Onyx Manual, Version 2.0.0

Jason Evans

August 21, 2001

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.

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

Contents

List of Tables

Chapter 1

Onyx Language Reference

Onyx is a stack-based, threaded, interpreted language. Its closest relative is Adobe PostScript, 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 a mechanism for binding procedures, which makes interpreter performance approximately the same 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.
- 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.

1.1 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 1.1 for object type classifications.

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

Table 1.1: 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 1.2 enumerates under what circumstances object interpretation results in execution. Table 1.3 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:

```
[1 2 3]
```

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
stack	data	data	data
string	data	code	code
thread	data	data	data

Table 1.2: Interpretation of objects by type and attribute

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.

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.

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
stack	data	data	data
string	data	code	code
thread	data	data	data

Table 1.3: Evaluation of objects by type and attribute

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.

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.

1.2 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> | <name> | <string>

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

<**name**> **:** Any token that cannot be interpreted as a number or a string is interpreted as an executable name. There are three syntaxes for names: executable, literal and immediately evaluated. Executable names are looked up in the dictionary stack and executed (unless execution is deferred). 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    % literal
//biz   % 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.

n Newline.

r Carriage return.

t Tab.

b Backspace.

f Formfeed.

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

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 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
```

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')
```

1.3 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 manipulated via the **begin** and **end** operators. The initial dictionaries on the dictionary stack cannot be removed.

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.

1.4 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
```

1.5 Error handling

The error handling mechanisms in Onyx are simple but flexible. When an error occurs, the following operations are performed as a result of executing **throw**:

1. Store a snapshot of the thread state in the currenterror dictionary.
2. Push the object whose execution caused the error onto the operand stack.
3. Find an error handler in errordict corresponding to the current error.
4. Execute the error handler. The standard error handlers in turn execute errordict's **handleerror**.
5. Execute errordict's stop operator.

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**.

1.6 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.

1.6.1 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.

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.

1.6.2 Explicit synchronization

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

1.7 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 1.8.4.

1.8 Dictionary reference

All operators built in to Onyx have corresponding names that are composed entirely of lower case letters. 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 in names (for example, capital letters, numbers, underscore, etc.).

1.8.1 currenterror

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

Table 1.4: 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.
–	**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. The name is usually one of the errors defined in errordict.

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>
```

1.8.2 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*.

1.8.3 errordict

Each thread has its own errordict, which is used by default by the error handling machinery. **throw** calls the error handlers, which call errordict's **handleerror**, after which **throw** calls errordict's **stop**. Any of the names in errordict can be redefined to suit the thread's or application's error handling requirements.

Table 1.5: errordict summary by functional group

Input(s)	Op/Proc/Var	Output(s)	Description
Control operators			
–	**handleerror**	–	Print a state dump.
–	**stop**	–	Last operation during error handling.
Error operators			
–	**dstackunderflow**	–	Dictionary stack underflow.
–	**estackoverflow**	–	Maximum recursion depth exceeded.
–	**invalidaccess**	–	Permission error.
–	**invalidexit**	–	Exit outside of any loop.
–	**invalidfileaccess**	–	Insufficient file permissions.
–	**ioerror**	–	I/O error.
–	**limitcheck**	–	Value outside of legal range.
–	**rangecheck**	–	Out of bounds string or array access.
–	**stackunderflow**	–	Not enough objects on ostack.
–	**syntaxerror**	–	Scanner syntax error.
–	**typecheck**	–	Incorrect argument type.
–	**undefined**	–	Name not defined in dstack.
–	**undefinedfilename**	–	Bad filename.
–	**undefinedresult**	–	Divide by 0.
–	**unmatchedfino**	–	No fino on ostack.
–	**unmatchedmark**	–	No mark on ostack.
–	**unregistered**	–	Non-enumerated error.

– dstackunderflow –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: An attempt was made to remove one of the initial dictionaries from dstack.

– estackoverflow –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Maximum interpreter recursion was exceeded.

– 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>
```

– invalidaccess –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Permission error.

– invalidexit –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Exit 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 –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Insufficient file permissions.

– ioerror –:

Input(s): None.

Output(s): None.

Errors(s): None.

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

– limitcheck –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Value outside of legal range.

– rangecheck –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Out of bounds string or array access.

– stackunderflow –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Not enough objects on ostack.

– 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
```

– syntaxerror –:

Input(s): None.
Output(s): None.
Errors(s): None.
Description: Scanner syntax error.

– typecheck –:

Input(s): None.
Output(s): None.
Errors(s): None.
Description: Incorrect argument type.

– undefined –:

Input(s): None.
Output(s): None.
Errors(s): None.
Description: Name not defined in any of the dictionaries on dstack.

– undefinedfilename –:

Input(s): None.
Output(s): None.
Errors(s): None.
Description: Bad filename.

– undefinedresult –:

Input(s): None.
Output(s): None.
Errors(s): None.
Description: Attempt to divide by 0.

– unmatchedfino –:

Input(s): None.
Output(s): None.
Errors(s): None.
Description: No fino on ostack.

– unmatchedmark –:

Input(s): None.
Output(s): None.
Errors(s): None.
Description: No mark on ostack.

– unregistered –:

Input(s): None.
Output(s): None.
Errors(s): None.
Description: Non-enumerated error.

1.8.4 gcdict

The gcdict dictionary provides garbage collection control and status capabilities.

Table 1.6: gcdict summary by functional group

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 befor 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>
```

1.8.5 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.

1.8.6 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.

In addition, the following 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.

Table 1.7: 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.
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.
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>
```

***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>
```

1.8.7 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, 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 index type /sprintsdict load exch get eval
} def
```

Table 1.8: 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.
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.
stack depth	**stacktype**	string	Create syntactical string from stack.
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>
```

***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>
```

1.8.8 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 1.9: systemdict summary by functional group

Input(s)	Op/Proc/Var	Output(s)	Description
Operand stack operators			
–	**mark**	mark	Create a mark.
–	**count**	count	Get the number of objects on ostack.
mark ...	**counttomark**	mark ... count	Get the depth of the topmost mark on ostack.
object	**dup**	object object	Duplicate an object.
objects count	**ndup**	objects objects	Duplicate objects.
object ... index	**index**	object ... object	Duplicate object on ostack at a given index.
a b	**exch**	b a	Exchange the top two objects on ostack.
region count amount	**roll**	*rolled*	Roll the top *count* objects up by *amount*.

Continued on next page...

Table 1.9: *continued*

Input(s)	Op/Proc/Var	Output(s)	Description
any	**pop**	–	Remove the top object from ostack.
objects count	**npop**	–	Remove objects from ostack.
objects	**clear**	–	Pop all objects off ostack.
mark ...	**cleartomark**	–	Remove objects from ostack through topmost mark.
–	**ostack**	stack	Get a current ostack snapshot.
Execution, control, and execution stack operators			
object	**eval**	–	Evaluate object.
boolean object	**if**	–	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.
proc	**loop**	–	Iterate 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	Scan for a token.
file/string	**token**	rem object true	
object	**start**	–	Evaluate object.
–	**quit**	–	Unwind to innermost start context.
object	**stopped**	boolean	Evaluate object.
–	**stop**	–	Unwind to innermost stopped or start context.
name	**throw**	object	Throw an error.
–	**estack**	stack	Get a current estack snapshot.
–	**countestack**	count	Get current estack depth.
–	**istack**	stack	Get a current istack snapshot.
status	**die**	–	Exit program.
–	**fork**	pid	Fork a new process.
args	**exec**	–	Overlay a new program and execute it.
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.

Continued on next page...

Table 1.9: *continued*

Input(s)	Op/Proc/Var	Output(s)	Description
–	**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.
nanoseconds	**nsleep**	–	Nanosleep.
Stack operators			
–	**(**	fino	Begin a stack declaration.
fino objects	**)**	stack	Create a stack.
–	**stack**	stack	Create a stack.
stack object	**spush**	–	Push an object onto a 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 index	**sindex**	–	Duplicate object in a stack at a given index.
stack	**sexch**	–	Exchange top objects on stack.
stack count amount	**sroll**	–	Roll objects on stack.
stack	**spop**	object	Pop an object off stack.
stack	**sclear**	–	Remove all objects on stack.
stack	**scleartomark**	–	Remove objects from stack down through topmost mark.
(a) (b)	**catenate**	(a b)	Catenate two stacks.
srcstack dststack	**copy**	dststack	Copy stack contents.
Integer and math operators			
a b	**add**	r	Add a and b.
a b	**sub**	r	Subtract b from a.
a b	**mul**	r	Multiply a and b.
a b	**div**	r	Divide a by b.
a b	**mod**	r	Mod a by b.
a b	**exp**	r	Raise a to the power of b.
a	**abs**	r	Get the absolute value of a.
a	**neg**	r	Get the negative of a.
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.
string index length	**getinterval**	substring	Get a string interval.
string index substring	**putinterval**	–	Copy substring into string.
'a' 'b'	**catenate**	'ab'	Catenate two strings.

Continued on next page...

Table 1.9: *continued*

Input(s)	Op/Proc/Var	Output(s)	Description
srcstring dststring	**copy**	dstsubstring	Copy string.
object depth	**sprints**	string	Create syntactical string from object.
object flags	**outputs**	string	Create formatted string from object.
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.
array	**length**	count	Get array length.
array index	**get**	object	Get array element.
array index object	**put**	–	Set array element.
array index length	**getinterval**	subarray	Get an array interval.
array index subarray	**putinterval**	–	Copy subarray into array.
[a] [b]	**catenate**	[a b]	Catenate two arrays.
srcarray dstarray	**copy**	dstsubarray	Copy array.
Dictionary and dictionary stack operators			
–	**gcdict**	dict	Get gcdict.
–	**globaldict**	dict	Get globaldict.
–	**sprintsdict**	dict	Get sprintsdict.
–	**outputsdict**	dict	Get outputsdict.
–	**envdict**	dict	Get envdict.
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 key	**where** **where**	false dict true	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.

Continued on next page...

Table 1.9: *continued*

Input(s)	Op/Proc/Var	Output(s)	Description
–	**countdstack**	count	Get number of stacks on dstack.
–	**cleardstack**	–	Pop all dstack elements pushed by **begin**.
File and filesystem operators			
filename flags	**open**	file	Open a file.
file	**close**	–	Close file.
file file string	**read** **read**	integer boolean string boolean	Read from file.
file	**readline**	string boolean	Read a line from file.
file	**bytesavailable**	count	Get number of buffered readable bytes.
file integer/string	**write**	–	Write to file.
string	**print**	–	Print string to stdout.
object depth	**sprint**	–	Syntactically print object to stdout.
object flags	**output**	–	Formatted print to stdout.
–	**pstack**	–	Syntactically print ostack elements.
file	**flushfile**	–	Flush file buffer.
–	**flush**	–	Flush stdout buffer.
file length	**truncate**	–	Truncate file.
file offset	**seek**	–	Move file position pointer.
file	**tell**	offset	Get file position pointer offset.
path mode	**mkdir**	–	Create a directory.
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.
file/filename	**status**	dict	Get file information.
path proc	**dirforeach**	–	Iterate on directory entries.
–	**pwd**	path	Get present working directory.
path	**cd**	–	Change present working directory.
–	**stdin**	file	Get stdin.
–	**stdout**	file	Get stdout.
–	**stderr**	file	Get stderr.
Logical and bitwise operators			
a b	**lt**	boolean	a less than b? (integer, string)
a b	**le**	boolean	a less than or equal to b? (integer, string)
a b	**eq**	boolean	a equal to b? (any type)
a b	**ne**	boolean	a not equal to b? (any type)

Continued on next page...

Table 1.9: *continued*

Input(s)	Op/Proc/Var	Output(s)	Description
a b	**ge**	boolean	a greater than or equal to b? (integer, string)
a b	**gt**	boolean	a greater than b? (integer, 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			
object	**type**	name	Get object type.
object	**echeck**	boolean	Evaluatable?
object	**xcheck**	boolean	Executable?
object	**cve**	object	Set evaluatable attribute.
object	**cvx**	object	Set executable attribute.
object	**cvlit**	object	Set literal attribute.
string	**cvn**	name	Convert string to name.
object	**cvs**	string	Convert object to string.
hook	**hooktag**	tag	Get hook tag.
integer radix	**cvrs**	string	Convert integer to radix string.
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.
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.

Continued on next page...

Table 1.9: *continued*

Input(s)	Op/Proc/Var	Output(s)	Description
object	**lcheck**	boolean	Implicitly locked?
Miscellaneous operators			
–	**#!**	mark	Begin interpreter magic.
mark names	**!#**	–	End interpreter magic.
–	**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.

***mark names* !# –:**

Input(s):

mark: A mark object.

names: Zero or more name objects.

Output(s): None.

Errors(s):

unmatchedmark.

Description: Remove mark and name objects constructed as a side effect of interpreter magic. This operator is an alias of cleartomark.

Example(s):

```
onyx:0> #!/usr/local/bin/onyx pstack
/onyx
/bin
/local
/usr
-mark-
onyx:5> !#
onyx:0>
```

– #! *mark*:

Input(s): None.

Output(s):

mark: A mark object.

Errors(s): None.

Description: Create a mark object in preparation for an interpreter path. This operator is an alias of mark.

Example(s):

```
onyx:0> #! pstack
-mark-
onyx:1>
```

– (*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.

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>
```

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

Input(s):

a: An integer.

b: An integer.

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>
```

***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>
```

***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 evaluatable array.

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>
```

***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]* catenate *[a b]*:**

***(a) (b)* catenate *(a b)*:**

***‘a’ ‘b’* catenate *‘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’] catenate
onyx:1> 1 sprint
[‘a’ ‘b’]
onyx:0> (‘a’) (‘b’) catenate
onyx:1> 1 sprint
(‘a’ ‘b’)
onyx:0> ‘a’ ‘b’ catenate
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>
```

***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>
```

***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>
```

– cleardstack –:

Input(s): None.

Output(s): None.

Errors(s): None.

Description: Pop all dictionaries on dstack that were pushed by **begin**.

Example(s):

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

```
(-dict- -dict- -dict- -dict-)
onyx:0>
```

***mark ...* cleartomark –:**

Input(s):

mark: A mark object.

...: Zero or more non-mark objects.

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>
```

***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>
```

– 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):

mark: A mark object.

...: Zero or more non-mark objects.

Output(s):

mark: The same mark that was passed in.

...: The same non-mark objects that were 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: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 1.6.1 for implicit synchronization details.

Example(s):

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

***object* cve *object*:**

Input(s):

object: An object.

Output(s):

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

Errors(s):

stackunderflow.

Description: Set the evaluatable attribute for *object*.

Example(s):

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

***object* cvlit *object*:**

Input(s):

object: An object.

Output(s):

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

Errors(s):

stackunderflow.

Description: Set the literal attribute for *object*.

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>
```

***object* cvs *string*:**

Input(s):

object: An object.

Output(s):

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

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-'`.

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

Other types: `‘--nostringval--’`.

Errors(s):

stackunderflow.

Description: Convert *object* 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> ‘foo\nbar\\biz\‘baz’ cvs 1 sprint
‘\‘foo\\nbar\\\\biz\\\‘baz\’’
onyx:0> mutex cvs 1 sprint
‘--nostringval--’
onyx:0>
```

***object* cvx *object*:**

Input(s):

object: An object.

Output(s):

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

Errors(s):

stackunderflow.

Description: Set the executable attribute for *object*.

Example(s):

```
onyx:0> [1 2 3] cvx 1 sprint
{1 2 3}
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*, 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’
‘ps’
‘pdf’
‘html’
onyx:0> pwd {‘Cookfile.inc’ search
     {pop ‘Yes: ’ print 1 sprint pop exit}
     {‘Not: ’ print 1 sprint} ifelse
} dirforeach
Not: ‘.’
Not: ‘..’
Not: ‘CVS’
Not: ‘.cvsignore’
Not: ‘Cookfile’
Yes: ‘Cookfile.inc’
onyx:0>
```

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

Input(s):

a: An integer.

b: A non-zero integer.

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
```

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

– 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>
```

***object* dup *object object*:**

Input(s):

object: An object.

Output(s):

object: The same object that was passed in.

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>
```

***object* echeck *boolean*:**

Input(s):

object: An object.

Output(s):

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

Errors(s):

stackunderflow.

Description: Check *object* 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):

dstackunderflow.

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 1.8.2 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: *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>
```

– 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>
```

***object* eval –:**

Input(s):

object: An object.

Output(s): None.

Errors(s):

stackunderflow.

Description: Evaluate *object*. See Section 1.1 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):

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**, **loop**, **foreach**, and **dirforeach**.

Example(s):

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

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

Input(s):

a: An integer.

b: A non-negative. integer.

Output(s):

r: *a* to the *b* power.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

Description: Return *a* to the *b* power.

Example(s):

```
onyx:0> 5 0 exp 1 sprint
1
onyx:0> 5 1 exp 1 sprint
5
onyx:0> 5 2 exp 1 sprint
25
```

```
onyx:0> -5 3 exp 1 sprint
-125
onyx:0> 5 -3 exp 1 sprint
Error /rangecheck
ostack: (5 -3)
dstack: (-dict- -dict- -dict- -dict-)
estack/istack trace (0..2):
0:      --exp--
1:      -file-
2:      --start--
onyx:3>
```

– 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>
```

– 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.

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>
```

– fork *pid*:

Input(s): None.

Output(s):

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

Errors(s):

limitcheck.

Description: Fork a new process. Care must be taken when using the fork operator due to the fact that onyx consumes programs on the fly. The child process cannot reliably scan onyx code from the same source as the parent, so the child process should be forked into an environment where it is executing an object that has already been constructed by the interpreter, which in turn avoids unwinding the onyx execution stack.

Example(s):

```
onyx:0> {fork dup 0 eq
        {pop ‘Child\n’ print flush}
        {‘Parent\n’ print flush waitpid}
        ifelse} eval
Parent
Child
onyx:0>
```

– gcdict *dict*:

Input(s): None.

Output(s):

dict: A dictionary.

Errors(s): None.

Description: Get gcdict. See Section 1.8.2 for details on envdict.

Example(s):

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

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

Input(s):

a: An integer or string.

b: The same type as *a*.

Output(s):

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

Errors(s):

stackunderflow.

typecheck.

Description: Compare two integers 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> ‘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 *object*:**

***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):

object: 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 1.8.2 for details on envdict.

Example(s):

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

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

Input(s):

a: An integer or string.

b: The same type as *a*.

Output(s):

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

Errors(s):

stackunderflow.

typecheck.

Description: Compare two integers or strings.

Example(s):

```
onyx:0> 1 1 gt 1 sprint
false
onyx:0> 2 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):

***boolean object* if –:**

Input(s):

boolean: A boolean.

object: An object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Evaluate *object* 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 1.1 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>
```

***object . . . index* index *object . . . object*:**

Input(s):

object: An object.

. . . : *index* objects.

index: Depth (count starts at 0, not counting *index*) of the object to duplicate on ostack.

Output(s):

object: The same object that was passed in.

. . . : The same *index* objects that were passed in.

Errors(s):

rangecheck.

stackunderflow.

typecheck.

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

Example(s):

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

– 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>
```

***object* lcheck *boolean*:**

Input(s):

object: An array, dict, file, or string.

Output(s):

boolean: True if *object* is implicitly locked, false otherwise.

Errors(s):

stackunderflow.

typecheck.

Description: Check if *object* 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: An integer or string.

b: The same type as *a*.

Output(s):

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

Errors(s):

stackunderflow.

typecheck.

Description: Compare two integers 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> '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>
```

***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>
```

***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>
```

***proc* loop –:**

Input(s):

proc: An object to evaluate.

Output(s): None.

Errors(s):

stackunderflow.

Description: Iteratively 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: An integer or string.

b: The same type as *a*.

Output(s):

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

Errors(s):

stackunderflow.

typecheck.

Description: Compare two integers or strings.

Example(s):

```
onyx:0> 1 2 lt 1 sprint
true
onyx:0> 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>
```

***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>
```

***a b* mod *r*:**

Input(s):

a: An integer.
b: A non-zero integer.

Output(s):

r: The modulus of *a* and *b*.

Errors(s):

stackunderflow.
typecheck.
undefinedresult.

Description: Return the modulus of *a* and *b*.

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>
```

***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>
```

***a b* mul *r*:**

Input(s):

a: An integer.

b: An integer.

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>
```

– 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* 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: *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>
```

***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>
```

***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* off 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>
```

***filename flags* 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.

Output(s):

file: A file object.

Errors(s):

invalidfileaccess.
ioerror.
limitcheck.
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>
```

***object depth* output –:**

Input(s):

object: An object to print syntactically.

depth: Maximum recursion depth.

Output(s): None.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Syntactically print *object*.

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 catenate </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>
```

***object flags* outputs *string*:**

Input(s):

object: An object to print syntactically.

depth: Formatting flags. See Section 1.8.6 for details on the supported flags.

Output(s):

string: A formatted string representation of *object*.

Errors(s):

stackunderflow.

typecheck.

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

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 catenate </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 1.8.6 for details on outputsdict.

Example(s):

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

– 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>
```

***any* pop –:**

Input(s):

any: Any object.

Output(s): None.

Errors(s):

stackunderflow.

Description: Remove the top object off ostack and discard it.

Example(s):

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

– 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 object* 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 *object* or *integer*, respectively.

key: An object to use as a key in *dict*.

object: 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>
```

– 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>
```

***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>
```

***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>
```

***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* 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>
```

***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>
```

***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. An 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>
```

***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 1.6.1 for implicit synchronization details.

Example(s):

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

***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>
```

***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>
```

***stack index* sindex –:**

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 sindex
onyx:1> 1 sprint
(3 2 1 0 2)
onyx:0>
```

***stack* spop *object*:**

Input(s):

stack: A stack object.

Output(s):

object: 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>
```

***object depth* sprint –:**

Input(s):

object: An object to print syntactically.

depth: Maximum recursion depth.

Output(s): None.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Syntactically print *object*.

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>
```

***object depth* sprints *string*:**

Input(s):

object: An object to print syntactically.

depth: Maximum recursion depth.

Output(s):

string: A syntactical string representation of *object*.

Errors(s):

stackunderflow.

typecheck.

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

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 1.8.7 for details on sprintsdict.

Example(s):

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

***stack object* spush –:**

Input(s):

stack: A stack object.

object: An object.

Output(s): None.

Errors(s):

stackunderflow.

typecheck.

Description: Push *object* onto *stack*.

Example(s):

```
onyx:0> () dup 1 spush
onyx:1> pstack
(1)
onyx:1>
```

***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 *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
()
```

***object* start –:**

Input(s):

object: An object.

Output(s): None.

Errors(s):

stackunderflow.

Description: Evaluate *object*. 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 stderr.

Errors(s): None.

Description: Get stdout.

Example(s):

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

– stdin *file*:

Input(s): None.

Output(s):

file: A file object corresponding to stdin.

Errors(s): None.

Description: Get stdin.

Example(s):

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

– stdout *file*:

Input(s): None.

Output(s):

file: A file object corresponding to stdout.

Errors(s): None.

Description: Get stdout.

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>
```

***object* stopped *boolean*:**

Input(s):

object: An object to evaluate.

Output(s):

boolean: True if stop operator was executed, false otherwise.

Errors(s):

invalidexit.

stackunderflow.

Description: Evaluate *object*. 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>
```

***a b* sub *r*:**

Input(s):

a: An integer.

b: An integer.

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>
```

***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):

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>
```

***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>
```

***name* throw *object*:**

Input(s):

name: The name of an error, which can be any of the following:

dstackunderflow.
estackoverflow.
invalidaccess.
invalidexit.
invalidfileaccess.
ioerror.
limitcheck.
rangecheck.
stackunderflow.
syntaxerror.
typecheck.
undefined.
undefinedfilename.
undefinedresult.

unmatchedmark.
unmatchedfino.
unregistered.

Output(s):

object: The object that was being executed when the error was thrown.

Errors(s):

stackunderflow.
typecheck.
undefined.

Description: Throw an error. This operator goes through 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. Evaluate the error handler in the errordict dictionary that corresponds to *name*.
6. Evaluate the errordict dictionary's **stop**.

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 index dup lock 1000000000 timedwait 1 sprint unlock
true
onyx:0>
```

***file*/*string* token *false*:**

***file*/*string* token *file*/*substring object 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.

object: 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>
```

***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>
```

***object* type *name*:**

Input(s):

object: An object.

Output(s):

name: An executable name that corresponds to the type of *object*:

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 *object*.

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>
```

***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>
```

***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>
```

– 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> {fork dup 0 eq
        {pop ‘Child\n’ print flush}
        {‘Parent\n’ print flush waitpid}
        ifelse} eval
Parent
Child
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>
```

***file integer/string* write –:**

Input(s):

file: A file object.

integer: An integer that represents an ascii character value.

string: A string object.

Output(s): None.

Errors(s):

ioerror.

stackunderflow.

typecheck.

Description: Write *integer* or *string* to *file*.

Example(s):

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

***object* xcheck *boolean*:**

Input(s):

object: An object.

Output(s):

boolean: True if *object* has the executable or evaluatable attribute, false otherwise.

Errors(s):

stackunderflow.

Description: Check *object* 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>
```

1.8.9 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 1.10: threaddict summary

Input(s)	Op/Proc/Var	Output(s)	Description
–	**threaddict**	dict	Get threaddict.

Continued on next page...

Table 1.10: *continued*

Input(s)	Op/Proc/Var	Output(s)	Description
–	**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 1.8.1 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 1.8.3 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 1.8.10 for details on userdict.

Errors(s): None.

Description: Get userdict.

Example(s):

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

1.8.10 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 2

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.

2.1 Usage

onyx -h

onyx -V

onyx -e <expr>

onyx <file> [<args>]

2.1.1 Options

-e <expr>: Execute <expr> as Onyx code.

-h: Display usage information and exit.

-V: Display the version number and exit.

2.2 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”.

ONYX_MPATH: A colon-separated list of additional paths that are searched by **mrequire** (documented in Section 2.3).

ONYX_RPATH: A colon-separated list of additional paths that are searched by **require** (documented in Section 2.3).

2.3 Language differences

If *onyx* is being run interactively:

- The name “stop” is redefined in the initial thread’s errordict to recursively execute 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.

If *onyx* is being run non-interactively:

- The name “stop” in errordict is redefined to call the die operator with an argument of 1.

In addition, the following definitions exist regardless of execution mode:

***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.

Example(s):

```
onyx:0> ‘/usr/local/share/canonware/onyx/nxmod/gtk.nxm’ ‘_cw_modgtk_init’
onyx:2> modload
onyx:0>
```

***module symbol* mrequire –:**

Input(s):

module: A string that represents a module filename. The module is searched for in the directories listed in the ONYX_MPATH environment variable.

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: Load a module using the ONYX_MPATH environment variable.

Example(s):

```
onyx:0> ‘gtk.nxm’ ‘_cw_modgtk_init’ mrequire
onyx:0>
```

***file* require –:**

Input(s):

file: A string that represents an Onyx source filename. The file is searched for in the directories listed in the ONYX_RPATH environment variable.

Output(s): None.

Errors(s):

stackunderflow.
typecheck.
undefinedfilename.

Description: Execute an Onyx source file using the ONYX_RPATH environment variable.

Example(s):

```
onyx:0> ‘ls.nx’ require flush
1001 1001 512 994886434000000000 .
1001 1001 512 994848629000000000 ..
1001 1001 512 994884381000000000 CVS
1001 1001 512 994884387000000000 bin
1001 1001 66 981033481000000000 .cvsignore
1001 1001 300 992069954000000000 COPYING
1001 1001 435 994831422000000000 ChangeLog
1001 1001 5084 994839465000000000 Cookfile.in
1001 1001 3633 993470163000000000 INSTALL
1001 1001 501 991968935000000000 README
1001 1001 3778 993470163000000000 aclocal.m4
1001 1001 26702 949579579000000000 config.guess
1001 1001 19814 949579579000000000 config.sub
1001 1001 101712 994840482000000000 configure
1001 1001 7812 994840477000000000 configure.in
1001 1001 5585 949579579000000000 install-sh
1001 1001 512 994845044000000000 doc
1001 1001 512 994754556000000000 lib
1001 1001 512 994830956000000000 mod
1001 1001 512 994884962000000000 test
1001 1001 3271 994884963000000000 config.log
1001 1001 3448 994884961000000000 config.cache
1001 1001 16934 994884961000000000 config.status
1001 1001 64838 994884988000000000 Cookfile.list
```

```
1001 1001 5985 994884961000000000 Cookfile
onyx:0>
```

Chapter 3

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.

3.1 Compilation

Use the following compiler command line to compile applications with *libonyx*.

```
cc <file> -lonyx -lpthread
```

3.2 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.

3.3 Global variables

libonyx defines the following global variables, which can be used by the application:

cw_g_mem: *mem* instance, default memory allocator.

3.4 Object instantiation

Many classes provide a means for external memory allocation, and in some cases for automatic cleanup during destruction. This feature enables improved performance and cache locality, but if misunderstood, can result in mysterious memory corruption and leaks.

3.5 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.

3.6 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).

3.7 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.

3.8 Integration issues

3.8.1 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.

3.8.2 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.

3.9 Guidlines 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 *nxo_thread_error*()), 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.

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.

3.10 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*, *out_std*, and *out_err*.

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_realloc_t*(void *a_arg, 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 memory range to be reallocated.
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_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_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 file descriptor 2.

Exception(s): None.

Description: If the expression evaluates to zero, print an error message to file descriptor 2 and *abort*().

Note: This macro is only active if the *_CW_ASSERT* cpp macro is defined.

void *_cw_not_reached*(void):

Input(s): None.

Output(s): Error printed to file descriptor 2.

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 file descriptor 2.

Exception(s): None.

Description: If *a_pointer* is NULL, print an error message to file descriptor 2 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 file descriptor 2.

Exception(s): None.

Description: Print the contents of *a_str*, followed by a carriage return, to file descriptor 2.

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.

3.11 Classes

3.11.1 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, the *mem* class 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_opaque_alloc_t *a_alloc, cw_opaque_dealloc_t *a_dealloc, void *a_arg, 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_alloc: Pointer to an allocation function to use internally.

a_dealloc: Pointer to a deallocation function to use internally.

a_arg: Opaque pointer to pass to *a_alloc*() and *a_dealloc*().

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.

3.11.2 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*.

3.11.3 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 more additional information.

API

***dch_new*(cw_dch_t *a_dch, cw_opaque_alloc_t *a_alloc, cw_opaque_dealloc_t *a_dealloc, void *a_arg, 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_alloc: Pointer to an allocation function to use internally.

a_dealloc: Pointer to a deallocation function to use internally.

a_arg: Opaque pointer to pass to *a_alloc*() and *a_dealloc*().

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*.

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*.

3.11.4 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 *out_err*.

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*().

API

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, 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_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.

3.11.5 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).

3.11.6 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*.

3.11.7 nx

The *nx* class encapsulates an Onyx interpreter instance. It contains a number of interpreter-global objects, as well as the garbage collector. Reclamation 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_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* .

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* .

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* .

3.11.8 nxa

The *nxa* class implements garbage collection. The garbage collector runs a separate thread that is controlled via an asynchronous message queue. The collector thread is only responsible for doing asynchronous collection due to allocation inactivity and all sweeping; all other marking is synchronously done in the thread context of the mutator that triggers collection.

API

void *nxa_new*(cw_nxa_t *a_nxa, cw_nx_t *a_nx):

Input(s):

a_nxa: Pointer to a *nxa*.

a_nx: Pointer to a *nx*.

Output(s): None.

Exception(s):

_CW_ONYXX_OOM.

Description: Constructor.

void *nxa_delete*(cw_nxa_t *a_nxa):

Input(s):

a_nxa: Pointer to a *nxa*.

Output(s): None.

Exception(s): None.

Description: Destructor.

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_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: Do a synchronous 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* .

3.11.9 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*.

3.11.10 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_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*(cw_nxo_t *a_a, 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 meaninful (integer, string), *a_a* is less than *a_b*.

0: *a_a* and *a_b* are equal.

1: For types which it is meaninful (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*(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_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*(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*(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*.

3.11.11 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*(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*(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*.

3.11.12 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*(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*.

3.11.13 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*.

3.11.14 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*(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*(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.

3.11.15 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):

Input(s):

a_nxo: Pointer to a file *nxo*.

a_fd: File descriptor number.

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_nxo_threade_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:

NXO_THREADE_NONE.
NXO_THREADE_IOERROR.
NXO_THREADE_INVALIDFILEACCESS.
NXO_THREADE_LIMITCHECK.

Exception(s): None.

Description: Open a file.

cw_nxo_threade_t *nxo_file_close*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a file *nxo*.

Output(s):

retval:

NXO_THREADE_NONE.
NXO_THREADE_IOERROR.

Exception(s): None.

Description: Close a file.

cw_sint32_t *nxo_file_fd_get*(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_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: NXO_THREADE_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_nxo_threade_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:
NXO_THREADE_NONE.
NXO_THREADE_IOERROR.
r_string: If *retval* is NXO_THREADE_NONE, 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_nxo_threade_t *nxo_file_write*(cw_nxo_t *a_nxo, const cw_uint8_t *a_str, cw_uint32_t a_len):

Input(s):
a_nxo: Pointer to a file *nxo*.
a_str: Pointer to data to write.
a_len: Length of *a_str*.

Output(s):
retval:
NXO_THREADE_NONE.
NXO_THREADE_IOERROR.

Exception(s): None.

Description: Write the *a_len* bytes of data pointed to *a_str*.

cw_nxo_threade_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:
NXO_THREADE_NONE.
NXO_THREADE_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: NXO_THREADE_IOERROR.
>= 0: Current file position.

Exception(s): None.

Description: Get the current file position.

cw_nxo_threade_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:
NXO_THREADE_NONE.
NXO_THREADE_IOERROR.

Exception(s): None.

Description: Move the current file position to *a_position*.

cw_uint32_t *nxo_file_buffer_size_get*(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*(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_nxo_threade_t *nxo_file_buffer_flush*(cw_nxo_t *a_nxo):

Input(s):

a_nxo: Pointer to a file *nxo*.

Output(s):

retval:

NXO_THREADE_NONE.

NXO_THREADE_IOERROR.

Exception(s): None.

Description: Flush any buffered write data to disk, and discard any buffered read data.

3.11.16 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.

3.11.17 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.

void *cw_nxo_hook_delete_t*(void *a_data, cw_nx_t *a_nx):

Input(s):

a_data: Opaque data pointer.

a_nx: Pointer to an *nx*.

Output(s): None.

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*(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*(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.

3.11.18 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*(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*.

3.11.19 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.

3.11.20 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*.

3.11.21 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*(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*(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*.

3.11.22 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.

3.11.23 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.

3.11.24 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*(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*.

3.11.25 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.

3.11.26 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_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_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_nxo_t * *nxo_stack_get*(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_nget*(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_down_get*(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_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*.

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*.

3.11.27 nxo_string

The *nxo_string* class is a subclass of the *nxo* class.

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*.

cw_uint32_t *nxo_string_len_get*(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*(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*(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*.

3.11.28 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.

Errors are of type cw_nxo_threade_t:

NXO_THREADE_NONE: No error.

NXO_THREADE_DSTACKUNDERFLOW: No poppable dictionary on dstack.

NXO_THREADE_ESTACKOVERFLOW: estack too deep.

NXO_THREADE_INTERRUPT: Interrupt.

NXO_THREADE_INVALIDACCESS: Permission error.

NXO_THREADE_INVALIDEXIT: **exit** operator called outside loop.

NXO_THREADE_INVALIDFILEACCESS: Insufficient file permissions.

NXO_THREADE_IOERROR: read()/write()/etc. error.

NXO_THREADE_LIMITCHECK: Value outside legal range.

NXO_THREADE_RANGECHECK: Out of bounds string or array use.

NXO_THREADE_STACKUNDERFLOW: Not enough objects on ostack.

NXO_THREADE_SYNTAXERROR: Scanner syntax error.

NXO_THREADE_TIMEOUT: Timeout.

NXO_THREADE_TYPECHECK: Incorrect argument type.

NXO_THREADE_UNDEFINED: Object not found in dstack.

NXO_THREADE_UNDEFINEDFILENAME: Bad filename.

NXO_THREADE_UNDEFINEDRESULT: Divide by 0.

NXO_THREADE_UNMATCHEDFINO: No fino on ostack.

NXO_THREADE_UNMATCHEDMARK: No mark on ostack.

NXO_THREADE_UNREGISTERED: Other non-enumerated error.

API

cw_nxn_t *nxo_threade_nxn*(cw_nxo_threade_t a_threade):

Input(s):
a_threade: A *threade* error code.

Output(s):
retval: An *nxn* corresponding to *a_threade*.

Exception(s): None.

Description: Return an *nxn* corresponding to *a_threade*.

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*(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*(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_SLASH_CONT: One '/' seen.
THREADTS_COMMENT: '%' seen, but no line break yet.
THREADTS_INTEGER: Scanning an integer.
THREADTS_INTEGER_RADIX: Scanning a radix integer.
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_HEX_CONT: '\' 'x' seen in a string.

THREADTS_STRING_HEX_FINISH: First hex digit of a "\xDD" string escape sequence 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 1.2 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_error*(cw_nxo_t *a_nxo, cw_nxo_threade_t a_threade):

Input(s):

a_nxo: Pointer to a thread *nxo*.

a_threade: An error code.

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*(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*.

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*.

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*.

3.11.29 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 ussed 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):

Input(s):

a_head: Pointer to a *ql* head.

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_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* 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_foreach_reverse*(<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.

3.11.30 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, <field_name> a_field):

Input(s):

a_qr_a: Pointer to a *qr*.

a_qr_b: Pointer to a *qr*.

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, <field_name> a_field):

Input(s):

a_qr_a: Pointer to a *qr*.

a_qr_b: Pointer to a *qr*.

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_foreach_reverse*(<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.

3.11.31 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.

3.11.32 thd

The *thd* class implements a wrapper around the system threads library (POSIX threads only, so far). In most regards, this is a thin wrapper around the normal threads functionality provided by the system, but some extra information is kept in order to allow implmentation of thread suspension/resumption, "critical sections", and "single sections".

In most cases, the additional functionality is implemented with the aid of signals. As a result, system calls may be interrupted by signals. The system calls will be automaticalaly 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*.

3.11.33 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.

***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.

3.11.34 xep

The *xep* class implements exception handling, with support for *xep_try*, *xep_catch*(), and *xep_finally* 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_finally {
        /* Execute after everything else. */
}
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. 0 to 127 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; 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.

***xep_finally ...*:**

Input(s): None.

Output(s): None.

Exception(s): None.

Description: Begin a block of code that is executed if no exceptions are thrown in the exception handling code block or if an exception handler is executed.

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.

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.

3.12 Dictionaries

3.12.1 gcdict

The *gcdict* functions implement the operators contained in *gcdict* . Only the C API is documented here; see Section 1.8.4 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.

3.12.2 systemdict

The *systemdict* functions implement the operators contained in *systemdict* . Only the C API is documented here; see Section 1.8.8 for operator semantics.

API

void *systemdict_abs*(cw_nxo_t *a_thread):
void *systemdict_add*(cw_nxo_t *a_thread):
void *systemdict_and*(cw_nxo_t *a_thread):
void *systemdict_array*(cw_nxo_t *a_thread):
void *systemdict_begin*(cw_nxo_t *a_thread):
void *systemdict_bind*(cw_nxo_t *a_thread):

void *systemdict_broadcast*(cw_nxo_t *a_thread):
void *systemdict_bytesavailable*(cw_nxo_t *a_thread):
void *systemdict_catenate*(cw_nxo_t *a_thread):
void *systemdict_cd*(cw_nxo_t *a_thread):
void *systemdict_chmod*(cw_nxo_t *a_thread):
void *systemdict_chown*(cw_nxo_t *a_thread):
void *systemdict_clear*(cw_nxo_t *a_thread):
void *systemdict_cleardstack*(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_copy*(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_cve*(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_def*(cw_nxo_t *a_thread):
void *systemdict_detach*(cw_nxo_t *a_thread):
void *systemdict_dict*(cw_nxo_t *a_thread):
void *systemdict_dirforeach*(cw_nxo_t *a_thread):
void *systemdict_div*(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_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_fork*(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_gt*(cw_nxo_t *a_thread):
void *systemdict_hooktag*(cw_nxo_t *a_thread):
void *systemdict_if*(cw_nxo_t *a_thread):
void *systemdict_ifelse*(cw_nxo_t *a_thread):
void *systemdict_index*(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_load*(cw_nxo_t *a_thread):
void *systemdict_lock*(cw_nxo_t *a_thread):
void *systemdict_loop*(cw_nxo_t *a_thread):
void *systemdict_lt*(cw_nxo_t *a_thread):
void *systemdict_mark*(cw_nxo_t *a_thread):
void *systemdict_mkdir*(cw_nxo_t *a_thread):
void *systemdict_mod*(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_ndup*(cw_nxo_t *a_thread):
void *systemdict_ne*(cw_nxo_t *a_thread):
void *systemdict_neg*(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_open*(cw_nxo_t *a_thread):
void *systemdict_or*(cw_nxo_t *a_thread):
void *systemdict_ostack*(cw_nxo_t *a_thread):
void *systemdict_pid*(cw_nxo_t *a_thread):
void *systemdict_pop*(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_realtime*(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_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_sdup*(cw_nxo_t *a_thread):

void *systemdict_seek*(cw_nxo_t *a_thread):
void *systemdict_self*(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_setlocking*(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_signal*(cw_nxo_t *a_thread):
void *systemdict_sindex*(cw_nxo_t *a_thread):
void *systemdict_spop*(cw_nxo_t *a_thread):
void *systemdict_sprint*(cw_nxo_t *a_thread):
void *systemdict_spush*(cw_nxo_t *a_thread):
void *systemdict_srand*(cw_nxo_t *a_thread):
void *systemdict_sroll*(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_sub*(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_tell*(cw_nxo_t *a_thread):
void *systemdict_test*(cw_nxo_t *a_thread):
void *systemdict_thread*(cw_nxo_t *a_thread):
void *systemdict_timedwait*(cw_nxo_t *a_thread):
void *systemdict_token*(cw_nxo_t *a_thread):
void *systemdict_truncate*(cw_nxo_t *a_thread):
void *systemdict_trylock*(cw_nxo_t *a_thread):
void *systemdict_type*(cw_nxo_t *a_thread):
void *systemdict_uid*(cw_nxo_t *a_thread):
void *systemdict_undef*(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_wait*(cw_nxo_t *a_thread):
void *systemdict_waitpid*(cw_nxo_t *a_thread):
void *systemdict_where*(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