw_nxo_t *r_output, cw_uint32_t *r_count):

Input(s):

a_thread: Pointer to a thread *nxo*.

a_pattern: Pointer to a string that specifies a regular expression.

a_plen: Length of *a_pattern*.

a_global: Substitute as many times as possible if TRUE.

a_insensitive: Match with case insensitivity if TRUE.

a_multiline: Treat input as a multi-line string if TRUE.

a_singleline: Treat input as a single line, so that the dot metacharacter matches any character, including a newline.

a_template: Pointer to a string that specifies a substitution template.

a_tlen: Length of *a_template*.

a_input: Pointer to a string *nxo*.

r_output: Pointer to an *nxo* to dup a string reference to.

r_count: Pointer to a cw_uint32_t.

Output(s):

retval:

NXN_ZERO: Success.

NXN_regexerror: Regular expression error.

***r_output:** A string that was created by substituting regular expression matches (specified by *a_pattern*) according to *a_template*.

***r_count:** Number of substitutions made. If 0 substitutions were made, **r_output* is a duplicate of *a_input*, rather than a copy.

Exception(s):

CW_ONYXX_OOM.

Description: Create a string by substituting according to *a_template* for each substring within *input* that matches a regular expression.

As a side effect, set the thread's match cache, which can be queried via *nxo_regex_submatch*().

This function combines *nxo_regsub_new*() and *nxo_regsub_subst*() in such a way that no Onyx regsub object is created, thus providing a more efficient way of doing a one-off subst.

4.10.30 nxo_stack

The *nxo_stack* class is a subclass of the *nxo* class.

API

void *nxo_stack_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, cw_bool_t a_locking):

Input(s):

a_nxo: Pointer to a stack *nxo*.

a_nx: Pointer to an *nx*.

a_locking: Implicit locking mode.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *nxo_stack_copy*(cw_nxo_t *a_to, cw_nxo_t *a_from):

Input(s):

a_to: Pointer to a stack *nxo*.

a_from: Pointer to a stack *nxo*.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Copy the objects in *a_from* onto *a_to*.

cw_uint32_t *nxo_stack_count*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a stack *nxo*.

Output(s):

retval: Number of objects on *a_nxo*.

Exception(s): None.

Description: Return the number of objects on *a_nxo*.

cw_nxo_t * *nxo_stack_push*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a stack *nxo*.

Output(s):

retval: Pointer to a no *nxo* that has been pushed onto *a_nxo*.

Exception(s):

CW_ONYXX_OOM.

Description: Push a no *nxo* onto *a_nxo* and return a pointer to it.

cw_nxo_t * *nxo_stack_bpush*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a stack *nxo*.

Output(s):

retval: Pointer to a no *nxo* that has been pushed onto the bottom of *a_nxo*.

Exception(s):

CW_ONYXX_OOM.

Description: Push a no *nxo* onto the bottom of *a_nxo* and return a pointer to it.

cw_nxo_t * *nxo_stack_under_push*(cw_nxo_t *a_nxo, cw_nxo_t *a_object):

Input(s):

a_nxo: Pointer to a stack *nxo*.

a_object: Pointer to an *nxo* on *a_nxo*.

Output(s):

retval: Pointer to a no *nxo* that has been pushed under *a_object* on *a_nxo*.

Exception(s):

CW_ONYXX_OOM.

Description: Push a no *nxo* under *a_object* on *a_nxo*.

cw_bool_t *nxo_stack_pop*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a stack *nxo*.

Output(s):

retval:

FALSE: Success.

TRUE: Stack underflow.

Exception(s): None.

Description: Pop an object off of *a_nxo*.

cw_bool_t *nxo_stack_bpop*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a stack *nxo*.

Output(s):

retval:

FALSE: Success.

TRUE: Stack underflow.

Exception(s): None.

Description: Pop an object off the bottom of *a_nxo*.

cw_bool_t *nxo_stack_npop*(cw_nxo_t *a_nxo, cw_uint32_t a_count):

Input(s):

a_nxo: Pointer to a stack *nxo*.

a_count: Number of objects to pop off of *a_nxo*.

Output(s):

retval:

FALSE: Success.

TRUE: Stack underflow.

Exception(s): None.

Description: Pop *a_count* objects off of *a_nxo*.

cw_bool_t *nxo_stack_nbpop*(cw_nxo_t *a_nxo, cw_uint32_t a_count):

Input(s):

a_nxo: Pointer to a stack *nxo*.

a_count: Number of objects to pop off the bottom of *a_nxo*.

Output(s):

retval:

FALSE: Success.

TRUE: Stack underflow.

Exception(s): None.

Description: Pop *a_count* objects off the bottom of *a_nxo*.

void *nxo_stack_remove*(cw_nxo_t *a_nxo, cw_nxo_t *a_object):

Input(s):

a_nxo: Pointer to a stack *nxo*.

a_object: Pointer to an object on *a_nxo*.

Output(s): None.

Exception(s): None. Remove *a_object* from *a_nxo*.

Description:

cw_nxo_t * *nxo_stack_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a stack *nxo*.

Output(s):

retval:

non-NULL: Pointer to the top *nxo* on *a_nxo*.

NULL: Stack underflow.

Exception(s): None.

Description: Return a pointer to the top *nxo* on *a_nxo*.

cw_nxo_t * *nxo_stack_bget*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a stack *nxo*.

Output(s):

retval:

non-NULL: Pointer to the bottom *nxo* on *a_nxo*.

NULL: Stack underflow.

Exception(s): None.

Description: Return a pointer to the bottom *nxo* on *a_nxo*.

cw_nxo_t * *nxo_stack_nget*(const cw_nxo_t *a_nxo, cw_uint32_t a_index):

Input(s):

a_nxo: Pointer to a stack *nxo*.

a_index: Index of object in *a_nxo* to return a pointer to.

Output(s):

retval:

non-NULL: Pointer to the *nxo* on *a_nxo* at index *a_index*.

NULL: Stack underflow.

Exception(s): None.

Description: Return a pointer to the *nxo* on *a_nxo* at index *a_index*.

cw_nxo_t * *nxo_stack_nbget*(const cw_nxo_t *a_nxo, cw_uint32_t a_index):

Input(s):

a_nxo: Pointer to a stack *nxo*.

a_index: Index, counting from the bottom, of object in *a_nxo* to return a pointer to.

Output(s):

retval:

non-NULL: Pointer to the *nxo* on *a_nxo* at index *a_index*, counting from the bottom.

NULL: Stack underflow.

Exception(s): None.

Description: Return a pointer to the *nxo* on *a_nxo* at index *a_index*, counting from the bottom.

cw_nxo_t * *nxo_stack_down_get*(const cw_nxo_t *a_nxo, cw_nxo_t *a_object):

Input(s):

a_nxo: Pointer to a stack *nxo*.

a_object: Pointer to an object on *a_nxo*, or NULL for the top object on *a_nxo*.

Output(s):

retval:

non-NULL: Pointer to the *nxo* on *a_nxo* under *a_object*.

NULL: Stack underflow.

Exception(s): None. Return a pointer to the *nxo* on *a_nxo* under *a_object*.

Description:

cw_nxo_t * *nxo_stack_up_get*(const cw_nxo_t *a_nxo, cw_nxo_t *a_object):

Input(s):

a_nxo: Pointer to a stack *nxo*.

a_object: Pointer to an object on *a_nxo*, or NULL for the bottom object on *a_nxo*.

Output(s):

retval:

non-NULL: Pointer to the *nxo* on *a_nxo* over *a_object*.

NULL: Stack underflow.

Exception(s): None. Return a pointer to the *nxo* on *a_nxo* over *a_object*.

Description:

cw_bool_t *nxo_stack_exch*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a stack *nxo*.

Output(s):

retval:

FALSE: Success.

TRUE: Stack underflow.

Exception(s): None.

Description: Exchange the top two objects on *a_nxo*.

void *nxo_stack_rot*(cw_nxo_t *a_nxo, cw_sint32_t a_amount):

Input(s):

a_nxo: Pointer to a stack *nxo*.

a_amount: Amount to rotate upward. A negative value rotates downward.

Output(s): None.

Exception(s): None.

Description: Rotate *a_nxo* up by *a_amount*.

cw_bool_t *nxo_stack_roll*(cw_nxo_t *a_nxo, cw_uint32_t a_count, cw_sint32_t a_amount):

Input(s):

a_nxo: Pointer to a stack *nxo*.

a_count: Number of objects in roll region.

a_amount: Amount to roll upward. A negative value rolls downward.

Output(s):

retval:

FALSE: Success.

TRUE: Stack underflow.

Exception(s): None.

Description: Roll the top *a_count* objects on *a_nxo* up by *a_amount*.

4.10.31 nxo_string

The *nxo_string* class is a subclass of the *nxo* class. Strings are not '\0'-terminated, mainly since substrings are references to other strings, and the termination character wouldn't be consistently useful. *nxo_string_cstring*() is useful for creating '\0'-terminated copies of strings for situations where other C functions expect terminated strings.

API

void *nxo_string_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx, cw_bool_t a_locking, cw_uint32_t a_len):

Input(s):

- **a_nxo:** Pointer to a string *nxo*.
- **a_nx:** Pointer to an *nx*.
- **a_locking:** Implicit locking mode.
- **a_len:** Length in bytes of string to create.

Output(s): None.

Exception(s):

- **CW_ONYXX_OOM.**

Description: Constructor.

void *nxo_string_substring_new*(cw_nxo_t *a_nxo, cw_nxo_t *a_string, cw_nx_t *a_nx, cw_uint32_t a_offset, cw_uint32_t a_len):

Input(s):

- **a_nxo:** Pointer to a string *nxo*.
- **a_string:** Pointer to a string *nxo* to create a substring of.
- **a_nx:** Pointer to an *nx*.
- **a_offset:** Offset into *a_string*.
- **a_len:** Length in bytes of substring to create.

Output(s): None.

Exception(s):

- **CW_ONYXX_OOM.**

Description: Substring constructor.

void *nxo_string_copy*(cw_nxo_t *a_to, cw_nxo_t *a_from):

Input(s):

- **a_to:** Pointer to a string *nxo*.
- **a_from:** Pointer to a string *nxo*.

Output(s): None.

Exception(s): None.

Description: Copy the contents of *a_from* to *a_to*. The length of *a_to* must be at least that of *a_from*.

void *nxo_string_cstring*(cw_nxo_t *a_to, cw_nxo_t *a_from, cw_nxo_t *a_thread):

Input(s):

a_to: Pointer to an *nxo*.

a_from: Pointer to a string or name *nxo*.

a_thread: Pointer to a thread *nxo*.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Create a copy of *a_from*, but append a '\0' character to make it usable in calls to typical C functions that expect a terminated string.

cw_uint32_t *nxo_string_len_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a string *nxo*.

Output(s):

retval: Length of *a_nxo*.

Exception(s): None.

Description: Return the length of *a_nxo*.

void *nxo_string_el_get*(const cw_nxo_t *a_nxo, cw_nxoi_t a_offset, cw_uint8_t *r_el):

Input(s):

a_nxo: Pointer to a string *nxo*.

a_offset: Offset of character to get.

r_el: Pointer to space to copy a character to.

Output(s):

***r_el:** A copy of the character of *a_nxo* at offset *a_offset*.

Exception(s): None.

Description: Get a copy of the character of *a_nxo* at offset *a_offset*.

void *nxo_string_el_set*(cw_nxo_t *a_nxo, cw_uint8_t a_el, cw_nxoi_t a_offset):

Input(s):

a_nxo: Pointer to a string *nxo*.

a_el: A character.

a_offset: Offset of character in *a_nxo* to replace with *a_el*.

Output(s): None.

Exception(s): None.

Description: Copy *a_el* into the element of *a_nxo* at offset *a_offset*.

void *nxo_string_lock*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a string *nxo*.

Output(s): None.

Exception(s): None.

Description: If implicit locking is activated for *a_nxo*, lock it.

void *nxo_string_unlock*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a string *nxo*.

Output(s): None.

Exception(s): None.

Description: If implicit locking is activated for *a_nxo*, unlock it.

cw_uint8_t * *nxo_string_get*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a string *nxo*.

Output(s):

retval: Pointer to the string internal to *a_nxo*.

Exception(s): None.

Description: Return a pointer to the string internal to *a_nxo*.

void *nxo_string_set*(cw_nxo_t *a_nxo, cw_uint32_t a_offset, const cw_uint8_t *a_str, cw_uint32_t a_len):

Input(s):

a_nxo: Pointer to a string *nxo*.

a_offset: Offset into *a_nxo* to replace.

a_str: String to replace a range of *a_nxo* with.

a_len: Length in bytes of *a_str*.

Output(s): None.

Exception(s): None.

Description: Replace *a_len* bytes of *a_nxo* at offset *a_offset* with *a_str*.

4.10.32 nxo_thread

The *nxo_thread* class is a subclass of the *nxo* class.

The *threadp* class is a helper class that contains scanner position information. The *threadp* state is used when recording syntax errors.

API

void *nxo_threadp_new*(cw_nxo_threadp_t *a_threadp):

Input(s):

a_threadp: Pointer to space for a *threadp*.

Output(s): None.

Exception(s): None.

Description: Constructor.

void *nxo_threadp_delete*(cw_nxo_threadp_t *a_threadp, cw_nxo_t *a_thread):

Input(s):

a_threadp: Pointer to a *threadp*.

a_thread: Pointer to a thread *nxo*.

Output(s): None.

Exception(s): None.

Description: Destructor.

void *nxo_threadp_position_get*(const cw_nxo_threadp_t *a_threadp, cw_uint32_t *r_line, cw_uint32_t *r_column):

Input(s):

a_threadp: Pointer to space for a *threadp*.

r_line: Pointer to a location to store a line number.

r_column: Pointer to a location to store a column number.

Output(s):

***r_line:** Line number.

***r_column:** Column number.

Exception(s): None.

Description: Retrieve the line number and column number.

void *nxo_threadp_position_set*(cw_nxo_threadp_t *a_threadp, cw_uint32_t a_line, cw_uint32_t a_column):

Input(s):

a_threadp: Pointer to space for a *threadp*.

a_line: Line number.

a_column: Column number.

Output(s): None.

Exception(s): None.

Description: Set the line number and column number.

void *nxo_thread_new*(cw_nxo_t *a_nxo, cw_nx_t *a_nx):

Input(s):

a_nxo: Pointer to a thread *nxo*.

a_nx: Pointer to an *nx*.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor.

void *nxo_thread_start*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s): None.

Exception(s): Application dependent.

Description: Start a thread running by calling the **start** operator such that the top object on ostack will be executed.

void *nxo_thread_exit*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s): None.

Exception(s): None.

Description: Terminate the thread. This has the same effect as a detached thread exiting. Calling this function may is necessary (depending on the application) to allow the thread to be garbage collected, much the same way as the **detach** and **join** operators do.

void *nxo_thread_thread*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: Create a new thread. The new thread calls *nxo_thread_start*().

void *nxo_thread_detach*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s): None.

Exception(s): None.

Description: Detach *a_nxo* so that when it exits it can be garbage collected.

void *nxo_thread_join*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s): None.

Exception(s): None.

Description: Wait for *a_nxo* to exit.

cw_nxo_threadts_t *nxo_thread_state*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: The current scanner state of *a_nxo*.

THREADTS_START: Start state.
THREADTS_COMMENT: '%' seen, but no line break yet.
THREADTS_INTEGER: Scanning an integer.
THREADTS_INTEGER_RADIX: Scanning a radix integer.

THREADTS_REAL_FRAC: Scanning the fractional portion of a real.

THREADTS_REAL_EXP: Scanning the exponent porttion of a real.

THREADTS_STRING: Scanning a string.

THREADTS_STRING_NEWLINE_CONT: '\r' seen in a string.

THREADTS_STRING_PROT_CONT: '\\' seen in a string.

THREADTS_STRING_CRLF_CONT: '\' '\r' seen in a string.

THREADTS_STRING_CTRL_CONT: '\' 'c' seen in a string.

THREADTS_STRING_HEX_CONT: '\' 'x' seen in a string.

THREADTS_STRING_HEX_FINISH: First hex digit of a "\xDD" string escape sequence seen.

THREADTS_NAME_START: '!', '$', or '~' seen.

THREADTS_NAME: Scanning a name.

Exception(s): None.

Description: Return the current scanner state. In general this is only useful when implementing an interactive environment for which the prompt behaves differently depending on what state the scanner is in. For example the interactive *onyx* shell needs only to know whether the scanner is in the start state.

cw_bool_t *nxo_thread_deferred*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval:

FALSE: Execution is not deferred.

TRUE: Execution is deferred.

Exception(s): None.

Description: Return whether the scanner is currently in deferred execution mode. See Section 2.2 for information on deferred execution. In general this is only useful when implementing an interactive environment for which the prompt behaves differently depending on what state the scanner is in.

void *nxo_thread_reset*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s): None.

Exception(s): None.

Description: Reset the scanner to the start state, and turn deferral off. This is a dangerous feature that should be used with great care. *nxo_no* objects should never be visible from inside the interpreter, so the caller must assure that any *nxo_no* objects are removed before further processing is done in the context of *a_nxo*.

void *nxo_thread_loop*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s): None.

Exception(s): Application specific.

Description: Execute the top object on estack. The caller is responsible for placing the object on estack, but it is removed before this function returns.

void *nxo_thread_interpret*(cw_nxo_t *a_nxo, cw_nxo_threadp_t *a_threadp, const cw_uint8_t *a_str, cw_uint32_t a_len):

Input(s):

a_nxo: Pointer to a thread *nxo*.

a_threadp: A *threadp*.

a_str: Pointer to a string to interpret.

a_len: Length in bytes of *a_str*.

Output(s): None.

Exception(s): Application specific.

Description: Interpret the string pointed to by *a_str*.

void *nxo_thread_flush*(cw_nxo_t *a_nxo, cw_nxo_threadp_t *a_threadp):

Input(s):

a_nxo: Pointer to a thread *nxo*.

a_threadp: A *threadp*.

Output(s): None.

Exception(s): Application specific.

Description: Do the equivalent of interpreting a carriage return in order to force acceptance of the previous token if no whitespace has yet followed.

void *nxo_thread_nerror*(cw_nxo_t *a_nxo, cw_nxn_t a_nxn):

Input(s):

a_nxo: Pointer to a thread *nxo*.

a_nxn: An nxn corresponding to the name of an error.

Output(s): None.

Exception(s): Application dependent.

Description: Throw an error.

void *nxo_thread_serror*(cw_nxo_t *a_nxo, const cw_uint8_t a_str, cw_uint32_t a_len):

Input(s):

a_nxo: Pointer to a thread *nxo*.

a_str: Pointer to a string that represents the name of an error.

a_len: The length of *a_str*.

Output(s): None.

Exception(s): Application dependent.

Description: Throw an error.

cw_bool_t *nxo_thread_dstack_search*(cw_nxo_t *a_nxo, cw_nxo_t *a_key, cw_nxo_t *r_value):

Input(s):

a_nxo: Pointer to a thread *nxo*.

a_key: Pointer to an *nxo*.

r_value: Pointer to an *nxo*.

Output(s):

retval:

FALSE: Success.

TRUE: *a_key* not found on dstack.

r_value: Top value in dstack associated with *a_key*.

Exception(s): None.

Description: Search dstack for the topmost definition of *a_key* and dup its value to *r_value*.

cw_bool_t *nxo_thread_currentlocking*(const cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval:

FALSE: Implicit locking deactivated for new objects.

TRUE: Implicit locking activated for new objects.

Exception(s): None.

Description: Return whether implicit locking is activated for new objects.

void *nxo_thread_setlocking*(cw_nxo_t *a_nxo, cw_bool_t a_locking):

Input(s):

a_nxo: Pointer to a thread *nxo*.

a_locking:

FALSE: Do not implicitly lock new objects.

TRUE: Implicitly lock new objects.

Output(s): None.

Exception(s): None.

Description: Activate or deactivate implicit locking for new objects.

cw_nx_t * *nxo_thread_nx_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nx*.

Exception(s): None.

Description: Return the *nx* associated with *a_nxo*.

cw_nxo_t * *nxo_thread_userdict_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nxo* that can safely be used without risk of being garbage collected.

Exception(s): None.

Description: Return a pointer to the userdict associated with *a_nxo*.

cw_nxo_t * *nxo_thread_errordict_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nxo* that can safely be used without risk of being garbage collected.

Exception(s): None.

Description: Return a pointer to the errordict associated with *a_nxo*.

cw_nxo_t * *nxo_thread_currenterror_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nxo* that can safely be used without risk of being garbage collected.

Exception(s): None.

Description: Return a pointer to the currenterror associated with *a_nxo*.

cw_nxo_t * *nxo_thread_ostack_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nxo* that can safely be used without risk of being garbage collected.

Exception(s): None.

Description: Return a pointer to the ostack associated with *a_nxo*.

cw_nxo_t * *nxo_thread_dstack_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nxo* that can safely be used without risk of being garbage collected.

Exception(s): None.

Description: Return a pointer to the dstack associated with *a_nxo*.

cw_nxo_t * *nxo_thread_estack_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nxo* that can safely be used without risk of being garbage collected.

Exception(s): None.

Description: Return a pointer to the estack associated with *a_nxo*.

cw_nxo_t * *nxo_thread_istack_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nxo* that can safely be used without risk of being garbage collected.

Exception(s): None.

Description: Return a pointer to the istack associated with *a_nxo*.

cw_nxo_t * *nxo_thread_tstack_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nxo* that can safely be used without risk of being garbage collected.

Exception(s): None.

Description: Return a pointer to the tstack associated with *a_nxo*.

cw_nxo_t * *nxo_thread_stdin_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nxo* that can safely be used without risk of being garbage collected.

Exception(s): None.

Description: Return a pointer to the stdin associated with *a_nxo*.

void *nxo_thread_stdin_set*(cw_nxo_t *a_nxo, cw_nxo_t *a_stdin):

Input(s):

a_nxo: Pointer to a thread *nxo*.

a_stdin: Pointer to a file *nxo*.

Output(s): None.

Exception(s): None.

Description: Set *a_nxo*'s stdin to *a_stdin*.

cw_nxo_t * *nxo_thread_stdout_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nxo* that can safely be used without risk of being garbage collected.

Exception(s): None.

Description: Return a pointer to the stdout associated with *a_nxo*.

void *nxo_thread_stdout_set*(cw_nxo_t *a_nxo, cw_nxo_t *a_stdout):

Input(s):

a_nxo: Pointer to a thread *nxo*.

a_stdout: Pointer to a file *nxo*.

Output(s): None.

Exception(s): None.

Description: Set *a_nxo*'s stdout to *a_stdout*.

cw_nxo_t * *nxo_thread_stderr_get*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a thread *nxo*.

Output(s):

retval: Pointer to an *nxo* that can safely be used without risk of being garbage collected.

Exception(s): None.

Description: Return a pointer to the stderr associated with *a_nxo*.

void *nxo_thread_stderr_set*(cw_nxo_t *a_nxo, cw_nxo_t *a_stderr):

Input(s):

a_nxo: Pointer to a thread *nxo*.

a_stderr: Pointer to a file *nxo*.

Output(s): None.

Exception(s): None.

Description: Set *a_nxo*'s stderr to *a_stderr*.

4.10.33 ql

The *ql* macros implement operations on a list. The type of the list elements and which field of the elements to use are determined by arguments that are passed into the macros. The macros are optimized for speed and code size, which means that there is minimal error checking built in. As a result, care must be taken to assure that these macros are used as intended, or strange things can happen.

Internally, the list is represented as a ring, so with some care, the *ql* and *qr* interfaces can be used in conjunction with each other.

Since a *ql* is actually a ring, it is possible to have multiple *ql* heads that share the same ring. This works just fine, with the caveat that operations on one *ql* can have side-effects on another.

API

***ql_head*(<ql_type> a_type):**

Input(s):

a_type: Data type for the *ql* elements.

Output(s): A data structure that can be used as a *ql* head.

Exception(s): None.

Description: Generate code for a *ql* head data structure.

***ql_head_initializer*(<ql_type> *a_head):**

Input(s):

a_head: Pointer to a *ql* head.

Output(s): None.

Exception(s): None.

Description: Statically initialize a *ql* head.

***ql_elm*(<ql_type> a_type):**

Input(s):

a_type: Data type for the *ql* elements.

Output(s): A data structure that can be used as a *ql* element.

Exception(s): None.

Description: Generate code for a *ql* element data structure.

void *ql_new*(<ql_head> *a_head):

Input(s):

a_head: Pointer to a *ql* head.

Output(s): None.

Exception(s): None.

Description: Constructor.

void *ql_elm_new*(<ql_type> *a_elm, <field_name> a_field):

Input(s):

a_elm: Pointer to an element.

a_field: Field within the *ql* elements to use.

Output(s): None.

Exception(s): None.

Description: Constructor.

<ql_type> * *ql_first*(<ql_head> *a_head):

Input(s):

a_head: Pointer to a *ql* head.

Output(s):

retval:

non-NULL: Pointer to the first element in *a_head*.

NULL: *a_head* is empty.

Exception(s): None.

Description: Return a pointer to the first element in the *ql*.

<ql_type> * *ql_last*(<ql_head> *a_head, <field_name> a_field):

Input(s):

a_head: Pointer to a *ql* head.

a_field: Field within the *ql* elements to use.

Output(s):

retval:

non-NULL: Pointer to the last element in *a_head*.

NULL: *a_head* is empty.

Exception(s): None.

Description: Return a pointer to the last element in the *ql*.

<**ql_type**> * ***ql_next***(<**ql_head**> ***a_head,** <**ql_type**> ***a_elm,** <**field_name**> **a_field):**

Input(s):

a_head: Pointer to a *ql* head.

a_elm: Pointer to an element.

a_field: Field within the *ql* elements to use.

Output(s):

retval:

non-NULL: Pointer to the element after *a_elm*.

NULL: *a_elm* is the last element in *a_head*.

Exception(s): None.

Description: Return a pointer to the element in *a_head* after *a_elm*.

<**ql_type**> * ***ql_prev***(<**ql_head**> ***a_head,** <**ql_type**> ***a_elm,** <**field_name**> **a_field):**

Input(s):

a_head: Pointer to a *ql* head.

a_elm: Pointer to an element.

a_field: Field within the *ql* elements to use.

Output(s):

retval:

non-NULL: Pointer to the element before *a_elm*.

NULL: *a_elm* is the first element in *a_head*.

Exception(s): None.

Description: Return a pointer to the element in *a_head* before *a_elm*.

void *ql_before_insert*(<**ql_head**> ***a_head,** <**ql_type**> ***a_qlelm,** <**ql_type**> ***a_elm,** <**field_name**> **a_field):**

Input(s):

a_head: Pointer to a *ql* head.

a_qlelm: Pointer to an element within *a_head*.

a_elm: Pointer to an element.

a_field: Field within the *ql* elements to use.

Output(s): None.

Exception(s): None.

Description: Insert *a_elm* into *a_head* before *a_qlelm*.

void *ql_after_insert*(<**ql_type**> ***a_qlelm,** <**ql_type**> ***a_elm,** <**field_name**> **a_field):**

Input(s):

a_qlelm: Pointer to an element within *a_head*.

a_elm: Pointer to an element.

a_field: Field within the *ql* elements to use.

Output(s): None.

Exception(s): None.

Description: Insert *a_elm* into *a_head* after *a_qlelm*.

void *ql_head_insert*(<ql_head> *a_head, <ql_type> *a_elm, <field_name> a_field):

Input(s):

a_head: Pointer to a *ql* head.
a_elm: Pointer to an element.
a_field: Field within the *ql* elements to use.

Output(s): None.

Exception(s): None.

Description: Insert *a_elm* at the head of *a_head*.

void *ql_tail_insert*(<ql_head> *a_head, <ql_type> *a_elm, <field_name> a_field):

Input(s):

a_head: Pointer to a *ql* head.
a_elm: Pointer to an element.
a_field: Field within the *ql* elements to use.

Output(s): None.

Exception(s): None.

Description: Insert *a_elm* at the tail of *a_head*.

void *ql_remove*(<ql_head> *a_head, <ql_type> *a_elm, <field_name> a_field):

Input(s):

a_head: Pointer to a *ql* head.
a_elm: Pointer to an element.
a_field: Field within the *ql* elements to use.

Output(s): None.

Exception(s): None.

Description: Remove *a_elm* from *a_head*.

void *ql_head_remove*(<ql_head> *a_head, <ql_type> a_type, <field_name> a_field):

Input(s):

a_head: Pointer to a *ql* head.
a_type: Data type for the *ql* elements.
a_field: Field within the *ql* elements to use.

Output(s): None.

Exception(s): None.

Description: Remove the head element of *a_head*.

void *ql_tail_remove*(<ql_head> *a_head, <ql_type> a_type, <field_name> a_field):

Input(s):

a_head: Pointer to a *ql* head.
a_type: Data type for the *ql* elements.
a_field: Field within the *ql* elements to use.

Output(s): None.

Exception(s): None.

Description: Remove the tail element of *a_head*.

***ql_foreach*(<ql_type> *a_var, <ql_type> *a_head, <field_name> a_field):**

Input(s):
a_var: The name of a temporary variable to use for iteration.
a_head: Pointer to a *ql* head.
a_field: Field within the *ql* elements to use.

Output(s): None.

Exception(s): None.

Description: Iterate through the *ql*, storing a pointer to each element in *a_var* along the way.

***ql_reverse_foreach*(<ql_type> *a_var, <ql_type> *a_head, <field_name> a_field):**

Input(s):
a_var: The name of a temporary variable to use for iteration.
a_head: Pointer to a *ql* head.
a_field: Field within the *ql* elements to use.

Output(s): None.

Exception(s): None.

Description: Iterate through the *ql* in the reverse direction, storing a pointer to each element in *a_var* along the way.

4.10.34 qr

The *qr* macros implement operations on a ring. The type of the ring elements and which field of the elements to use are determined by arguments that are passed into the macros. The macros are optimized for speed and code size, which means that there is minimal error checking built in. As a result, care must be taken to assure that these are used as intended, or strange things can happen.

API

***qr*(<qr_type> a_type):**

Input(s):
a_type: Data type for the *qr*.

Output(s): A data structure that can be used for a *qr*.

Exception(s): None.

Description: Generate code for a *qr* data structure.

void *qr_new*(<qr_type> *a_qr, <field_name> a_field):

Input(s):

a_qr: Pointer to a *qr*.
a_field: Field within the *qr* elements to use.

Output(s): None.

Exception(s): None.

Description: Constructor.

<qr_type> * *qr_next*(<qr_type> *a_qr, <field_name> a_field):

Input(s):

a_qr: Pointer to a *qr*.
a_field: Field within the *qr* elements to use.

Output(s):

retval: Pointer to the next element in the *qr*.

Exception(s): None.

Description: Return a pointer to the next element in the *qr*.

<qr_type> * *qr_prev*(<qr_type> *a_qr, <field_name> a_field):

Input(s):

a_qr: Pointer to a *qr*.
a_field: Field within the *qr* elements to use.

Output(s):

retval: Pointer to the previous element in the *qr*.

Exception(s): None.

Description: Return a pointer to the previous element in the *qr*.

void *qr_before_insert*(<qr_type> *a_qrelm, <qr_type> *a_qr, <field_name> a_field):

Input(s):

a_qrelm: Pointer to an element in a *qr*.
a_qr: Pointer to an element that is the only element in its ring.
a_field: Field within the *qr* elements to use.

Output(s): None.

Exception(s): None.

Description: Insert *a_qr* before *a_qrelm*.

void *qr_after_insert*(<qr_type> *a_qrelm, <qr_type> *a_qr, <field_name> a_field):

Input(s):

a_qrelm: Pointer to an element in a *qr*.
a_qr: Pointer to an element that is the only element in its ring.
a_field: Field within the *qr* elements to use.

Output(s): None.

Exception(s): None.

Description: Insert *a_qr* after *a_qrelm*.

void *qr_meld*(<qr_type> *a_qr_a, <qr_type> *a_qr_b, <qr_type> a_type, <field_name> a_field):

Input(s):

- **a_qr_a:** Pointer to a *qr*.
- **a_qr_b:** Pointer to a *qr*.
- **a_type:** Data type for the *qr* elements.
- **a_field:** Field within the *qr* elements to use.

Output(s): None.

Exception(s): None.

Description: Meld *a_qr_a* and *a_qr_b* into one ring.

void *qr_split*(<qr_type> *a_qr_a, <qr_type> *a_qr_b, <qr_type> a_type, <field_name> a_field):

Input(s):

- **a_qr_a:** Pointer to a *qr*.
- **a_qr_b:** Pointer to a *qr*.
- **a_type:** Data type for the *qr* elements.
- **a_field:** Field within the *qr* elements to use.

Output(s): None.

Exception(s): None.

Description: Split a ring at *a_qr_a* and *a_qr_b*.

void *qr_remove*(<qr_type> *a_qr, <field_name> a_field):

Input(s):

- **a_qr:** Pointer to a *qr*.
- **a_field:** Field within the *qr* elements to use.

Output(s): None.

Exception(s): None.

Description: Remove *a_qr* from the ring.

***qr_foreach*(<qr_type> *a_var, <qr_type> *a_qr, <field_name> a_field):**

Input(s):

- **a_var:** The name of a temporary variable to use for iteration.
- **a_qr:** Pointer to a *qr*.
- **a_field:** Field within the *qr* elements to use.

Output(s): None.

Exception(s): None.

Description: Iterate through the *qr*, storing a pointer to each element in *a_var* along the way.

***qr_reverse_foreach*(<qr_type> *a_var, <qr_type> *a_qr, <field_name> a_field):**

Input(s):

- **a_var:** The name of a temporary variable to use for iteration.
- **a_qr:** Pointer to a *qr*.
- **a_field:** Field within the *qr* elements to use.

Output(s): None.

Exception(s): None.

Description: Iterate through the *qr* in the reverse direction, storing a pointer to each element in *a_var* along the way.

4.10.35 qs

The *qs* macros implement operations on a stack. The type of the stack elements and which field of the elements to use are determined by arguments that are passed into the macros. The macros are optimized for speed and code size, which means that there is minimal error checking built in. As a result, care must be taken to assure that these macros are used as intended, or strange things can happen.

API

qs_head(<**qs_type**> **a_type):**

Input(s):

a_type: Data type for the *qs*.

Output(s): A data structure that can be used as a *qs* head.

Exception(s): None.

Description: Generate code for a *qs* head data structure.

qs_head_initializer(<**qs_type**> ***a_head):**

Input(s):

a_head: Pointer to a *qs* head.

Output(s): None.

Exception(s): None.

Description: Statically initialize a *qs* head.

qs_elm(<**qs_elm_type**> **a_type):**

Input(s):

a_type: Data type for the *qs* elements.

Output(s): A data structure that can be used as a qs element.

Exception(s): None.

Description: Generate code for a *qs* element data structure.

void *qs_new*(<**qs_type**> ***a_head):**

Input(s):

a_head: Pointer to a *qs* head.

Output(s): None.

Exception(s): None.

Description: Constructor.

void *qs_elm_new*(<**qs_elm_type**> ***a_elm,** <**field_name**> **a_field):**

Input(s):

a_head: Pointer to a *qs* element.

a_field: Field within the *qs* elements to use.

Output(s): None.

Exception(s): None.

Description: Constructor.

<**qs_type**> * ***qs_top***(<**qs_type**> ***a_head**):

Input(s):

a_head: Pointer to a *qs* head.

Output(s):

retval: Pointer to the top element in the *qs*.

Exception(s): None.

Description: Return a pointer to the top element in the *qs*.

<**qs_type**> * ***qs_down***(<**qs_elm_type**> ***a_elm,** <**field_name**> **a_field):**

Input(s):

a_elm: Pointer to a *qs* element.

a_field: Field within the *qs* elements to use.

Output(s):

retval:

non-NULL: Pointer to the next element in the *qs*.

NULL: *a_elm* is the bottom element in the *qs*.

Exception(s): None.

Description: Return a pointer to the next element in the *qs* below *a_elm*.

void *qs_push*(<**qs_type**> ***a_head,** <**qs_elm_type**> ***a_elm,** <**field_name**> **a_field):**

Input(s):

a_head: Pointer to a *qs* head.

a_elm: Pointer to an element.

a_field: Field within the *qs* elements to use.

Output(s): None.

Exception(s): None.

Description: Push *a_elm* onto the *qs*.

void *qs_under_push*(<**qs_elm_type**> ***a_qselm,** <**qs_elm_type**> ***a_elm,** <**field_name**> **a_field):**

Input(s):

a_qselm: Pointer to a *qs* element.

a_elm: Pointer to an element.

a_field: Field within the *qs* elements to use.

Output(s): None.

Exception(s): None.

Description: Push *a_elm* under *a_qselm*.

void *qs_pop*(<**qs_type**> ***a_head,** <**field_name**> **a_field):**

Input(s):

a_head: Pointer to a *qs* head.

a_field: Field within the *qs* elements to use.

Output(s): None.

Exception(s): None.

Description: Pop an element off of *a_head*.

***qs_foreach*(<qs_elm_type> *a_var, <qs_type> *a_head, <field_name> a_field):**

Input(s):

a_var: The name of a temporary variable to use for iteration.

a_head: Pointer to a *qs* head.

a_field: Field within the *qs* elements to use.

Output(s): None.

Exception(s): None.

Description: Iterate down the *qs*, storing a pointer to each element in *a_var* along the way.

4.10.36 thd

The *thd* class implements a wrapper around the system POSIX threads library or GNU pth library. In most regards, this is a thin wrapper around the normal threading functionality, but some extra information is kept in order to allow implmentation of thread suspension/resumption, "critical sections", and "single sections".

The suspendibility of each thread is determined by the arguments passed to *thd_new*(). The initial thread is always suspendible. Other threads that are created via some mechanism other than *thd_new*() are not suspendible.

Depending on how *libonyx* is built, the additional functionality is implemented with the aid of the *SIGUSR1* and *SIGUSR2* signals. As a result, system calls may be interrupted by signals. The system calls will be automatically restarted if they have made no progress at the time of interruption, but will return a partial result otherwise. Therefore, if any of the additional functionality is utilized, the application must be careful to handle partial system call results. At least the following system calls can be interrupted: *read*(), *write*(), *sendto*(), *recvfrom*(), *sendmsg*(), *recvmsg*(), *ioctl*(), and *wait*(). See the system documentation for additional information.

API

cw_thd_t * *thd_new*(void *(*a_start_func)(void *), void *a_arg, cw_bool_t a_suspendible):

Input(s):

a_start_func: Pointer to a start function.

a_arg: Argument passed to *a_start_func*().

a_suspendible:

FALSE: Not suspendible.

TRUE: Suspendible.

Output(s):

retval: Pointer to a *thd*.

Exception(s):

CW_ONYXX_OOM.

Description: Constructor (creates a new thread).

void *thd_delete*(cw_thd_t *a_thd):

Input(s):

a_thd: Pointer to a *thd*.

Output(s): None.

Exception(s): None.

Description: Destructor.

void * *thd_join*(cw_thd_t *a_thd):

Input(s):

a_thd: Pointer to a *thd*.

Output(s):

retval: Return value from thread entry function.

Exception(s): None.

Description: Join (wait for) the thread associated with *a_thd*.

cw_thd_t * *thd_self*(void):

Input(s): None.

Output(s):

retval: Pointer to the calling thread's *thd* structure.

Exception(s): None.

Description: Return a pointer to the *thd* structure that corresponds to the calling thread.

void *thd_yield*(void):

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Give up the rest of the calling thread's time slice.

int *thd_sigmask*(int a_how, const sigset_t *a_set, sigset_t *r_oset):

Input(s):

a_how:

SIG_BLOCK: Block signals in *a_set*.

SIG_UNBLOCK: Unblock signals in *a_set*.

SIG_SETMASK: Set signal mask to *a_set*.

a_set: Pointer to a signal set.

r_oset:

non-NULL: Pointer space to store the old signal mask.

NULL: Ignored.

Output(s):

retval: Always zero, unless the arguments are invalid.

***r_oset:** Old signal set.

Exception(s): None.

Description: Set the calling thread's signal mask.

void *thd_crit_enter*(void):

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Enter a critical region where the calling thread may not be suspended by *thd_suspend*(), *thd_trysuspend*(), or *thd_single_enter*().

void *thd_crit_leave*(void):

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Leave a critical section; the calling thread may once again be suspended.

void *thd_single_enter*(void):

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Enter a critical region where all other suspendible threads must be suspended.

void *thd_single_leave*(void):

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Leave a critical section where all other threads must be suspended. All threads that were suspended in *thd_single_enter*() are resumed.

void *thd_suspend*(cw_thd_t *a_thd):

Input(s):

a_thd: Pointer to a *thd*.

Output(s): None.

Exception(s): None.

Description: Suspend *a_thd*.

cw_bool_t *thd_trysuspend*(cw_thd_t *a_thd):

Input(s):

a_thd: Pointer to a *thd*.

Output(s):

retval:

FALSE: Success.

TRUE: Failure.

Exception(s): None.

Description: Try to suspend *a_thd*, but fail if it is in a critical section.

void *thd_resume*(cw_thd_t *a_thd):

Input(s):

a_thd: Pointer to a *thd*.

Output(s): None.

Exception(s): None.

Description: Resume (make runnable) *a_thd*.

4.10.37 tsd

The *tsd* class implements thread-specific data. A *tsd* instance can be created, then any number of threads can use that same instance to store and retrieve a thread-specific pointer to data.

API

void *tsd_new*(cw_tsd_t *a_tsd, void (*a_func)(void *)):

Input(s):

a_tsd: Pointer to space for a *tsd*.

a_func: Pointer to a cleanup function, or NULL.

Output(s): None.

Exception(s): None.

Description: Constructor.

void *tsd_delete*(cw_tsd_t *a_tsd):

Input(s):

a_tsd: Pointer to a *tsd*.

Output(s): None.

Exception(s): None.

Description: Destructor.

void * *tsd_get*(cw_tsd_t *a_tsd):

Input(s):

a_tsd: Pointer to a *tsd*.

Output(s):

retval: Pointer to thread-specific data.

Exception(s): None.

Description: Get thread-specific data pointer.

void *tsd_set*(cw_tsd_t *a_tsd, void *a_val):

Input(s):

a_tsd: Pointer to a *tsd*.

a_val: Pointer to thread-specific data.

Output(s): None.

Exception(s): None.

Description: Set thread-specific data pointer.

4.10.38 xep

The *xep* class implements exception handling, with support for *xep_try* and *xep_catch*() blocks. Minimal use must include at least:

```
xep_begin();
xep_try
{
    /* Code that might throw an exception. */
}
xep_end();
```

A more complete skeleton looks like:

```
xep_begin();
xep_try
{
    /* Code that might throw an exception. */
}
xep_catch(SOME_EXCEPTION)
{
    /* Handle exception... */
    xep_handled();
}
xep_catch(ANOTHER_EXCEPTION)
xep_mcatch(YET_ANOTHER)
{
    /* React to exception, but propagate... */
}
xep_acatch
{
    /* Handle all exceptions not explicitly handled above... */
    xep_handled();
}
xep_end();
```

Note that there is some serious cpp macro magic behind the *xep* interface, and as such, if usage deviates significantly from the above templates, compiler errors may result.

Exception values are of type cw_xepv_t. *CW_ONYXX_MIN* to *CW_ONYXX_MAX* are reserved by *libonyx*, and other ranges may be reserved by other libraries. See their documentation for details.

An exception is not implicitly handled if an exception handler is executed for that exception. Instead, *xep_handled*() must be manually called to avoid propagating the exception up the handler chain.

It is not legal to return from a function within an exception handling code block, nor is it legal to jump out of an exception handling block; doing so will corrupt the exception handler chain.

API

void *xep_begin*(void):

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Begin an exception handling code block.

void *xep_end*(void):

Input(s): None.

Output(s): None.

Exception(s): None.

Description: End an exception handling block.

***xep_try* ...:**

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Begin a block of code that is to be executed, with the possibility that an exception might be thrown.

***xep_catch*(cw_xepv_t a_xepv) ...:**

Input(s):

a_xepv: Exception number.

Output(s): None.

Exception(s): None.

Description: Begin a block of code that catches an exception. The exception is not considered handled unless *xep_handled*() is called.

***xep_mcatch*(cw_xepv_t a_xepv) ...:**

Input(s):

a_xepv: Exception number.

Output(s): None.

Exception(s): None.

Description: Begin a block of code that catches an exception. Must immediately follow a *xep_catch*() call. This interface is used for the case where more than one exception type is to be handled by the same code block. The exception is not considered handled unless *xep_handled*() is called.

***xep_acatch* ...:**

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Begin a block of code that catches all exceptions not explicitly caught by *xep_catch*() and *xep_mcatch*() blocks. There may only be one *xep_acatch* block within a try/catch block. The exception is not considered handled unless *xep_handled*() is called.

cw_xepv_t *xep_value*(void):

Input(s): None.

Output(s):

retval: Value of the current exception being handled.

Exception(s): None.

Description: Return the value of the current exception being handled.

const char * *xep_filename*(void):

Input(s): None.

Output(s):

retval: Filename where the current exception being handled was thrown.

Exception(s): None.

Description: Return the filename where the current exception being handled was thrown.

cw_uint32_t *xep_line_num*(void):

Input(s): None.

Output(s):

retval: Line number where the current exception being handled was thrown.

Exception(s): None.

Description: Return the line number where the current exception being handled was thrown.

void *xep_throw_e*(cw_xepv_t a_xepv, const char *a_filename, cw_uint32_t a_line_num):
void *xep_throw*(cw_xepv_t a_xepv):

Input(s):

a_xepv: Exception number to throw.

a_filename: Should be *__FILE__*.

a_line_num: Should be *__LINE__*.

Output(s): None.

Exception(s):

a_xepv.

Description: Throw an exception.

void *xep_retry*(void):

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Implicitly handle the current exception and retry the *xep_try* code block.

void *xep_handled*(void):

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Mark the current exception as handled.

4.11 Dictionaries

4.11.1 gcdict

The *gcdict* functions implement the operators contained in **gcdict**. Only the C API is documented here; see Section 2.10.4 for operator semantics.

API

void *gcdict_active*(cw_nxo_t *a_thread):
void *gcdict_collect*(cw_nxo_t *a_thread):
void *gcdict_period*(cw_nxo_t *a_thread):
void *gcdict_setactive*(cw_nxo_t *a_thread):
void *gcdict_setperiod*(cw_nxo_t *a_thread):
void *gcdict_setthreshold*(cw_nxo_t *a_thread):
void *gcdict_stats*(cw_nxo_t *a_thread):
void *gcdict_threshold*(cw_nxo_t *a_thread):

Input(s):

a_thread: Pointer to a thread.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: C interfaces to Onyx operators that control garbage collection.

4.11.2 systemdict

The *systemdict* functions implement the operators contained in **systemdict**. Only the C API is documented here; see Section 2.10.9 for operator semantics.

API

void *systemdict_abs*(cw_nxo_t *a_thread):
void *systemdict_accept*(cw_nxo_t *a_thread):
void *systemdict_acos*(cw_nxo_t *a_thread):

void *systemdict_acosh*(cw_nxo_t *a_thread):
void *systemdict_add*(cw_nxo_t *a_thread):
void *systemdict_adn*(cw_nxo_t *a_thread):
void *systemdict_and*(cw_nxo_t *a_thread):
void *systemdict_array*(cw_nxo_t *a_thread):
void *systemdict_asin*(cw_nxo_t *a_thread):
void *systemdict_asinh*(cw_nxo_t *a_thread):
void *systemdict_atan*(cw_nxo_t *a_thread):
void *systemdict_atan2*(cw_nxo_t *a_thread):
void *systemdict_atanh*(cw_nxo_t *a_thread):
void *systemdict_aup*(cw_nxo_t *a_thread):
void *systemdict_bdup*(cw_nxo_t *a_thread):
void *systemdict_begin*(cw_nxo_t *a_thread):
void *systemdict_bind*(cw_nxo_t *a_thread):
void *systemdict_bindsocket*(cw_nxo_t *a_thread):
void *systemdict_bpop*(cw_nxo_t *a_thread):
void *systemdict_broadcast*(cw_nxo_t *a_thread):
void *systemdict_bytesavailable*(cw_nxo_t *a_thread):
void *systemdict_cat*(cw_nxo_t *a_thread):
void *systemdict_cd*(cw_nxo_t *a_thread):
void *systemdict_ceiling*(cw_nxo_t *a_thread):
void *systemdict_chmod*(cw_nxo_t *a_thread):
void *systemdict_chown*(cw_nxo_t *a_thread):
void *systemdict_chroot*(cw_nxo_t *a_thread):
void *systemdict_clear*(cw_nxo_t *a_thread):
void *systemdict_cleartomark*(cw_nxo_t *a_thread):
void *systemdict_close*(cw_nxo_t *a_thread):
void *systemdict_condition*(cw_nxo_t *a_thread):
void *systemdict_connect*(cw_nxo_t *a_thread):
void *systemdict_copy*(cw_nxo_t *a_thread):
void *systemdict_cos*(cw_nxo_t *a_thread):
void *systemdict_cosh*(cw_nxo_t *a_thread):
void *systemdict_count*(cw_nxo_t *a_thread):
void *systemdict_countdstack*(cw_nxo_t *a_thread):
void *systemdict_countestack*(cw_nxo_t *a_thread):
void *systemdict_counttomark*(cw_nxo_t *a_thread):
void *systemdict_currentdict*(cw_nxo_t *a_thread):
void *systemdict_currentlocking*(cw_nxo_t *a_thread):
void *systemdict_cvds*(cw_nxo_t *a_thread):
void *systemdict_cve*(cw_nxo_t *a_thread):
void *systemdict_cves*(cw_nxo_t *a_thread):
void *systemdict_cvlit*(cw_nxo_t *a_thread):
void *systemdict_cvn*(cw_nxo_t *a_thread):
void *systemdict_cvrs*(cw_nxo_t *a_thread):
void *systemdict_cvs*(cw_nxo_t *a_thread):
void *systemdict_cvx*(cw_nxo_t *a_thread):
void *systemdict_dec*(cw_nxo_t *a_thread):
void *systemdict_def*(cw_nxo_t *a_thread):
void *systemdict_detach*(cw_nxo_t *a_thread):
void *systemdict_dict*(cw_nxo_t *a_thread):
void *systemdict_die*(cw_nxo_t *a_thread):

void *systemdict_dirforeach*(cw_nxo_t *a_thread):
void *systemdict_div*(cw_nxo_t *a_thread):
void *systemdict_dn*(cw_nxo_t *a_thread):
void *systemdict_dstack*(cw_nxo_t *a_thread):
void *systemdict_dup*(cw_nxo_t *a_thread):
void *systemdict_echeck*(cw_nxo_t *a_thread):
void *systemdict_egid*(cw_nxo_t *a_thread):
void *systemdict_end*(cw_nxo_t *a_thread):
void *systemdict_eq*(cw_nxo_t *a_thread):
void *systemdict_estack*(cw_nxo_t *a_thread):
void *systemdict_euid*(cw_nxo_t *a_thread):
void *systemdict_eval*(cw_nxo_t *a_thread):
void *systemdict_exch*(cw_nxo_t *a_thread):
void *systemdict_exec*(cw_nxo_t *a_thread):
void *systemdict_exit*(cw_nxo_t *a_thread):
void *systemdict_exp*(cw_nxo_t *a_thread):
void *systemdict_floor*(cw_nxo_t *a_thread):
void *systemdict_flush*(cw_nxo_t *a_thread):
void *systemdict_flushfile*(cw_nxo_t *a_thread):
void *systemdict_for*(cw_nxo_t *a_thread):
void *systemdict_foreach*(cw_nxo_t *a_thread):
void *systemdict_forkexec*(cw_nxo_t *a_thread):
void *systemdict_ge*(cw_nxo_t *a_thread):
void *systemdict_get*(cw_nxo_t *a_thread):
void *systemdict_getinterval*(cw_nxo_t *a_thread):
void *systemdict_gid*(cw_nxo_t *a_thread):
void *systemdict_gstderr*(cw_nxo_t *a_thread):
void *systemdict_gstdin*(cw_nxo_t *a_thread):
void *systemdict_gstdout*(cw_nxo_t *a_thread):
void *systemdict_gt*(cw_nxo_t *a_thread):
void *systemdict_hooktag*(cw_nxo_t *a_thread):
void *systemdict_ibdup*(cw_nxo_t *a_thread):
void *systemdict_ibpop*(cw_nxo_t *a_thread):
void *systemdict_idiv*(cw_nxo_t *a_thread):
void *systemdict_idup*(cw_nxo_t *a_thread):
void *systemdict_if*(cw_nxo_t *a_thread):
void *systemdict_ifelse*(cw_nxo_t *a_thread):
void *systemdict_inc*(cw_nxo_t *a_thread):
void *systemdict_iobuf*(cw_nxo_t *a_thread):
void *systemdict_ipop*(cw_nxo_t *a_thread):
void *systemdict_istack*(cw_nxo_t *a_thread):
void *systemdict_join*(cw_nxo_t *a_thread):
void *systemdict_known*(cw_nxo_t *a_thread):
void *systemdict_lcheck*(cw_nxo_t *a_thread):
void *systemdict_le*(cw_nxo_t *a_thread):
void *systemdict_length*(cw_nxo_t *a_thread):
void *systemdict_link*(cw_nxo_t *a_thread):
void *systemdict_listen*(cw_nxo_t *a_thread):
void *systemdict_ln*(cw_nxo_t *a_thread):
void *systemdict_load*(cw_nxo_t *a_thread):
void *systemdict_localtime*(cw_nxo_t *a_thread):

void *systemdict_lock*(cw_nxo_t *a_thread):
void *systemdict_log*(cw_nxo_t *a_thread):
void *systemdict_loop*(cw_nxo_t *a_thread):
void *systemdict_lt*(cw_nxo_t *a_thread):
void *systemdict_match*(cw_nxo_t *a_thread):
void *systemdict_mkdir*(cw_nxo_t *a_thread):
void *systemdict_mkfifo*(cw_nxo_t *a_thread):
void *systemdict_mod*(cw_nxo_t *a_thread):
void *systemdict_modload*(cw_nxo_t *a_thread):
void *systemdict_monitor*(cw_nxo_t *a_thread):
void *systemdict_mul*(cw_nxo_t *a_thread):
void *systemdict_mutex*(cw_nxo_t *a_thread):
void *systemdict_nbpop*(cw_nxo_t *a_thread):
void *systemdict_ncat*(cw_nxo_t *a_thread):
void *systemdict_ndn*(cw_nxo_t *a_thread):
void *systemdict_ndup*(cw_nxo_t *a_thread):
void *systemdict_ne*(cw_nxo_t *a_thread):
void *systemdict_neg*(cw_nxo_t *a_thread):
void *systemdict_nip*(cw_nxo_t *a_thread):
void *systemdict_nonblocking*(cw_nxo_t *a_thread):
void *systemdict_not*(cw_nxo_t *a_thread):
void *systemdict_npop*(cw_nxo_t *a_thread):
void *systemdict_nsleep*(cw_nxo_t *a_thread):
void *systemdict_nup*(cw_nxo_t *a_thread):
void *systemdict_offset*(cw_nxo_t *a_thread):
void *systemdict_open*(cw_nxo_t *a_thread):
void *systemdict_or*(cw_nxo_t *a_thread):
void *systemdict_ostack*(cw_nxo_t *a_thread):
void *systemdict_over*(cw_nxo_t *a_thread):
void *systemdict_peername*(cw_nxo_t *a_thread):
void *systemdict_pid*(cw_nxo_t *a_thread):
void *systemdict_pipe*(cw_nxo_t *a_thread):
void *systemdict_poll*(cw_nxo_t *a_thread):
void *systemdict_pop*(cw_nxo_t *a_thread):
void *systemdict_pow*(cw_nxo_t *a_thread):
void *systemdict_ppid*(cw_nxo_t *a_thread):
void *systemdict_print*(cw_nxo_t *a_thread):
void *systemdict_put*(cw_nxo_t *a_thread):
void *systemdict_putinterval*(cw_nxo_t *a_thread):
void *systemdict_pwd*(cw_nxo_t *a_thread):
void *systemdict_quit*(cw_nxo_t *a_thread):
void *systemdict_rand*(cw_nxo_t *a_thread):
void *systemdict_read*(cw_nxo_t *a_thread):
void *systemdict_readline*(cw_nxo_t *a_thread):
void *systemdict_readlink*(cw_nxo_t *a_thread):
void *systemdict_realtime*(cw_nxo_t *a_thread):
void *systemdict_recv*(cw_nxo_t *a_thread):
void *systemdict_regex*(cw_nxo_t *a_thread):
void *systemdict_regsub*(cw_nxo_t *a_thread):
void *systemdict_rename*(cw_nxo_t *a_thread):
void *systemdict_repeat*(cw_nxo_t *a_thread):

void *systemdict_rmdir*(cw_nxo_t *a_thread):
void *systemdict_roll*(cw_nxo_t *a_thread):
void *systemdict_rot*(cw_nxo_t *a_thread):
void *systemdict_round*(cw_nxo_t *a_thread):
void *systemdict_sadn*(cw_nxo_t *a_thread):
void *systemdict_saup*(cw_nxo_t *a_thread):
void *systemdict_sbdup*(cw_nxo_t *a_thread):
void *systemdict_sbpop*(cw_nxo_t *a_thread):
void *systemdict_sbpush*(cw_nxo_t *a_thread):
void *systemdict_sclear*(cw_nxo_t *a_thread):
void *systemdict_scleartomark*(cw_nxo_t *a_thread):
void *systemdict_scount*(cw_nxo_t *a_thread):
void *systemdict_scounttomark*(cw_nxo_t *a_thread):
void *systemdict_sdn*(cw_nxo_t *a_thread):
void *systemdict_sdup*(cw_nxo_t *a_thread):
void *systemdict_seek*(cw_nxo_t *a_thread):
void *systemdict_self*(cw_nxo_t *a_thread):
void *systemdict_send*(cw_nxo_t *a_thread):
void *systemdict_serviceport*(cw_nxo_t *a_thread):
void *systemdict_setegid*(cw_nxo_t *a_thread):
void *systemdict_setenv*(cw_nxo_t *a_thread):
void *systemdict_seteuid*(cw_nxo_t *a_thread):
void *systemdict_setgid*(cw_nxo_t *a_thread):
void *systemdict_setgstderr*(cw_nxo_t *a_thread):
void *systemdict_setgstdin*(cw_nxo_t *a_thread):
void *systemdict_setgstdout*(cw_nxo_t *a_thread):
void *systemdict_setiobuf*(cw_nxo_t *a_thread):
void *systemdict_setlocking*(cw_nxo_t *a_thread):
void *systemdict_setnonblocking*(cw_nxo_t *a_thread):
void *systemdict_setsockopt*(cw_nxo_t *a_thread):
void *systemdict_setstderr*(cw_nxo_t *a_thread):
void *systemdict_setstdin*(cw_nxo_t *a_thread):
void *systemdict_setstdout*(cw_nxo_t *a_thread):
void *systemdict_setuid*(cw_nxo_t *a_thread):
void *systemdict_sexch*(cw_nxo_t *a_thread):
void *systemdict_shift*(cw_nxo_t *a_thread):
void *systemdict_sibdup*(cw_nxo_t *a_thread):
void *systemdict_sibpop*(cw_nxo_t *a_thread):
void *systemdict_sidup*(cw_nxo_t *a_thread):
void *systemdict_signal*(cw_nxo_t *a_thread):
void *systemdict_sin*(cw_nxo_t *a_thread):
void *systemdict_sinh*(cw_nxo_t *a_thread):
void *systemdict_sipop*(cw_nxo_t *a_thread):
void *systemdict_snbpop*(cw_nxo_t *a_thread):
void *systemdict_sndn*(cw_nxo_t *a_thread):
void *systemdict_sndup*(cw_nxo_t *a_thread):
void *systemdict_snip*(cw_nxo_t *a_thread):
void *systemdict_snpop*(cw_nxo_t *a_thread):
void *systemdict_snup*(cw_nxo_t *a_thread):
void *systemdict_socket*(cw_nxo_t *a_thread):
void *systemdict_socketpair*(cw_nxo_t *a_thread):

void systemdict_sockname(cw_nxo_t *a_thread):
void systemdict_sockopt(cw_nxo_t *a_thread):
void systemdict_sover(cw_nxo_t *a_thread):
void systemdict_split(cw_nxo_t *a_thread):
void systemdict_spop(cw_nxo_t *a_thread):
void systemdict_spush(cw_nxo_t *a_thread):
void systemdict_sqrt(cw_nxo_t *a_thread):
void systemdict_srand(cw_nxo_t *a_thread):
void systemdict_sroll(cw_nxo_t *a_thread):
void systemdict_srot(cw_nxo_t *a_thread):
void systemdict_stack(cw_nxo_t *a_thread):
void systemdict_start(cw_nxo_t *a_thread):
void systemdict_status(cw_nxo_t *a_thread):
void systemdict_stderr(cw_nxo_t *a_thread):
void systemdict_stdin(cw_nxo_t *a_thread):
void systemdict_stdout(cw_nxo_t *a_thread):
void systemdict_stop(cw_nxo_t *a_thread):
void systemdict_stopped(cw_nxo_t *a_thread):
void systemdict_string(cw_nxo_t *a_thread):
void systemdict_stuck(cw_nxo_t *a_thread):
void systemdict_sub(cw_nxo_t *a_thread):
void systemdict_submatch(cw_nxo_t *a_thread):
void systemdict_subst(cw_nxo_t *a_thread):
void systemdict_sunder(cw_nxo_t *a_thread):
void systemdict_sup(cw_nxo_t *a_thread):
void systemdict_sym_lp(cw_nxo_t *a_thread) (“(”):
void systemdict_sym_rp(cw_nxo_t *a_thread) (“)”):
void systemdict_sym_gt(cw_nxo_t *a_thread) (“>”):
void systemdict_sym_rb(cw_nxo_t *a_thread) (“]”):
void systemdict_symlink(cw_nxo_t *a_thread):
void systemdict_tan(cw_nxo_t *a_thread):
void systemdict_tanh(cw_nxo_t *a_thread):
void systemdict_tell(cw_nxo_t *a_thread):
void systemdict_test(cw_nxo_t *a_thread):
void systemdict_thread(cw_nxo_t *a_thread):
void systemdict_threadsdict(cw_nxo_t *a_thread):
void systemdict_threaddstack(cw_nxo_t *a_thread):
void systemdict_threadestack(cw_nxo_t *a_thread):
void systemdict_threadistack(cw_nxo_t *a_thread):
void systemdict_threadostack(cw_nxo_t *a_thread):
void systemdict_timedwait(cw_nxo_t *a_thread):
void systemdict_token(cw_nxo_t *a_thread):
void systemdict_trunc(cw_nxo_t *a_thread):
void systemdict_truncate(cw_nxo_t *a_thread):
void systemdict_trylock(cw_nxo_t *a_thread):
void systemdict_tuck(cw_nxo_t *a_thread):
void systemdict_type(cw_nxo_t *a_thread):
void systemdict_uid(cw_nxo_t *a_thread):
void systemdict_umask(cw_nxo_t *a_thread):
void systemdict_undef(cw_nxo_t *a_thread):
void systemdict_under(cw_nxo_t *a_thread):

void *systemdict_unless*(cw_nxo_t *a_thread):
void *systemdict_unlink*(cw_nxo_t *a_thread):
void *systemdict_unlock*(cw_nxo_t *a_thread):
void *systemdict_unsetenv*(cw_nxo_t *a_thread):
void *systemdict_until*(cw_nxo_t *a_thread):
void *systemdict_up*(cw_nxo_t *a_thread):
void *systemdict_wait*(cw_nxo_t *a_thread):
void *systemdict_waitpid*(cw_nxo_t *a_thread):
void *systemdict_where*(cw_nxo_t *a_thread):
void *systemdict_while*(cw_nxo_t *a_thread):
void *systemdict_write*(cw_nxo_t *a_thread):
void *systemdict_xcheck*(cw_nxo_t *a_thread):
void *systemdict_xor*(cw_nxo_t *a_thread):
void *systemdict_yield*(cw_nxo_t *a_thread):

Input(s):

a_thread: Pointer to a thread.

Output(s): None.

Exception(s):

CW_ONYXX_OOM.

Description: C interfaces to onyx operators.

Index