Unit `tom`

TOM Standard Class Library and Runtime
edition *DRAFT* (unreleased), for `tom` version 1.0.
July 1998

Pieter J. Schoenmakers

Written by Pieter J. Schoenmakers `<tiggr@gerbil.org>`

1 Introduction

This manual documents unit `tom`, which is the TOM standard class library; it includes the TOM run-time library, `trt`.

Most of this manual was extracted from the sources of the unit `tom`. Unfortunately, the order of the classes and extensions in the printed manual is dictated by sorting the files making up unit `tom`. This is a restriction of `tm`, the TOM documentation extractor.

2 Unit `tom`

This chapter contains the documentation of the classes that make up unit `tom`. Each class starts at the top of a page; the instance description follows without a pagebreak.

class tom.All

The `All` class does not serve a purpose. It is the stateless `All` instance which is inherited by both the `State` class and instance, and, supposedly, by all objects not inheriting from `State`.

instance tom.All

Inherits

State supers: Conditions, Constants

variables

<h4>Paramount constants.</h4>

`const TRUE = !0;`
The boolean truth.

`const FALSE = !TRUE;`
The boolean non-truth.

`const YES = TRUE;`
An alternative name for `TRUE`.

`const NO = FALSE;`
An alternative name for `FALSE`.

methods

```
String
  description;
```

Return a string informally describing this object.

This returns the result of having the receiver `write` itself into a new `String` which is subsequently returned.

```
boolean (result)
  eq All other
post
  self == other == result;
```

The selector equivalent of '==', i.e. the returned `result` is `TRUE` iff the receiving object and the `other` object are the same object.

The postcondition states that this method is not overridable.

```
boolean (result)
  equal id other
post
  self == other -> result;
```

Return `TRUE` when the receiving object considers itself equal to the other object. For instance, two `Number` objects holding the same value will return `TRUE`.

The receiving object should be able to assume the `other` object is of the same kind, or at least shares with it a common superclass, as in the case of, for instance, `CharString` and `ByteString` which are both subclasses of `String` and can compare with each other.

As stated by the postcondition, an object must be `equal` to itself. This knowledge may be used by a caller to prevent a method invocation.

```
int
  hash;
```

Return a hash value for the receiving object. The default implementation returns some bit pattern deduced from `self`.

Two distinct objects considering themselves `equal` should also return the same `hash` value.

```
int
  hashq;
```

Hash the address of the receiving object. For classes not redefining `hash`, this performs the same function.

```
id (self)
  self;
```

Return the receiving object.

```
deferred OutputStream
  write OutputStream s;
```

All objects, even classes, know how to (descriptively) write themselves to a stream.

A default implementation of this method is provided by the `State` class and instance. <h4>Inheritance issues</h4>

```
deferred boolean
  classp;
```

Return `TRUE` iff the receiving object is a class object. An implementation for this method is provided by the `State` class and instance.

```
boolean
  isKindOf class (State) a_class;
```

Return `TRUE` iff the class of the receiving object is a subclass of the `class`.

```
deferred class (id)
  kind;
```

Return the class of the receiving object.

```
boolean
  respondsTo selector sel;
```

Return YES iff the selector sel can be safely sent to the receiver. The default implementation only checks whether the receiving object provides a direct implementation of the sel; any checking through an alternative forwardDelegate should be performed by the object itself. <h4>Forwarding and performing</h4>

```
All (self)
  forwardDelegate selector sel;
```

Return an object of which the method indicated by the selector sel should be invoked. This method is invoked if the receiving object does not directly respond to sel. The default implementation returns self. The object returned could be a delegate which is to act upon behalf of the receiving object for the intended call of the selector sel.

```
InvocationResult
  forwardInvocation Invocation invocation;
```

Return the result of forwarding the invocation, for example by firing it at an appropriate object. The default implementation raises a program-condition SelectorCondition.

```
dynamic
  perform selector sel
        : Array arguments = nil;
```

Send the receiving object a message with the selector sel and the, possibly unboxed, arguments. The number of elements of arguments must match the number of arguments dictated by the selector.

Unboxing the arguments means that if an int argument is needed, the int at int method will be used to retrieve the argument from the arguments, possibly resulting in the object retrieved being asked for its intValue.

```
dynamic
  perform selector sel
     with dynamic arguments;
```

Send the receiving object a message with the selector sel and the arguments. The number of arguments must match the number of arguments dictated by the selector.

If the selector sel accepts more than one argument, arguments should be a tuple. The tuple-ization of the actual arguments to the selector sel and the elements of the arguments tuple is ignored.

```
Thread
  performInThread selector sel
              with dynamic arguments;
```

Like perform with but create a new thread for the performance. Return the newly created thread or nil upon failure.

```
boolean
  invocationp;
```

Return YES iff the receiving object is an Invocation. Only Invocation objects are supposed to return YES. <h4>Object manipulation</h4>

```
dynamic
  valueOfVariableNamed ByteString name;
```

Retrieve the value of the variable with the indicated `name`. If there is more than one variable with the same name and expected return type, the first is returned.

```
void
         setValue dynamic value
  ofVariableNamed ByteString name;
```

Set the value of the variable named `name` in the receiving object to the `value`. It is an error if the type of the value does not exactly match the actual type of the variable: no conversion is performed.

```
int
  typeOfVariableNamed String name
                 from Extension ext
pre
  [[self stateExtensions] memq ext] != nil;
```

Return the type of the variable named `name` as introduced by the extension `ext`. This returns one of the `TYPEDESC_* Constants`.

```
Any
  valueOfVariableNamed String name
                  from Extension ext
pre
  [[self stateExtensions] memq ext] != nil;
```

Return the boxed value of the variable named `name` as introduced by the extension `ext`.

```
Extension
  extensionNamed String name
      inherited: boolean check_supers = NO;
```

Return the `Extension` object of this object for the extension named `name`. If `name == nil`, the main extension is returned.

```
Indexed
  stateExtensions;
```

Return an array of state introducing extensions of the receiving object. This includes the extensions introduced by superclasses.

<h4>Conditions and condition handling</h4>

```
void
  throw dynamic value;
```

Throw execution to the catch specified for the receiving object, returning the `value`. If the `value` is `void`, the default value for the type to be returned by the catch is returned.

```
void
  preconditionFailed selector sel;
```

This method is invoked for a failed precondition of a method invocation of the receiving object. The method is identified by the `selector sel`. The default implementation raises a `condition-condition` `SelectorCondition`.

Method precondition checking is enabled is the option `:cc-pre` is provided on the program's command line. The code for precondition checking is normally compiled in by the compiler. This code is omitted by passing the `-fno-checks` or `-fno-pre-checks` option to the compiler.

```
void
  postconditionFailed selector sel;
```

This method is invoked for a failed postcondition of a method invocation of the receiving object. The method is identified by the `selector sel`. The default implementation raises a `condition-condition` `SelectorCondition`.

Similar to precondition checking, postcondition checking is enabled by the `:cc-post` option on the command line of this program and not providing `-fno-checks` or `-fno-post-checks` to the compiler.

```
protected void
  unimplemented selector sel
       message: String message = nil;
```

Moan about the selector `sel` not yet having been implemented by the receiving object. This raises an `unimplemented` `SelectorCondition`.

```
protected void
  shouldNotImplement selector sel;
```

Contrairy to what the inheritance tells you about the `selector sel` being invokable for the receiving object, that object thinks otherwise.

```
protected void
  subclassResponsibility selector sel;
```

Moan about the receiving object defining a method for the selector `sel`, but actually the implementation of the method by the object thinks it should be implemented by a subclass. <h4>Cons methods</h4>

```
boolean
  consp;
```

Return `TRUE` iff the receiving object is a `Cons` cell. The default implementation returns `NO`.

```
OutputStream
  writeListElement OutputStream s;
```

Finish outputing the list, of which the receiving object is the tail, to the stream `s`. The default implementation writes itself as a dotted cdr at the end of the list. <h4>Coding methods</h4>

```
deferred boolean
  persistent-coding-p;
```

Return `YES` iff the receiving object is a persistent object. This is significant for distributed objects, where class objects and `Selector` instances must be persistent across different invocations.

<h4>Miscellaneous</h4>

```
pointer
  address;
```

Return the address of the receiving object as a pointer. This is here solely to be able to print the address of objects, for debugging purposes.

```
boolean
  coding-permanent-object-p;
```

Return `YES` if the receiving object should be maintained in the permanent object store when coding. This does not matter for archiving; it makes a difference for DO. Class objects and `Selector`s return `TRUE` for this; the default implementation returns `FALSE`.

```
void
   dump (boolean, boolean) (allow_self, allow_simple)
  level int level;
```

Dump the graph of which the receiving object is the root to stderr.

```
void
  dump;
```

Like `void dump (boolean, boolean)`, allowing self/simple printing and doing infinite recursion.

```
boolean
  dump_simple_p;
```

Return `TRUE` iff the receiving object can be dumped simply. This will be true for class objects, strings, numbers, etc. This method is overridden by `dump_self_p`. The default implementation returns `FALSE`.

```
boolean
  dump_self_p;
```

Return `TRUE` iff the receiving object wants to dump itself instead of having its variables scrutinized. This is used by collection objects and others which employ `pointer` typed variables. The default implementation returns `FALSE`.

```
OutputStream
  dump_simple OutputStream s;
```

Dump the receiving object to the stream `s`, simply. This is only ever invoked if the object returns `YES` for `dump_simple_p`. The default implementation simply prints `self` to the stream.

```
protected void
  dumpSelf MutableKeyed done
    indent MutableByteString prefix
    simple boolean allow_simple
     level int level
        to OutputStream s;
```

Have the receiving object dump itself. Only ever invoked if it returns `TRUE` for `dump_self_p`. The default implementation invokes `shouldNotImplement`.

```
deferred protected void
    dump MutableKeyed done
  indent MutableByteString prefix
  simple boolean allow_simple
   level int level
      to OutputStream s;
```

Hard worker for `dump`.

```
boolean
  gc_dead_p;
```

Return `YES` iff the receiving object has not yet been marked alive during the current run of the garbage collector. Class objects are never dead.

```
void
  gc_mark;
```

Mark the receiving object as being alive. This method is only needed by the container garbage collection scheme.

This method is invoked during Garbage Collection. During GC, the `Runtime` library is running in panic mode. If anything goed wrong, for instance a condition is signaled or raised, the program will abort. Moral: be careful during garbage collection.

class tom.Array

`Array` is the superclass of all arrays; it is an `Indexed Collection`

Inherits

State supers: State, Indexed, C

instance tom.Array

variables

`public int length;`
: The number of elements in the array.

`pointer contents;`
: A pointer to the elements of this array.

methods

```
void
  dealloc;
```

Clean up the memory this array is using.

```
deferred int
  elementByteSize;
```

Return the size, in bytes, of the elements contained in this `Array`.

```
id
  initAsCopyOf id other;
```

Get the elements from the `other`, and invoke `[self initCopy]`.

```
id (self)
  initCopy;
```

Duplicate our `contents` since that is what we own.

```
id (self)
  initWith int n
        at pointer addr;
```

Initialize with the indicated pointer and integer for contents and length.

```
Any
  member All object;
```

Return the element contained in this `Array`, which is `equal` to the `object`.

```
Any
  memq All object;
```

Like `member`, but the element is identified on reference equality.

```
deferred (pointer, int) (address, number)
  pointerToElements (int, int) (start, len)
pre
  start >= 0 && len >= -1
post
  number >= 0 && !number == !address;
```

Return the `address` of the first element of the receiving array in the range `(start, len)`, and the `number` of elements in that range.

```
void
  makeVanishingElementsPerform Invocation inv;
```

Like `makeElementsPerform`, but allow the element currently messaged to vanish from this array.

class tom.Bag

A Bag is a `Keyed Collection`.

Inherits

State supers: HashTable, Keyed

instance tom.Bag

methods

```
int
  at All object;
```

Return the number of times the element `key` is present in the bag.

```
Any
  at All object;
```

Return the `object` if present; `nil` otherwise.

```
Enumerator
  enumerator;
```

Undocumented.

```
id
  initWithEnumerator Enumerator e;
```

Undocumented.

class tom.MutableBag

Inherits

State supers: Bag, MutableKeyed

instance tom.MutableBag

methods

```
void
    add All object
  count int num;
```

Add the `object` `num` times.

```
void
  add All object;
```

Add the `object`.

class tom.BucketDictElement

Inherits

State supers: BucketElement

instance tom.BucketDictElement

variables

`All key;` This bucket element's key.

`public Any value;`
The value in this bucket element.

methods

```
Any
  key;
```

Undocumented.

```
id
  initWith (All, All) (k, v);
```

Designated initializer.

```
Any
  member All k;
```

Return the value associated with the key `k`, asking the `next` element if this element does not match.

The implementation by `BucketDictElement`, considers its `key` and returns its `value`.

```
Any
  member All k
   equal selector cmp;
```

Like member, but using the selector `cmp` to have the objects compare themselves.

```
Any
  memq All k;
```

Undocumented.

```
int
  add (All, All) (k, v);
```

Add the `(k, v)` pair to this bucket, if the key is not already present. Return the number by which this bucket's length has increased.

```
int
  addq (All, All) (k, v);
```

Add the (k, v) pair to this bucket, if the key is not already present. Return the number by which this bucket's length has increased.

```
(id, int)
  remove All k;
```

Remove the object with the key equal to k. Return the replacement for this element, and the number of bucket elements that were removed from this bucket list (max 1).

```
(id, int)
  removeq All k;
```

Remove the object with the identical key k. Return the replacement for this element, and the number of bucket elements that were removed from this bucket list (max 1).

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
(id, int)
  gc_mark_values;
```

Starting with this bucket element, remove the objects of which the `value` is `gc_dead`. Return the replacement for this element, and the number of bucket elements that were removed from this bucket list.

```
(id, int)
  gc_mark_keys;
```

Similar to `gc_mark_values`, but consider the liveness of the `key` instead of the `value`.

```
int
  rehash;
```

Rehash the key of the receiving element.

```
int
  rehashq;
```

Rehashq the key of the receiving element.

class tom.BucketElement

Inherits

State supers: State

instance tom.BucketElement

variables

`public id next;`
The next element in this bucket.

methods

```
Any
  member All key
pre
  key != nil;
```

Return the value associated with the `key`, asking the `next` element if this element does not match.

The implementation by `BucketElement` considers itself to be both the key and the value.

```
Any
  member All key
   equal selector cmp
pre
  key != nil;
```

Like member, but using the selector `cmp` to have the objects compare themselves.

```
Any
  memq All key
pre
  key != nil;
```

Like `member`, but use reference equality instead of the `equal` method.

```
int
  addElement id elt;
```

Add the `elt` to this bucket, if it is not already present. Return the number by which this bucket's length has increased.

```
int
  addqElement id elt;
```

Add the `elt` to this bucket, if it is not already present. Return the number by which this bucket's length has increased.

```
(id, int) (replacement, decrease)
  remove id elt
post
  !decrease -> replacement == self;
```

Remove the `elt` from this bucket, if present. Return the number by which the length of this bucket has decreased, and the replacement remainder of the bucket list.

```
(id, int) (replacement, decrease)
  removeq id elt
post
  !decrease -> replacement == self;
```

Remove the `elt` from this bucket, if present. Return the number by which the length of this bucket has decreased, and the replacement remainder of the bucket list.

```
void
  resizing_feed HashTable ht;
```

For resizing a hashtable, feed this element and those following elements, to the hashtable using the hashtable's `resizing_add`.

```
void
  resizing_add id n;
```

While resizing a hashtable, accept a new `next`.

```
int
  rehash;
```

Rehash the key of the receiving element.

```
int
  rehashq;
```

Rehashq the key of the receiving element.

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
(id, int)
  gc_mark_values;
```

Starting with this bucket element, remove those bucket elements of which the objects are `gc_dead`. Return the replacement for this element, and the number of bucket elements that were removed from this bucket list.

class tom.BucketIntDictElement

Inherits

State supers: BucketElement

instance tom.BucketIntDictElement

variables

`public int key;`
: This bucket element's key.

`public Any value;`
: The value in this bucket element.

methods

```
id
  initWith (int, All) (k, v);
```

Designated initializer.

```
Any
  member int k;
```

Return the value associated with the key `k`, asking the `next` element if this element does not match.

The implementation by `BucketIntDictElement`, considers its `key` and returns its `value`.

```
Any
  memq int k;
```

Undocumented.

```
int
  add (int, All) (k, v);
```

Add the `(k, v)` pair to this bucket, if the key is not already present. Return the number by which this bucket's length has increased.

```
int
  addq (int, All) (k, v);
```

Add the `(k, v)` pair to this bucket, if the key is not already present. Return the number by which this bucket's length has increased.

```
int
  rehash;
```

Return the integer key.

```
int
  rehashq;
```

Return the integer key.

```
(id, int)
  remove int k;
```

Remove the object with the key `k`. Return the replacement for this element, and the number of bucket elements that were removed from this bucket list (max 1).

```
(id, int)
  gc_mark_values;
```

Starting with this bucket element, remove the objects of which the `value` is `gc_dead`. Return the replacement for this element, and the number of bucket elements that were removed from this bucket list.

class tom.BucketPointerDictElement

Inherits

State supers: BucketElement

instance tom.BucketPointerDictElement

variables

`public pointer key;`
This bucket element's key.

`public Any value;`
The value in this bucket element.

methods

```
id
  initWith (pointer, All) (k, v);
```
Designated initializer.

```
Any
  member pointer k;
```
Return the value associated with the key `k`, asking the `next` element if this element does not match.

The implementation by `BucketDictElement`, considers its `key` and returns its `value`.

```
int
  add (pointer, All) (k, v);
```
Add the `(k, v)` pair to this bucket, if the key is not already present. Return the number by which this bucket's length has increased.

```
(id, int)
  gc_mark_values;
```
Starting with this bucket element, remove the objects of which the `value` is `gc_dead`. Return the replacement for this element, and the number of bucket elements that were removed from this bucket list.

```
int (code)
  rehash;
```
Rehash the pointer key.

```
int (code)
  rehashq;
```

Rehash the pointer key.

```
(id, int)
  remove pointer k;
```

Remove the object with the key `k`. Return the replacement for this element, and the number of bucket elements that were removed from this bucket list (max 1).

class tom.BucketSetElement

Inherits

State supers: BucketElement

instance tom.BucketSetElement

variables

`Any value;`
The key/value in this bucket element.

methods

```
Any
  key;
```
Undocumented.

```
id
  initWith All v;
```
Designated initializer.

```
Any
  member All key;
```
Return the value associated with the `key`, asking the `next` element if this element does not match.

The implementation by `BucketSetElement`, considers its `value` as the both the key and the value.

```
Any
  member All key
   equal selector cmp;
```
Like member, but using the selector `cmp` to have the objects compare themselves.

```
Any
  memq All key;
```
Undocumented.

```
int
  add All key;
```
Add the `key` to this bucket, if it is not already present. Return the number by which this bucket's length has increased.

```
int
  addq All key;
```

Add the `key` to this bucket, if it is not already present. Return the number by which this bucket's length has increased.

```
(id, int) (replacement, decrease)
  remove All key;
```

Remove this bucket, if it holds the `key`. Return the number by which the length of this bucket list has decreased, and the replacement remainder of the bucket list.

```
(id, int) (replacement, decrease)
  removeq All key;
```

Remove this bucket, if it holds the `key`. Return the number by which the length of this bucket list has decreased, and the replacement remainder of the bucket list.

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
void
  gc_mark_containers;
```

Tell the `value` to `gc_container_mark_elements`.

```
(id, int)
  gc_mark_values;
```

Starting with this bucket element, remove the objects of which the `value` is `gc_dead`. Return the replacement for this element, and the number of bucket elements that were removed from this bucket list.

```
int
  rehash;
```

Rehash the key of the receiving element.

```
int
  rehashq;
```

Rehashq the key of the receiving element.

class tom.Bundle

Inherits

State supers: State, Constants, Runtime

variables

`static Bundle main;`
The main bundle, i.e. the bundle describing the program and the units.

`static MutableIndexed units_path;`
The path as registered by the units.

methods

```
OutputStream (s)
  help OutputStream s
  done MutableKeyed done;
```

Undocumented.

```
private pointer
  load String object
  unit String unit;
```

Load the `object`, returning the underlying operating system handle upon success, or the NULL pointer upon failure. An `error` is signaled in case of the latter. The `unit` is the name of the unit supposedly contained in the `object`. This unit, when present, will be resolved.

```
pointer
    loadUnit String unit
  fromObject String object;
```

Load the `unit` from the `object`.

```
pointer
  load String object;
```

Derive the unit name from the `object` name and invoke `loadUnit fromObject`.

```
void
  load MutableArray arguments;
```

Accept `:main-bundle-dir` option and allocate the `main` bundle if found.

```
String
    locate-file String file
      extension String ext
  with-version: String version = nil;
```

Forward this to the main bundle.

```
instance (id)
  main;
```

Return the main bundle, creating it iff necessary.

```
void
  registerUnitDirectory String dir;
```

Register the `dir` as to contain resources for one of the units.

instance tom.Bundle

variables

`public String directory;`
: The directory.

`pointer handle;`
: Iff not the null pointer, the handle (in the underlying operating system) to the code loaded for this bundle. Iff it is the null pointer, the code has not (yet) been loaded.

methods

```
id (self)
  init String d;
```

Undocumented.

```
String
    locate-file String file
      extension String ext
  with-version: String version = nil;
```

Locate the `file` for the `version` in this bundle. If not found, search the main bundle. Iff still not found, it is searched for in the registered unit directories.

The extension `ext`, if not `nil`, is appended to the `file`, with a dot (`.`) in between.

class tom.ByteArray

Like the `CharArray`, the `ByteArray` is a particular kind of `Array`, which is here for abstraction purposes, but which is never actually used, since the `ByteString` holds the same kind of state, but provides much more functionality.

Inherits

State supers: Array

instance tom.ByteArray

methods

```
protected id
  initWithEnumerator Enumerator e;
```
Undocumented.

```
Any
  at int index;
```
Undocumented.

```
byte
  at int index;
```
Undocumented.

```
char
  at int index;
```
Undocumented.

```
int
  at int index;
```
Undocumented.

```
long
  at int index;
```
Undocumented.

```
float
  at int index;
```
Undocumented.

```
double
  at int index;
```

Undocumented.

```
int
  elementByteSize;
```

Undocumented.

```
int
  hash;
```

Undocumented.

```
(pointer, int)
  pointerToElements (int, int) (start, len);
```

Undocumented.

```
int
  writeRange (int, int) (start, len)
        into MutableByteArray destination
          at int position;
```

Undocumented.

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented. <h4>Copying</h4>

```
class (State)
  mutableCopyClass;
```

Return the `MutableByteArray` class.

class tom.ByteStream

Instances of the ByteStream class are an abstraction of the UNIX file descriptors.

Inherits

State supers: Descriptor, InputOutputStream <h4>InputStream</h4>

instance tom.ByteStream

methods

```
byte
  read;
```
Read a byte and return it, raising an exception on end-of-file or error.

```
int
  read;
```
Read a byte and return it, returning EOF on end-of-file.

```
int
  readRange (int, int) (start, num)
       into MutableByteArray buffer;
```
Have the buffer read at most num bytes from the receiving ByteStream starting at start.

```
void
  write byte b;
```
<h4>OutputStream</h4>

```
int
  write byte b;
```
Undocumented.

```
int
  writeBytes int num
        from pointer address;
```
Undocumented.

class tom.ByteString

A `ByteString` is a `String` and a `ByteArray`, which can do all kinds of nice string-like things.

Requesting a `substring` of a `ByteString` results in a `ByteSubstring` to be returned. This will mimic a `ByteString` as much as possible, including hashing, equality, uniquing, printing, copying, etc, but they do not share a common superclass between `String` and `ByteString`.

In the future, the `ByteString` instance actual functionality could be put into a `ByteFullstring`, enabling the `ByteSubstring` to actually become a subclass of `ByteString`...

Inherits

State supers: ByteArray, String, C, Constants

variables

`static CharacterEncoding default_encoding;`

The default character encoding for `ByteString` instances.

Never refer this variable directly; always ask the string (even if it is `self`) for its `encoding`. A normal `ByteString` will then return this `default_encoding`.

methods

```
OutputStream
  help OutputStream s
  done MutableKeyed done;
```

Output information on the `ByteString` unit arguments.

```
void
  load MutableArray arguments;
```

Set the default byte encoding. If it is not specified on the command line, `iso-8859-1` will be used.

Before this method is invoked by the runtime library, the `default_encoding` will be a `USASCIIEncoding`.

```
void
  switchToEncoding String name;
```

Switch to the encoding with the `name`, moaning if it fails (without changing the current encoding).

instance tom.ByteString

methods

```
char
  at int index;
```

Return the Unicode character for the byte at `index`.

```
(pointer, int)
  byteStringContents;
```

Undocumented.

```
boolean
  equal String other;
```

Undocumented.

```
int
  hashRange (int, int) (start, len);
```

Undocumented.

```
boolean
  equalByteString ByteString other;
```

Undocumented.

```
boolean
  equalCharString CharString other;
```

Undocumented.

```
boolean
  equalUniqueString UniqueString other;
```

Undocumented.

```
protected id (self)
  init (pointer, int) (p, num);
```

Initialize the newly allocated instance with the `num` bytes at `p`. The receiving instance will 'own' the memory at `p`.

```
id (self)
  initCopy (pointer, int) (p, num);
```

Initialize the newly allocated instance with a copy of the `num` bytes at `p`.

```
MutableByteString
  mutableSubstring (int, int) (start, len);
```

Return a new instance of the receiver's `mutableCopyClass`, initialized with a substring from the receiver's range `(start, len)`.

```
String
  substring (int, int) (start, len);
```
Undocumented.

```
UniqueByteString
  uniqueString;
```
Undocumented.

```
OutputStream
  write OutputStream s;
```
Undocumented. <h4>Copying</h4>

```
class (State)
  mutableCopyClass;
```
Return the `MutableByteString` class.

```
CharacterEncoding
  encoding;
```
Return the encoding of the receiving `ByteString`. The default implementation returns the `default_encoding`.

```
String
  stringByDecoding String encoding_name;
```
Undocumented.

```
String
  stringByDemapping CharArray demap;
```
Undocumented.

```
boolean
  isAlpha byte b;
```
Return `TRUE` the character denoted by the byte `b` in the encoding of the receiving string is a letter.

```
boolean
  isDigit byte b;
```
Return `TRUE` the character denoted by the byte `b` in the encoding of the receiving string is a digit.

```
boolean
  isLower byte b;
```
Return `TRUE` the character denoted by the byte `b` in the encoding of the receiving string is a lowercase letter.

```
boolean
  isPunct byte b;
```

Return `TRUE` the character denoted by the byte `b` in the encoding of the receiving string is a punctuation character.

```
boolean
  isSpace byte b;
```

Return `TRUE` the character denoted by the byte `b` in the encoding of the receiving string is a space character.

```
boolean
  isUpper byte b;
```

Return `TRUE` the character denoted by the byte `b` in the encoding of the receiving string is a uppercase letter.

```
byte
  toLower byte b;
```

Return the lower-case version of the byte `b`, according to the encoding of the receiving string. If the character is not in upper-case, it is returned unharmed.

```
byte
  toUpper byte b;
```

Return the upper-case equivalent of the byte `b`, according to the encoding of the receiving string. If the character is not in lower-case, it is returned unharmed.

```
int
  digitValue byte b;
```

Return the value equivalent of the byte `b`, for which this string should return `TRUE` when asked `isDigit`.

```
int
  alphaValue byte b;
```

Return the index of the letter `b` relative to the start of its letter range. Thus, 'a' returns 0, 'f' returns 5, etc.

```
id
  downcase;
```

This version of `downcase` overrides the implementation by `String`, since this one is faster due to avoiding the unnecessary conversion to/from Unicode.

```
id
  upcase;
```

Like `downcase`, this just is a faster implementation than the one provided by `String`.

class tom.ByteSubstring

A ByteSubstring is a substring of a constant ByteString. It tries to maskerade as one (even though it is certainly not an Array), possibly not perfect (yet).

Inherits

State supers: String, C

methods

```
instance (id)
  with (int, int) (start, len)
    in ByteString string;
```

Undocumented.

instance tom.ByteSubstring

variables

ByteString string;
: The string we're begin part of.

int start;
: The start of us in our string.

public int length;
: The length of us, which is never < 0.

methods

```
id
  init (int, int) (s, l)
    in ByteString str;
```

Designated initializer.

```
class (State)
  mutableCopyClass;
```

Return the MutableByteString class.

```
byte
  at int index;
```

Retrieve the byte at the index.

```
ByteNumber
  at int index;
```

Return the `ByteNumber` at the `index`.

```
Enumerator
  enumerator;
```
Return a restricted enumerator on the underlying string.

```
id
  initWithEnumerator Enumerator e;
```
Undocumented.

```
(pointer, int)
  pointerToElements (int, int) (begin, len);
```
Another low level access method.

```
ByteSubstring
  substring (int, int) (begin, len);
```
Return a new substring on our `string`—we do not cascade substrings.

```
MutableByteString
  mutableSubstring (int, int) (begin, len);
```
Undocumented.

```
(pointer, int)
  byteStringContents;
```
Low level access method.

```
boolean
  equal String other;
```
Undocumented.

```
int
  hash;
```
Undocumented.

```
int
  hashRange (int, int) (begin, len);
```
Undocumented.

```
boolean
  equalByteString ByteString other;
```
Undocumented.

```
boolean
  equalCharString CharString other;
```
Undocumented.

```
boolean
  equalUniqueString UniqueString other;
```

Undocumented.

```
UniqueByteString
  uniqueString;
```

Undocumented.

```
OutputStream
  write OutputStream s;
```

Undocumented.

class tom.C

The C class provides low-level memory manipulation functionality. With it, a lot of collection and string methods can be written in TOM instead of needing to be written in C.

Inherits

Behaviour supers: All

methods

```
void
  free pointer address;
```

Release the memory at address.

```
pointer
  malloc int length;
```

Return a pointer to a newly allocated memory region of length bytes.

```
pointer
  calloc (int, int) (num, bytes);
```

Return a pointer to newly allocated and zeroed memory region of num elements of each bytes size.

```
pointer
  realloc (pointer, int) (address, length);
```

Return a pointer to the resized memory region at address which must hold length bytes. The address returned can differ from the previous address.

```
int
  memcmp (pointer, pointer, int) (one, other, length);
```

Return 0 iff the length bytes at one equal the bytes at other.

```
pointer
  memcpy (pointer, pointer, int) (to, from, length);
```

Copy the length bytes from from to to. Return to.

```
pointer
  memmove (pointer, pointer, int) (to, from, length);
```

Copy the length bytes from from to to, safely. Return to.

```
void
  bzero (pointer, int) (p, num);
```

Set the num bytes at p to 0.

instance tom.C

The `C` instance can be and is totally empty.

class tom.CharArray

Like the `ByteArray`, the `CharArray` is a particular kind of `Array`, which is here for abstraction purposes, but which is never actually used, since the `CharString` holds the same kind of state, but provides much more functionality.

Inherits

State supers: Array

instance tom.CharArray

methods

```
Any
  at int index;
```
Undocumented.

```
byte
  at int index;
```
Undocumented.

```
char
  at int index;
```
Return the char value at `index`. This is the elementary retrieval method for character arrays.

```
int
  at int index;
```
Undocumented.

```
long
  at int index;
```
Undocumented.

```
float
  at int index;
```
Undocumented.

```
double
  at int index;
```
Undocumented.

```
int
  elementByteSize;
```

Undocumented.

```
protected id
  initWithEnumerator Enumerator e;
```

Undocumented.

```
(pointer, int)
  pointerToElements (int, int) (start, len);
```

Undocumented. <h4>Copying</h4>

```
class (State)
  mutableCopyClass;
```

Return the `MutableCharArray` class.

class tom.CharacterEncoding

The `CharacterEncoding` class defines the interface of the byte and character encodings for predicates and conversions.

Inherits

Behaviour supers: All

instance tom.CharacterEncoding

Inherits

Behaviour supers: All

methods

```
deferred String
  name;
```

Return the name of this encoding. <h4>Character conversions</h4>

```
deferred char
  decode byte b;
```

Return the decoded byte `b`, i.e. the Unicode character corresponding to the byte `b` in the receiving encoding.

```
deferred byte
  encode char c;
```

Return the byte encoding of the character `c`. If the byte equivalent of the character `c` does not exist in the receiving encoding, an `encoding-condition` is signaled, and the byte encoded is the `byteValue` of the object returned, or 127 if `nil` is returned.

<h4>Character predicates</h4>

```
deferred boolean
  isAlpha byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a letter.

```
deferred boolean
  isDigit byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a digit.

```
deferred boolean
  isLower byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a lowercase letter.

```
deferred boolean
  isPunct byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a punctuation character.

```
deferred boolean
  isSpace byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a space character.

```
deferred boolean
  isUpper byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a uppercase letter.

```
deferred byte
  toLower byte b;
```

Return the lowercase version of the byte `b`, according to the receiving encoding. If the character is not in uppercase, it is returned unharmed.

```
deferred byte
  toUpper byte b;
```

Return the uppercase version of the byte `b`, according to the receiving encoding. If the character is not in lowercase, it is returned unharmed.

```
deferred int
  digitValue byte b;
```

Return the numeric value of the digit denoted by the byte `b` in the receiving encoding.

```
deferred int
  alphaValue byte b;
```

Return the index of the letter `b` relative to the start of its letter range. Thus, 'a' returns 0, 'f' returns 5, etc.

class tom.CharEncoding

An instance of the `CharEncoding` class maintains information on on a particular mapping for encoding a subset of Unicode characters to 8-bit bytes. An example of such mappings is <code>iso-8859-1</code>, which is the well known western european byte encoding, of which <code>USASCII</code> is a subset.

Inherits

State supers: State, Constants, Conditions, CharacterEncoding

variables

`static MutableDictionary encodings;`
Currently known encodings.

methods

```
ByteArray
  loadBytes int num
       from String name
  extension String ext;
```

Load `num` bytes from the file with the `name` and the extension `ext` (sans dot). The full path of the file is obtained from the `main Bundle`.

```
instance (id)
  named String name;
```

Return the `CharEncoding` known as the `name`. This always succeeds, as a `CharEncoding` reads the resources it needs on demand.

instance tom.CharEncoding

variables

`public String name;`
The name of this encoding.

`CharArray decoding;`
The decoding map.

`IntDictionary encoding;`
The encoding map.

`ByteArray to_lower;`
The byte map for conversion to lower case within the encoding.

`ByteArray to_upper;`
The byte map for conversion to upper case within the encoding.

`ByteArray to_title;`
The byte map for conversion to title case within the encoding.

`ByteArray is_digit;`
The bitmap for testing whether a byte is a digit.

`ByteArray is_letter;`
The bitmap for testing whether a byte is a letter.

`ByteArray is_lower;`
The bitmap for testing whether a byte is lower case.

`ByteArray is_punct;`
The bitmap for testing whether a byte is a punctuation character.

`ByteArray is_space;`
Bitmap for space predicate.

`ByteArray is_upper;`
The bitmap for testing whether a byte is upper case.

methods

```
id
  init String n;
```
Designated initializer.

```
char
  decode byte b;
```
Return the decoded byte `b`, i.e. the Unicode character corresponding to the byte `b` in the receiving encoding.

```
CharArray
  decoding;
```
Return the `decoding` map, reading it iff necessary.

```
byte
  encode char c;
```
Return the byte encoding of the character `c`. If the byte equivalent of the character `c` does not exist in the receiving encoding, an `encoding-condition` is signaled, and the byte encoded is the `byteValue` of the object returned, or 127 if `nil` is returned.

```
IntDictionary
  encoding;
```
Return the `encoding` map, creating it from the `decoding` map if necessary.

```
protected ByteArray
  loadConversion String conversion;
```
Load and return the conversion table for the `conversion` of the receiving encoding.

```
protected ByteArray
  loadPredicateSet String predicate;
```

Load and return the predicate set for the `predicate` of the receiving encoding.

```
boolean
  isAlpha byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a letter.

```
boolean
  isDigit byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a digit.

```
boolean
  isLower byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a lowercase letter.

```
boolean
  isPunct byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a punctuation character.

```
boolean
  isSpace byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a space character.

```
boolean
  isUpper byte b;
```

Return `TRUE` the character denoted by the byte `b` in the receiving encoding is a uppercase letter.

```
byte
  toLower byte b;
```

Return the lowercase version of the byte `b`, according to the receiving encoding. If the character is not in uppercase, it is returned unharmed.

```
byte
  toUpper byte b;
```

Return the uppercase version of the byte `b`, according to the receiving encoding. If the character is not in lowercase, it is returned unharmed.

```
int
  digitValue byte b;
```

Return the numeric value of the digit denoted by the byte `b` in the receiving encoding.

```
int
  alphaValue byte b;
```

Return the index of the letter `b` relative to the start of its letter range. Thus, 'a' returns 0, 'f' returns 5, etc.

class tom.USASCIIEncoding

A replacement for a real `CharEncoding` used during program initialization.

Inherits

State supers: State, CharacterEncoding

variables

`static USASCIIEncoding shared;`
The one and only `USASCIIEncoding` object.

methods

```
instance (id)
  shared;
```

Undocumented.

instance tom.USASCIIEncoding

methods

```
String
  name;
```

We're really a dummy, so we do not have a name. In fact, that is how we're recognized.

```
char
  decode byte b;
```

This is acceptable for iso-8859-1.

```
byte
  encode char c;
```

This is acceptable for iso-8859-1.

```
boolean
  isAlpha byte b;
```

Undocumented.

```
boolean
  isDigit byte b;
```

Undocumented.

```
boolean
  isLower byte b;
```
Undocumented.

```
boolean
  isPunct byte b;
```
Undocumented.

```
boolean
  isSpace byte b;
```
Undocumented.

```
boolean
  isUpper byte b;
```
Undocumented.

```
byte
  toLower byte b;
```
Undocumented.

```
byte
  toUpper byte b;
```
Undocumented.

```
int
  digitValue byte b;
```
Undocumented.

```
int
  alphaValue byte b;
```
Undocumented.

class tom.CharString

Inherits

State supers: CharArray

instance tom.CharString

methods

```
boolean
  equal String other;
```

Undocumented.

```
boolean
  equalByteString ByteString other;
```

Undocumented.

```
boolean
  equalCharString id other;
```

Undocumented.

```
boolean
  equalUniqueString UniqueString other;
```

Undocumented.

```
protected id
  init (pointer, int) (p, num);
```

Initialize the newly allocated instance with the `num` characters at `p`. The receiving instance will 'own' the memory at `p`.

```
id (self)
  initCopy (pointer, int) (p, num);
```

Initialize the newly allocated instance with a copy of the `num` chars at `p`.

```
MutableCharString
  mutableSubstring (int, int) (start, len);
```

Undocumented.

```
CharString
  substring (int, int) (start, len);
```

Undocumented.

```
UniqueCharString
  uniqueString;
```

Undocumented.

```
class (State)
  classForCoder Encoder coder;
```

Undocumented. <h4>Copying</h4>

```
class (State)
  mutableCopyClass;
```

Return the `MutableCharString` class.

class tom.Condition

Inherits

State supers: State

methods

```
instance (id)
     for All object
   class ConditionClass condition_class
 message String msg;
```

Return a new `Condition` for the indicated circumstances.

instance tom.Condition

variables

`public ConditionClass condition_class;`
The condition class of the condition indicated by this `Condition`.

`public Any object;`
The object by/for which this condition was raised.

`public String message;`
The message explaing what actually happened.

`public boolean raised;`
Iff `TRUE`, this condition was raised, otherwise it was signaled.

methods

```
protected id
  initFor All o
    class ConditionClass cc
  message String msg;
```

Undocumented.

```
void
  raise;
```

Raise this condition; guaranteed never to return.

```
Any
  signal;
```

Signal this condition. If a handler performs a non-local break, this method does not return. If no handler is installed, `nil` is returned. If a handler returns something different from this condition, signaling is terminated and that value is returned.

```
OutputStream (s)
  writeFields OutputStream s;
```

Undocumented.

class tom.SelectorCondition

Inherits

State supers: Condition

methods

```
instance (id)
      for All object
    class ConditionClass condition_class
  message String msg
 selector selector sel;
```

Return a new `SelectorCondition` for the indicated circumstances.

instance tom.SelectorCondition

variables

`selector sel;`
    The selector which was sent to the `object`.

methods

```
protected id
   initFor All o
     class ConditionClass cc
   message String msg
  selector selector s;
```

Undocumented.

```
selector
  selector;
```

Return the selector, `sel`.

```
OutputStream (s)
  writeFields OutputStream s;
```

Undocumented.

class tom.ConditionClass

Instances of the `ConditionClass` define the hierarchy of conditions as carried by `Condition` instances. Conditions classes could be real tom classes, but the features provided by said mechanism are too baroque for this purpose—only the inheritance is needed.

The tom condition class hierarchy does not employ multiple inheritance.

TOM conditions are evidently modelled after CL.

Inherits

State supers: State

methods

```
instance (id)
  with instance (id) super_condition
  name ByteString name;
```
Undocumented.

instance tom.ConditionClass

variables

`id super_condition;`
  Our super condition class. The super condition class of the top `condition` is `nil`.

`public ByteString name;`
  Our descriptive name.

methods

```
protected id
  init id sc
  name ByteString nm;
```
Undocumented.

```
boolean
  isConditionSuper id other;
```
Return YES iff `other` is a super condition class of the receiving condition class.

class tom.Conditions

The `Conditions` class is an instance-less non-static-state-less class providing predefined conditions. Not-predefined, i.e. user defined, conditions should be made available to the world through an extension of `Conditions`.

variables

`static All unhandled_condition_handler;`
: The object informed of unhandled condition raises. Iff `nil`, the program aborts when a condition being raised is not handled.

`static selector unhandled_condition_selector;`
: The selector of the message to be sent to the `unhandled_raise_handler`. This method accepts a single argument, which will be a `Condition`.

`static ConditionClass condition;`
: Various (and numerous) condition classes, indented according to condition inheritance.

`static ConditionClass warning;`

`static ConditionClass unimplemented;`

`static ConditionClass encoding-condition;`

`static ConditionClass serious-condition;`

`static ConditionClass runtime-condition;`

`static ConditionClass runtime-fatal;`

`static ConditionClass nil-receiver;`

`static ConditionClass unrecognized-selector;`

`static ConditionClass uncaught-throw;`

`static ConditionClass program-condition;`

`static ConditionClass unknown-class-condition;`

`static ConditionClass coding-condition;`

`static ConditionClass type-condition;`

`static ConditionClass lock-condition;`

`static ConditionClass condition-condition;`

`static ConditionClass error;`

`static ConditionClass file-error;`

`static ConditionClass stream-error;`

`static ConditionClass stream-eos;`

`static ConditionClass signal-condition;`

`static ConditionClass signal-hup;`

`static ConditionClass signal-int;`

`static ConditionClass signal-bus;`

`static ConditionClass signal-segv;`

instance tom.Conditions

class tom.Cons

Inherits

State supers: State

methods

```
instance (id)
  with (All, All) (a, d);
```

Return a newly allocated instance with the `a` and `d` as the `car` and `cdr`, respectively.

```
instance (id)
  cons All a
     : All d = nil;
```

Return a newly allocated instance with the `a` and `d` as the `car` and `cdr`, respectively. The cdr `d` defaults to `nil`.

instance tom.Cons

variables

`public Any car;`
The element contained in this `Cons` cell, and the remainder of the list.

`public Any cdr;`

methods

```
protected id (self)
  init (All, All) (a, d);
```

Designated initializer.

```
boolean
  consp;
```

Return `YES`.

```
(Any, Any)
  decons;
```

Return the `(car, cdr)` in a tuple.

```
void
  set_car All c;
```

Set the `car` to the object `c`.

```
void
  set_cdr All c;
```
Set the `cdr` to the object `c`.

```
void
  encodeUsingCoder Encoder coder;
```
Encode the receiving object to the `coder`.

```
boolean
  equal id other;
```
Return `TRUE` if the receiving list is equal to the `other` list. Elements are compared with `equal`.

```
void
  initWithCoder Decoder coder;
```
Decode the receiving object from the `coder`.

```
id
  member All object;
```
Return the Cons cell whose car is `equal` to the `object`.

```
id
  memq All object;
```
Like `member`, but the element is identified on reference equality.

```
OutputStream
  write OutputStream s;
```
Output the list, of which the receiving `Cons` cell is the start, to the stream `s`.

```
OutputStream
  writeListElement OutputStream s;
```
Continue outputing the list, of which the receiving `Cons` cell is an element and not the head, to the stream `s`.

class tom.Constants

The constants used throughout the tom unit.

variables

<h4>`Trie` constants</h4>

`const TRIE_PLAIN = 0;`
The plain option indicates absence of other options.

`const TRIE_REVERSED = 1;`
Reverse the string before insertion or lookup.

`const TRIE_FOLD_CASE = 2;`
Ignore the case during a lookup; use lower-case characters during an insert.

`const TRIE_LOOKUP_PREFIX = 4;`
Do not require a full match in a lookup; the longest prefix will match with this option specified.

<h4>Open flags</h4>

`const FILE_EXIST_NOTHING = 1;`
If the file exists, do nothing (and return nil).

`const FILE_EXIST_RAISE = 2;`
If the file exists, raise a condition.

`const FILE_EXIST_TRUNCATE = 4;`
If the file exists and `output_p`, truncate it to zero length.

`const FILE_EXIST_SUPERSEDE = 8;`
If the file exists and `output_p`, a new version of the file will be created (by unlinking the old file first).

`const FILE_NOT_EXIST_NOTHING = 16;`
If the file does not exist, do nothing (and return nil). If only `input_p`, `nil` will also be returned if it can't be opened anyway.

`const FILE_NOT_EXIST_RAISE = 32;`
If the file does not exist, raise a condition.

`const FILE_NOT_EXIST_CREATE = 64;`
If the file does not exist and `output_p`, create it.

`const FILE_APPEND = 128;`
Every write will append to the end of the file.

`const FILE_MASK = 255;`
A mask for the above flags.

`const FILE_TYPE_NONEXISTENT = 0;`
Retrieving information about a file.

`const FILE_TYPE_OTHER = 1;`

`const FILE_TYPE_SOCKET = 2;`

`const FILE_TYPE_LINK = 3;`

`const FILE_TYPE_REGULAR = 4;`

`const FILE_TYPE_BLOCK = 5;`

`const FILE_TYPE_DIRECTORY = 6;`

`const FILE_TYPE_CHARACTER = 7;`

`const FILE_TYPE_FIFO = 8;`

<h4>Positioning a `SeekableStream`</h4>

`const STREAM_SEEK_SET = 0;`

Position absolute.

`const STREAM_SEEK_CUR = 1;`

Position relative to the current position.

`const STREAM_SEEK_END = 2;`

Position relative to the end of the file.

<h4>`TypeDescription` types.</h4>

`const TYPEDESC_VOID = 0;`

This value indicates the `void` type. The other values follow the same naming convention.

`const TYPEDESC_BOOLEAN = 1;`

`const TYPEDESC_BYTE = 2;`

`const TYPEDESC_CHAR = 3;`

`const TYPEDESC_INT = 4;`

`const TYPEDESC_LONG = 5;`

`const TYPEDESC_FLOAT = 6;`

`const TYPEDESC_DOUBLE = 7;`

`const TYPEDESC_POINTER = 8;`

`const TYPEDESC_SELECTOR = 9;`

`const TYPEDESC_REFERENCE = 10;`

`const TYPEDESC_DYNAMIC = 11;`

`const TYPEDESC_NUM = 12;`

This is not a real type; it merely denotes the number of `TYPEDESC_` values.

instance tom.Constants

class tom.Date

The Date class implements absolute times using doubles to represent the number of seconds passed since a certain reference date. Internally the time at which the Date class was initialized is used as reference. As absolute reference the first instant of January 1, 2001 is used. All gregorian calculation functions use an absolute date which is the number of days since the Gregorian date December 31, 1 BC.

Inherits

State supers: State

variables

`const EPSILON = 1e-06;`
: For two dates to be considered equal they should be no further apart than `EPSILON`.

`const OFFSET_DISTANT_FUTURE = 1e+100;`

`const OFFSET_DISTANT_PAST = -OFFSET_DISTANT_FUTURE;`

`const SECONDS_PER_DAY = 86400.0;`

`const ABSOLUTE_REFDATE = 730486;`
: The number of days from the imaginary Gregorian date Sunday, 31 december 1 BC to our reference date (January 1 2001).

`static public double relative_offset;`
: Some offset from the reference date relative to which all Date instances maintain their notion of time. This is set in `load`, ensuring a high accuracy of dates near the moment in time during which this program is running.

`static public Date distant_future;`
: A date in the very far future and a date in the very far past.

`static public Date distant_past;`

methods

```
double (now)
  relativeTimeIntervalSinceNow
post
  now > 0.0;
```

Return the number of seconds after `relative_offset` it is now.

```
double
  timeIntervalSinceReferenceDate;
```

Return the number of seconds after the absolute reference date it is now. This number is negative for dates before the first instant of January 1, 2001.

```
(int, double)
  absoluteAndSecondsOfTimeInterval double ti;
```

Return the absolute date and the seconds passed in that day for a time interval since the reference date.

```
int
  absoluteFromGregorian (int, int, int) (year, month, day);
```

The number of days elapsed between the Gregorian date 12/31/1 BC and `(year, month, day)`. The Gregorian date Sunday, December 31, 1 BC is imaginary.

```
int
  absoluteFromIso (int, int, int) (year, week, day);
```

The number of days elapsed between the Gregorian date 1 BC December 31 and DATE. The 'ISO year' corresponds approximately to the Gregorian year, but weeks start on Monday and end on Sunday. The first week of the ISO year is the first such week in which at least 4 days are in a year. The ISO commercial DATE has the form `(year, week, day)` in which `week` is in the range 1..52 and `day` is in the range 0..6 (1 == Monday, 2 == Tuesday, ..., 0 == Sunday). The Gregorian date Sunday, December 31, 1 BC is imaginary.

```
int
  dayNameOnOrBefore (int, int) (day_name, absolute);
```

Returns the absolute date of the `day_name` on or before `absolute`. `day_name`==0 means Sunday, `day_name`==1 means Monday, and so on.

Note: Applying this function to `absolute+6` gives us the `day_name` on or after an absolute day d. Similarly, applying it to `absolute+3` gives the `day_name` nearest to `absolute`, applying it to `absolute-1` gives the `day_name` previous to `absolute`, and applying it to `absolute+7` gives the `day_name` following `absolute`.

```
int
  dayNumber (int, int, int) (year, month, day);
```

Return the day number within the year of the date `(year, month, day)`. For example, dayNumber (1, 1, 1987) returns the value 1, while dayNumber (12, 31, 1980) returns 366.

```
int
  dayOfWeekOfAbsolute int absolute;
```

Return the Gregorian day of the week for `absolute` where 0==Sunday, 1==Monday, ..., 6==Saturday.

```
(int, int, int)
  gregorianFromAbsolute int date;
```

Compute the list (month, day, year) corresponding to the absolute DATE. The absolute date is the number of days elapsed since the (imaginary) Gregorian date Sunday, December 31, 1 BC.

```
boolean
  isLeapYear int year;
```

Return TRUE iff year is a Gregorian leap year.

```
(int, int, int)
  isoFromAbsolute int absolute;
```

Compute the 'ISO commercial date' corresponding to the absolute. The ISO year corresponds approximately to the Gregorian year, but weeks start on Monday and end on Sunday. The first week of the ISO year is the first such week in which at least 4 days are in a year. The ISO commercial date has the form (year week day) in which week is in the range 1..52 and day is in the range 0..6 (1 = Monday, 2 = Tuesday, ..., 0 = Sunday). The absolute date is the number of days elapsed since the (imaginary) Gregorian date Sunday, December 31, 1 BC.

```
int
  lastDayOfMonth int month
            year int year;
```

Return the last day of the month month of the year year.

```
void
  load MutableArray arguments;
```

Perform class initialization.

```
Date
  now;
```

Return a date instance representing this moment.

```
protected double
  relativeTimeIntervalOfAbsoluteAndSeconds (int, double) (absolute, seconds);
```

Return the absolute date and the seconds passed in that day for a time interval since the reference date.

```
double
  timeIntervalOfAbsoluteAndSeconds (int, double) (absolute, seconds);
```

Return the absolute date and the seconds passed in that day for a time interval since the reference date.

instance tom.Date

variables

`double relative_ti;`

methods

```
int
  compare id other;
```

Returns -1 if the receiver is earlier than `other` 0 if the difference is smaller that `EPSILON` and 1 if the receiver is after `other`.

```
id
  dateWithOffset double ti;
```
Return a new instance initialized at `ti` seconds after the receiver.

```
Date
  earlierDate Date other;
```
Return `other` if it is earlier than the receiver, return the receiver otherwise.

```
boolean
  equals id d;
```
Return `TRUE` iff the receiver is within `EPSILON` seconds of `d`.

```
protected id
  init double d;
```
Designated initializer.

```
id
  init;
```
Initialize with the current time.

```
id
  initWithTimeIntervalSinceNow double ti;
```
Initialize with `ti` seconds after the current time.

```
id
  initWithTimeIntervalSinceReferenceDate double ti;
```
Initialize with `ti` seconds after the absolute reference date January 1, 2001.

```
Date
  laterDate Date other;
```
Return `other` iff it is later than the receiver, return the receiver otherwise.

```
protected double
  relativeTimeInterval;
```
Return the number of seconds after `relative_offset`

```
double
  timeIntervalSinceDate Date d;
```
Return the number of seconds passed since `d`. This number is negative if the receiver is earlier than `d`.

```
double
  timeIntervalSinceNow;
```

Return the number of seconds passed since now. This number is negative for dates before now.

```
double
  timeIntervalSinceReferenceDate;
```

Return the number of seconds passed since the absolute reference date. This number is negative for all dates before the first instant of January 1 2001.

```
OutputStream
  write OutputStream s;
```

Print this date in a human readable format, relative to GMT.

```
void
  encodeUsingCoder Encoder coder;
```

<h4>Encoding</h4>

```
void
  initWithCoder Decoder coder;
```

Undocumented.

class tom.Descriptor

A `Descriptor` is the abstraction of the UNIX file descriptor.

Inherits

State supers: State, Conditions

instance tom.Descriptor

variables

`public int descriptor;`
The file descriptor. This will be -1 if we're not actively open.

methods

```
void
  close;
```
Close this descriptor. If it succeeds, every read, write, or other operation afterwards will certainly fail. If it fails, this may not be the case.

```
void
  dealloc;
```
Close the `descriptor` if it is not `-1`.

```
id
  init;
```
Invoke `[self init -1]` to avoid ever closing file descriptor 0 by accident.

```
protected id
  init int fd;
```
Designated initializer: Initialize with `fd` as the `descriptor`.

```
int
  type-of-file;
```
Return one of the `FILE_TYPE_*` constants for the receiving File. Signal a `file_error` and return `FILE_TYPE_NONEXISTENT` if not open.

class tom.DictionaryContainer

A `DictionaryContainer` is a class which can be inherited by `Dictionary`-like objects, to allow them to be a container, with respect to their value objects. It is a seperate class, for inheritance by the `PointerDictionary` class.

Inherits

State supers: Container

instance tom.DictionaryContainer

class tom.ObjectDictionary

An `ObjectDictionary` is a `Dictionary` mapping objects to objects. It is the superclass of `Dictionary` and `EqDictionary`. The latter hash the key objects on their address, and uses pointer equivalence.

Inherits

State supers: HashTable, Mapped, DictionaryContainer

instance tom.ObjectDictionary

methods

```
Enumerator
  enumerator;
```

Return a `valueEnumerator`.

```
Any
  member All object;
```

Invoke `HashTable`'s implementation.

```
Any
  memq All object;
```

Invoke `HashTable`'s implementation.

```
Enumerator
  keyEnumerator;
```

Return an enumerator on the keys of this dictionary.

```
Enumerator
  valueEnumerator;
```

Return an enumerator on the values of this dictionary.

```
OutputStream
  write OutputStream s;
```

Undocumented.

class tom.Dictionary

Inherits

State supers: ObjectDictionary

instance tom.Dictionary

class tom.MutableDictionary

Inherits

State supers: Dictionary, MutableHashTable, MutableMapped

instance tom.MutableDictionary

methods

```
void
  remove All key;
```
Remove the mapping for the `key`.

```
void
  add All object;
```
Undocumented.

```
void
  set All value
   at All key
pre
  value != nil && key != nil;
```
Undocumented.

class tom.DictionaryEnumerator

Inherits

State supers: HashTableEnumerator

instance tom.DictionaryEnumerator

variables

`redeclare BucketDictElement elt;`

class tom.DictionaryValueEnumerator

Inherits

State supers: DictionaryEnumerator

instance tom.DictionaryValueEnumerator

methods

```
(boolean, Any) (valid, object)
  next;
```

Undocumented.

class tom.EqDictionary

Inherits

State supers: ObjectDictionary, EqHashTable

instance tom.EqDictionary

methods

```
id
  initWithEnumerator Enumerator e;
```

Undocumented.

class tom.MutableEqDictionary

Inherits

State supers: EqDictionary, MutableEqHashTable, MutableMapped

instance tom.MutableEqDictionary

methods

```
     void
       remove All key;
```
Remove the mapping for the key.

```
     void
       set All value
        at All key
     pre
       value != nil && key != nil;
```
Undocumented.

class tom.WeakKeyMutableEqDictionary

The WeakKeyMutableEqDictionary is identical to a MutableEqDictionary, except that when it is a container, the references to the keys are weak, whereas in the case of an ordinary MutableEqDictionary the value references are weak.

Inherits

State supers: MutableEqDictionary

instance tom.WeakKeyMutableEqDictionary

methods

```
void
  gc_container_mark_elements;
```

Almost identical to `MutableHashTable`'s implementation, but the bucket elements are asked to `gc_mark_keys` instead of `gc_mark_values`.

class tom.EqHashTable

Inherits

State supers: HashTable

instance tom.EqHashTable

methods

```
Any
  at All key;
```

Return the `key` if present, or `nil` otherwise.

```
void
  resizing_add BucketElement elt;
```

Undocumented.

class tom.MutableEqHashTable

Inherits

State supers: EqHashTable, MutableHashTable

instance tom.MutableEqHashTable

methods

```
void
  add BucketElement elt;
```

Undocumented.

class tom.EqSet

Inherits

State supers: Set, EqHashTable

instance tom.EqSet

class tom.MutableEqSet

Inherits

State supers: EqSet, MutableEqHashTable, MutableSet, Container Retrieve a (any) object from the set. This removes an element from the set and returns it.

instance tom.MutableEqSet

methods

```
void
  add All object;
```

Add the `object` to the receiving set.

```
void
  remove All object;
```

Remove `elt` from the receiving set, if present.

```
void
  gc_mark_containers
pre
  [self isContainer];
```

This method is invoked by the garbage collector to have the set containing all containers make those containers mark their elements. This method relies on the container containing the containers be itself a container.

class tom.Extension

Inherits

State supers: State, Conditions

variables

`static MutableArray extensions;`
All extensions, here to be protected against gc.

methods

```
instance (id)
  new pointer p;
```

Designated allocator. Do not use `alloc` or plain `new`.

instance tom.Extension

variables

`pointer rti;`
The runtime structure describing this extension.

`Indexed var_names;`
The names of this extension's variables.

methods

```
void
  dealloc;
```

An `Extension` should never be deallocated. This method raises, which is a panic during garbage collection...

```
boolean
  implements selector sel;
```

Return `YES` if this extension provides an implementation for the selector `sel`.

```
protected id (self)
  init pointer r;
```

Designated initializer.

```
class (State)
  meta;
```

Return the class object to which this extension belongs.

```
String
  name;
```

Return the name of this extension.

```
Indexed
  variables;
```

Return the names of the variables in this extension.

```
int
  typeOfVariableNamed String name
                   in All object;
```

Undocumented.

```
Any
  valueOfVariableNamed String name
                    in All object;
```

Undocumented.

```
OutputStream
  writeFields OutputStream s;
```

Undocumented.

```
dynamic
  perform selector sel
       on All object
     with dynamic arguments;
```

The equivalent of `perform with` where the method invoked is defined by this extension instead of the receiving object. Obviously, the `object` should actually have this extension as one of its extensions, i.e. `[object isKindOf [self meta]]` should be a precondition (and a postcondition too, but we're not interested after the fact).

```
dynamic
  perform selector sel
       on All object
        : Array arguments = nil;
```

Undocumented.

class tom.File

The File class offers an abstraction from files in the filesystem.

Inherits

State supers: ByteStream, SeekableStream, Constants, Conditions

variables

`const OPEN_INPUT = 256;`

`const OPEN_OUTPUT = 512;`

methods

```
String
            basename String filename
  without-extension: String ext = nil;
```

Return the `filename` removing any directory component, and removing the extension `ext` if it matches and is not `nil`.

```
String
  current_directory;
```

Return the current directory, as a directory name (tailing slash).

```
void
  set_current_directory String directory;
```

Set the current `directory`. This raises a `file-error` when problems arise.

```
String
  directory-of-file String name;
```

Return the directory-name of the directory containing the file named `filename`.

```
String
  expand-filename String filename
     relative-to: String directory = nil;
```

Expand the `filename` relative to the `directory`. If `directory` is `nil`, expansion is relative to the current working directory.

```
String
  express-filename String filename
       relative-to String directory;
```

Express the `filename` in terms relative to the `directory`.

```
String
  filename-as-directory String filename;
```

Return the `filename` as the name of a directory.

```
MutableArray
  filenames-in-directory String dir_name;
```

Return the filenames in the directory named `dir_name`.

```
String
  locate-file String file
   along-path Indexed path;
```

Return the filename of the `file` somewhere along the `path`. Return `nil` if it could not be found.

```
boolean
  file-exists String name;
```

Return `YES` iff the file with the `name` exists.

```
int
  type-of-file String name
  follow-link: boolean follow_link = YES;
```

Return one of the `FILE_TYPE_*` constants.

```
instance (id)
      open String name
   input: boolean input_p = FALSE
  output: boolean output_p = FALSE
   flags: int action = 0;
```

Return a new `File`.

```
instance (id)
         open String name
   alongPath Indexed path
      subdir: String subdir1 = nil
  subsubdir: String subdir2 = nil;
```

Search for the file along the `path`.

For the subdirectories `subdir1` and `subdir2`, when not nil, the following attempts are made for a `dir` in the `path`: `dir`, `dir/subdir2`, `dir/subdir1`, and `dir/subdir1/subdir2`.

```
void
  remove String name;
```

Remove the file or directory with the `name`.

instance tom.File

variables

`public String name;`
: The name of the file.

`int flags;`
: If it is to be reopenend, these are the flags indicating how to do so.

methods

```
protected id
   init String n
  flags int f;
```

Designated initializer.

```
String
  directoryName;
```

Return the name of the directory containing our file.

```
id
  reopen;
```

Reopen the file according to the `flags`.

```
int
  type-of-file;
```

Return one of the `FILE_TYPE_*` constants for the receiving File. If the file is not open, the file is tested as if the file were open, i.e. following links.

```
OutputStream
  writeFields OutputStream s;
```

Undocumented. <h4>Positioning</h4>

```
long
  length;
```

Return the length of the file.

```
long
  position;
```

Return the current file position.

```
void
       seek long offset
  relative: int whence = STREAM_SEEK_SET;
```

Position the file ‘pointer’.

class tom.HashTable

Inherits

State supers: State, Keyed

variables

`const GOLDEN_BITS = -1640531527;`

Thirty-two bits by which to multiply a hashvalue to make all bits, more or less, significant.

This (unsigned) value is (1 `<<` 32) * frac (0.5 `+` 0.5 * sqrt (5)). The value of which the fractional part is taken is the golden ratio. (0x9e3779b9).

instance tom.HashTable

variables

`public int length;`

The number of stored objects.

`int size_shift;`

The 2log of the number of buckets.

`MutableObjectArray buckets;`

The buckets.

methods

```
id (self)
  init;
```

Designated initializer.

```
void
  empty;
```

Remove all elements from the table.

```
Any
  at All key;
```

Return the `key` if present, or `nil` otherwise.

```
Any
  member All object;
```

A different name for `at`.

```
Any
  member All key
   equal selector cmp;
```

Like `member`, but the elements are compared using the selector `cmp`.

```
Any
  memq All key;
```

Like `member`, but the element is identified on reference equality.

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
protected void
  adjust_length int inc
pre
  inc != 0;
```

Adjust the length of the hashtable, resizing if necessary.

```
protected void
  resize int new_shift;
```

Undocumented.

```
void
  resizing_add BucketElement elt;
```

Undocumented.

class tom.HashTableContainer

The HashTableContainer class is just a HashTable which knows how to mark its elements as a container. It is intended to be inherited by various class employing the HashTable class as a superclass for implementation reuse.

Inherits

State supers: HashTable, Container

instance tom.HashTableContainer

methods

```
void
  gc_container_mark_elements;
```

Undocumented.

class tom.MutableHashTable

Inherits

State supers: HashTableContainer, MutableKeyed

instance tom.MutableHashTable

methods

```
void
  add BucketElement elt;
```

Add `elt` to the receiving hashtable.

```
void
  remove BucketElement elt;
```

Remove `elt` from the receiving hashtable, if present.

class tom.HashTableEnumerator

Inherits

State supers: State, Enumerator

methods

```
instance (id)
  with Indexed b;
```

Undocumented.

instance tom.HashTableEnumerator

variables

`Indexed buckets;`
: The array the dictionary uses to store the buckets.

`int next;`
: The next bucket index we shall use.

`BucketElement elt;`
: The bucket element we're looking at.

methods

```
protected id
  init Indexed b;
```

Undocumented.

```
protected boolean
  next;
```

Update `elt` to point to the next bucket element.

class tom.Heap

Inherits

State supers: State, MutableCollection

instance tom.Heap

variables

`MutableArray elements;`
: The array used to store the elements.

`public int length;`
: The number of elements. This is equal to `[elements length]`.

`public mutable selector compare_selector;`
: The selector used to have two elements compare themselves. If this isn't set, `i_compare_r` will be used.

`boolean max_heap;`
: If this is `TRUE`, this is a heap of which the root node is the largest. Otherwise, the root is the smallest node.

methods

```
id
  init;
```

Invoke `[self init TRUE]`.

```
id (self)
  init boolean root_is_max;
```

Designated initializer.

```
boolean
  dump_simple_p;
```

Return `NO`.

```
Any
  extract_min
pre
  !max_heap;
```

Extract the root of the heap, which must be a heap storing the minimum value at its root.

```
Any (object)
  min
pre
  !max_heap;
```

Return the minimum value of the heap, which must be a heap storing the minimum value at its root. Return `nil` if the heap is empty.

```
Any
  extract_max
pre
  max_heap;
```

Extract the root of the heap, which must be a heap which stores the maximum value at its root.

```
Any (object)
  max
pre
  max_heap;
```

Return the maximum value of the heap, which must be a heap storing the maximum value at its root. Return `nil` if the heap is empty.

```
Any (object)
  extract_root
pre
  length > 0
post
  length >= 0 && [object heap_index] == 0;
```

Extract and return the root `object` of the heap.

```
Any (object)
  root
pre
  length > 0;
```

Return the root `object` of the heap, without extracting it.

```
void
  remove HeapElement elt
pre
  [elt heap_index] > 0 && elements[([elt heap_index] - 1)] == elt
post
  [elt heap_index] == 0;
```

Remove the `elt` from the receiving heap. The `elt` must be an element of the heap. <h4>MutableCollection</h4>

```
void
  add HeapElement object
pre
  [object heap_index] == 0
post
  length == 1 + old length;
```

Add the `object` to this Heap. It may not yet be part of any (other) heap, as stated by the precondition.

```
void
  addElementsFromEnumerator Enumerator e;
```

Undocumented.

```
void
  empty;
```

Undocumented. <h4>Enumerable</h4>

```
Enumerator
  enumerator;
```

Return an enumerator on the elements of this heap. Note that the order of the elements returned is undefined. <h4>Internal methods</h4>

```
protected void
  build_heap;
```

Build a heap from the elements already in `elements`.

```
int
  compare (Comparable, Comparable) (one, other);
```

Let `one` compare itself with the `other`, either using the `compare_selector`, or, if not set, the `int compare id` method.

```
protected void
  heapify int index;
```

Heapify from the node at index `i` (which is off by 1 compared to the index of the element in the `elements` array).

class tom.HeapElement

Inherits

State supers: State, Comparable

instance tom.HeapElement

variables

`public mutable int heap_index;`

The index of the element in the `Heap` it is stored. This index is 1 more than the index in the `elements` `Array` of the `Heap`. If this element is not part of a heap, this index is 0.

class tom.IntArray

Inherits

State supers: Array

methods

```
instance (id)
  with dynamic elements;
```
Undocumented.

instance tom.IntArray

methods

```
protected id
  initWithEnumerator Enumerator e;
```
Undocumented.

```
Any
  at int index;
```
Undocumented.

```
byte
  at int index;
```
Undocumented.

```
char
  at int index;
```
Undocumented.

```
int
  at int index;
```
Undocumented.

```
long
  at int index;
```
Undocumented.

```
float
  at int index;
```
Undocumented.

```
double
  at int index;
```

Undocumented.

```
int
  elementByteSize;
```

Undocumented.

```
int
  hash;
```

Undocumented.

```
(pointer, int)
  pointerToElements (int, int) (start, len);
```

Undocumented. <h4>Copying</h4>

```
class (State)
  mutableCopyClass;
```

Return the `MutableIntArray` class.

class tom.IntDictionary

A `IntDictionary` maps an `int` to an object reference.

Inherits

State supers: HashTable

instance tom.IntDictionary

methods

```
Any
  at int key;
```

Undocumented.

```
Enumerator
  enumerator;
```

Return a `valueEnumerator`.

```
Enumerator
  valueEnumerator;
```

Return an Enumerator on the values stored in this dict.

class tom.MutableIntDictionary

Inherits

State supers: IntDictionary, HashTableContainer

instance tom.MutableIntDictionary

methods

```
void
  freeze;
```

Undocumented.

```
void
  remove int key;
```

Remove the mapping for the `key`.

```
void
  set All value
   at int key
pre
  value != nil;
```

Associate the `value` with the `key`.

class tom.IntegerRangeSet

The `IntegerRangeSet` is good at holding ranges of integers. Ranges are stored in an unbalanced tree. The number of ranges dictates the memory usage by the set. Testing for membership is O(log n) where n is the number of ranges.

Inherits

State supers: State, Enumerable

instance tom.IntegerRangeSet

variables

`IntegerRangeSetNode root;`
: The root of the tree holding information.

methods

```
boolean
  add int v;
```
Add `v` to the receiving set. Return `FALSE` if it was already present; `TRUE` otherwise.

```
boolean
  equal id other;
```
Undocumented.

```
boolean
  equalIntegerRangeSetNode IntegerRangeSetNode other;
```
Undocumented.

```
id
  intersectionWith id other;
```
Return a new set being the intersection of the receiving set and the `other` set.

```
boolean
  isEmpty;
```
Undocumented.

```
boolean
  isSubsetOf id other;
```
Return `YES` if the receiving set is a subset of the `other` set.

```
(boolean, int) (non_empty, value)
  highestPresent;
```

Return the highest value present in the set.

```
(boolean, int) (non_empty, value)
  lowestPresent;
```
Return the lowest value present in the set.

```
boolean
  member int v;
```
Return `TRUE` iff the set contains `v`.

```
int
  nextNonPresent int i;
```
Return the smallest element `>=` `i` that is not yet in the set.

```
(boolean, int)
  nextPresent int i;
```
Return the smallest element `>` `i` that is contained in the tree, preceded by whether such an element actually is in the tree.

```
int
  previousNonPresent int i;
```
Return the largest element `<=` `i` that is not yet in the set.

```
(boolean, int)
  previousPresent int i;
```
Return the largest element `<=` `i` that is in the set.

```
boolean (b)
  remove int v;
```
Remove `v` from the set, returning `TRUE` if it was actually contained.

```
void
  shiftFrom int i;
```
Increase the value of all elements in the tree `>=` `i` by one.

```
(boolean, int)
  smallestElement;
```
Return the smallest value contained, preceded by whether we're not empty.

```
void
  uniteWith id other;
```
Modify the receiving set by adding the elements from the other set.

```
OutputStream
  write OutputStream s;
```
Undocumented.

```
Enumerator
  enumerator;
```

Undocumented.

```
protected id
  initWithEnumerator Enumerator e;
```

Undocumented.

class tom.IntegerRangeSetNode

Inherits

State supers: State

instance tom.IntegerRangeSetNode

variables

`id left;` The left and right subtrees.

`id right;`

`int offset;`
The offset from our parent.

`int size;` The number of integers in this node.

methods

```
boolean
  add int v;
```

Add `v` to the tree rooted at the receiving node. Return `FALSE` if it was already present; `TRUE` otherwise.

```
boolean
  equalIntegerRangeSetNode id other;
```

Undocumented.

```
id
  init (int, int) (o, s);
```

Designated initializer.

```
int
  highestPresent;
```

Return the highest value present in the set.

```
int
  lowestPresent;
```

Return the lowest value present in the set.

```
boolean
  member int v;
```

Return `TRUE` iff the set contains `v`.

```
int
  nextNonPresent int i;
```

Return the smallest element `>=` `i` that is not yet in the tree.

```
(boolean, int)
  nextPresent int i;
```

Return the smallest element `>` `i` that is contained in the tree, preceded by whether such an element actually is in the tree.

```
int
  previousNonPresent int i;
```

Return the largest element `<=` `i` that is not yet in the tree.

```
(boolean, int)
  previousPresent int i;
```

Return the largest element in the tree which is smaller than `i`.

```
(id, boolean)
  remove int v;
```

Remove `v` from the receiving tree, returning the modified tree, and `TRUE` if it was actually removed.

```
void
  shiftFrom int i;
```

Increase the value of all elements in the tree `>=` `i` by one.

```
int
  smallestElement;
```

Return the smallest value contained.

```
OutputStream
   write OutputStream s
  offset int i;
```

Undocumented.

```
protected (id, id, int, int)
  dissect;
```

Return the guts of this object.

```
protected (id, int)
  mergeLRL int v;
```

Merge the subtree rooted at this node, to accomodate the value `v`, returning the modified tree and the extra size for our parent node. Our parent actually holds the value `v` (as the first value).

```
protected (id, int)
  mergeRLL int v;
```

Merge the subtree rooted at this node, to accomodate the value `v`, returning the modified tree and the extra size for our parent node. Our parent actually holds the value `v` (as the last value).

```
protected void
  offset int n;
```

Adjust the `offset` by `n`.

```
protected void
  set_right id n;
```

Set the `right` node.

```
protected void
  setRightMost id r;
```

Set the right most node of the receiving tree to `r`.

class tom.IntegerRangeSetNodeEnumerator

Inherits

State supers: State, Enumerator

instance tom.IntegerRangeSetNodeEnumerator

variables

`IntegerRangeSetNode root;`
: The root of the tree of nodes.

`int previous;`
: The previous integer value retrieved.

methods

```
     id
       init IntegerRangeSetNode r;
```

Undocumented.

```
     (boolean, Number)
       next;
```

Undocumented.

```
     (boolean, int) (valid, value)
       next;
```

Undocumented.

class tom.Invocation

An `Invocation` is an object holding a target object, a selector, and arguments to the selector. Thus, an `Invocation` holds everything needed to send a message. An `Invocation` can be fired at its target (with the `fire` method), or fired after retargeting (using the `fireAt` method).

An `Invocation` is incomplete when not all arguments needed to send the message have been specified. An incomplete `Invocation` can be fired in two different ways. First, by invoking `fireWith` and supplying values for the remaining arguments. Second, by invoking on the `Invocation` a method completing it. For example, if an `Invocation` `x` of the method `void with int a do int b` only has a value for the argument `a`, then invoking `[x do 23]` will (temporarily) complete the `Invocation` and send the full message (`with do`) to its target.

Inherits

State supers: State

methods

```
instance (id)
   for selector sel
   to: All target = nil
  with dynamic arguments;
```

Create a, potentially incomplete, invocation.

```
instance (id)
  for selector sel
  to: All target = nil
    : Indexed arguments = nil;
```

Create a, potentially incomplete, invocation.

```
instance (id)
    of selector sel
   to: All target = nil
  with dynamic arguments;
```

Create a complete invocation. This raises a `program-condition` if the resulting invocation is incomplete.

```
instance (id)
   of selector sel
  to: All target = nil
    : Indexed arguments = nil;
```

Create a complete invocation. This raises a `program-condition` if the resulting invocation is incomplete.

```
instance (id)
     of selector sel
    to: All target = nil
  using pointer ap;
```

Create an invocation. It shall be complete. This method is primarily intended to be used by `Proxy` in its forwarding from `forwardSelector arguments`.

Arguments are to be retrieved from the `va_list` pointed to by `ap`, i.e. `va_arg (*ap, ...)`.

instance tom.Invocation

variables

`InvocationResult result;`
: The result of the most recent invocation, or `nil` if we haven't fired yet, or have fired with a void return type (of the fire method).

`pointer invocation;`
: The underlying invocation structure.

methods

```
protected id
  init pointer i;
```

Designated initializer.

```
boolean
  isComplete;
```

Return `TRUE` iff the receiving invocation is complete, i.e. is has all the arguments needed and can be fired directly with `fire` or `fireAt`.

```
selector
  selector;
```

Return this invocation's selector.

```
Any
  target;
```

Return this invocation's target. <h4>Forwarding</h4>

```
protected InvocationResult
  forwardSelector selector sel
        arguments pointer ap;
```

Forward the selector `sel` with the arguments pointed to by the `va_list` pointed to by `ap`. Return the result of the invocation.

Only the incoming arguments from *ap will be retrieved, so that subsequent va_arg invocations on *ap can retrieve the outgoing argument pointers.

This method is invoked by the runtime library in an attempt to forward a message not directly implemented by the receiver. This method is used since it is faster than a forwardInvocation.

```
boolean
  invocationp;
```

Return YES. <h4>Performing the invocation</h4>

```
InvocationResult
  fire;
```

Perform the invocation. If invoked repeatedly, the invocation will be performed repeatedly.

```
void
  fire;
```

Similarly, but avoid the creation of an InvocationResult. The result of the receiving invocation is set to nil.

```
InvocationResult
  fireAt All target;
```

Perform the invocation after setting the receiver of this invocation to target.

```
void
  fireAt All target;
```

Similarly, but avoid the creation of an InvocationResult. The result of the receiving invocation is set to nil.

```
InvocationResult
  fireWith dynamic arguments;
```

Perform the invocation resulting from completing this invocation with the arguments. Return the result. The receiving invocation will remain incomplete.

```
void
  fireWith dynamic arguments;
```

Similarly, but avoid the creation of an InvocationResult. The result of the receiving invocation is set to nil.

```
Any
  objectAfterFire;
```

Shortcut to fire and return the first element of the result as an object.

```
Any
  objectOfResult;
```

Shortcut to retrieve the first element of the result as an object. This fires if needed.

```
InvocationResult
  result;
```
If the invocation has been fired at least once, return the (most recent) result. Otherwise, fire and return the result.

```
TypeDescription
  resultTypeDescription;
```
Return the type description of the result from this invocation.

```
void
  encodeUsingCoder Encoder coder;
```
<h4>Coding</h4>

```
void
  initWithCoder Decoder coder;
```
Undocumented.

```
protected void
  encodeToCoder Encoder coder;
```
Private method to do the hard work of encoding all information carried by this invocation.

```
protected void
  decodeFromCoder Decoder coder;
```
Private method to do the hard work of decoding all information from the `coder` into the invocation. <h4>Garbage collection methods</h4>

```
void
  dealloc;
```
Deallocate the structure underlying the invocation.

```
void
  gc_mark_elements;
```
Mark the objects in this invocation's arguments.

class tom.InvocationResult

Inherits

State supers: State, Constants

methods

```
protected instance (id)
  with pointer result;
```

Return a freshly allocated instance with the `result`, which is a `struct trt_invocation_result`.

instance tom.InvocationResult

variables

`pointer values;`
: The actual result.

methods

```
protected id
  init pointer result;
```

Designated initializer.

```
dynamic
  components;
```

Retrieve all values from this result. The expected return type must fully match the actual return type.

```
dynamic
  component int n;
```

Retrieve the `n`th element from the result. The index of the first element is 0.

The expected return type must fully match, or can be an object, in which case numeric values are returned in a `Number` instance, and selectors in a `Selector`. Pointer values on a mismatch cause a condition to be signaled.

```
int
  length;
```

Return the number of elements in this result.

```
void
  setReturnValues (pointer, pointer) (first_value, extra_values)
      forSelector selector sel;
```

From the values held by this `InvocationResult`, update the `builtin_return_type` value pointed to by `first_value`, and any extra return value pointers as pointed to by the `va_list extra_values`.

```
TypeDescription
  typeDescription;
```

Return a the `TypeDescription` for this result.

```
OutputStream
  writeFields OutputStream s;
```

Output the elements in this result. <h4>Private methods</h4>

```
void
  dealloc;
```

Free the `values`.

```
void
  gc_mark_elements;
```

Mark the objects in this result.

```
void
  encodeUsingCoder Encoder coder;
```

<h4>Coding</h4>

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
protected void
  encodeToCoder Encoder coder;
```

Private method to do the hard work of encoding all information carried by this result.

```
protected void
  decodeFromCoder Decoder coder;
```

Private method to do the hard work of decoding all information from the `coder` into this result.

class tom.Lock

The abstract lock.

Inherits

State supers: State

instance tom.Lock

methods

```
deferred void
  lock;
```

Undocumented.

```
deferred void
  unlock;
```

Undocumented.

```
deferred boolean
  tryLock;
```

Try to lock, and return `YES` if it succeeded.

class tom.SimpleLock

A simple lock is a binary lock.

Inherits

State supers: Lock

instance tom.SimpleLock

variables

`pointer lock;`
The underlying lock.

methods

```
void
  dealloc;
```
Undocumented.

```
id
  init;
```
Designated initializer.

```
void
  lock;
```
Undocumented.

```
void
  unlock;
```
Undocumented.

```
boolean
  tryLock;
```
Undocumented.

class tom.RecursiveLock

A recursive lock is a binary lock which can be obtained multiple times by the same thread.

Inherits

State supers: Lock

instance tom.RecursiveLock

variables

`pointer lock;`
The underlying lock.

methods

```
void
  dealloc;
```
Undocumented.

```
id
  init;
```
Designated initializer.

```
void
  lock;
```
Undocumented.

```
void
  unlock;
```
Undocumented.

```
boolean
  tryLock;
```
Undocumented.

class tom.Semaphore

A semaphore is a lock which can be locked a number of times (1 for a binary semaphore) before the next attempt to lock will block.

When allocated first, the first `lock` will block.

Inherits

State supers: Lock

methods

```
instance (id)
  new int num;
```

Return a new `Semaphore`, the first `num` `lock` operations will succeed.

instance tom.Semaphore

variables

`pointer sem;`
Pointer to the underlying structure.

methods

```
void
  dealloc;
```

Undocumented.

```
id
  init int num;
```

Designated initializer.

```
id
  init;
```

Another initializer.

```
void
  lock;
```

Undocumented.

```
void
  unlock;
```

Undocumented.

```
boolean
  tryLock;
```

Undocumented.

class tom.MutableArray

Inherits

State supers: Array, MutableIndexed

methods

```
instance (id)
  withCapacity int cap;
```

Return a new instance of the receiving class which can hold `cap` elements without the need for resizing.

instance tom.MutableArray

variables

`public int capacity;`
The capacity of the array.

methods

```
void
  add All object;
```

Store the `object` at the end in the receiving array. If the receiving array stores unboxed values, the `object` is queried for its value.

```
(int, int)
  adjustMutableRange (int, int) (start, len);
```

Adjust the range (`start`, `len`) to fit the capacity of the receiving `MutableArray`. If `len` == -1, it is adjusted to fit the capacity.

```
void
  empty
post
  length == 0;
```

Empty the receiving array. This frees any storage used by the array to store its elements.

```
deferred id
  initWithCapacity int capacity;
```

Initialize the newly allocated receiving object to be able to hold `capacity` items without needing to resize.

```
id
  initWith int n
        at pointer addr;
```

Initialize with the indicated pointer and integer for contents and length. The `capacity` is set to the `length`.

```
deferred void
  insert All object
      at int index
pre
  index >= 0 && index <= length;
```

Insert the `object` at the `index`, shifting the objects at that or a higher index up by 1 position.

```
deferred Any
  removeAt int index;
```

Remove the element from `index`, decreasing the index of all elements after `index`, and return the element boxed. If the receiving array stores unboxed values, such as integers, the value returned is the element boxed.

```
void
  removeElementAt int index;
```

Remove the element from `index`, decreasing the index of all elements after `index`.

```
deferred void
  removeElements (int, int) (start, length);
```

Remove the `length` elements from `start`. If `length` == -1, all elements from `start` are removed.

```
void
  resize int to
pre
  to >= 0;
```

Adjust the size of the array, filling any newly created entries with the default value for the type (i.e. 0).

```
deferred void
  resize (int, int) (start, num);
```

Adjust the size of the array by inserting `num` new entries at `start` filling newly created entries with the default values for the type (i.e. 0).

```
void
  truncate int new_length
pre
  new_length >= 0;
```

Adjust the length of this array to `new_length`.

```
void
  bubbleSortUsingKey selector key
         comparator: selector cmp = selector (int compare All);
```

Bublesort the receiving array on the `key` of the contained elements by comparing them using the `compare` selector.

```
private void
  bubbleSort (selector, selector) (key, compare)
       range (int, int) (start, beyond);
```

Hard working bubblesort (also used by the quicksorter for the last bits).

```
void
  quickSortUsingKey selector key
        comparator: selector compare = selector (int compare All);
```

Quicksort the receiving array on the `key` of the contained elements by comparing them using the `compare` selector.

```
private void
  quickSort (selector, selector) (key, compare)
      range (int, int) (start, beyond);
```

The hard worker for `quickSortUsingKey comparator`. <h4>Copying</h4>

```
id
  initCopy;
```

In addition to what our `super` does, adjust our (new) `capacity` to fit our `length`.

class tom.MutableByteArray

Inherits

State supers: ByteArray, MutableArray, OutputStream

instance tom.MutableByteArray

methods

```
void
  add byte b;
```
Undocumented.

```
void
  freeze;
```
Undocumented.

```
id
  initWithCapacity int capacity;
```
Undocumented.

```
protected id
  initWithEnumerator Enumerator e;
```
Undocumented.

```
byte
  removeAt int index;
```
Undocumented.

```
Any
  removeAt int index;
```
Undocumented.

```
void
  removeElements (int, int) (start, length);
```
Undocumented.

```
void
  resize (int, int) (start, num);
```
Undocumented.

```
void
  set byte b
   at int index;
```

Undocumented.

```
void
  set char c
   at int index;
```

Undocumented.

```
void
  set All object
   at int index;
```

Undocumented.

```
void
  swap (int, int) (i, j);
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
void
  close;
```

Undocumented.

```
int
       readRange (int, int) (start, num)
  fromByteStream ByteStream f;
```

Undocumented.

```
int
      readRange (int, int) (start, num)
  fromByteArray ByteArray source
              to int position;
```

Undocumented.

```
void
  write byte b;
```

Undocumented.

```
int
  write byte b;
```

Undocumented.

```
int
  writeBytes int num
        from pointer address;
```

Undocumented.

```
int
  writeBytes int num
        from pointer address
          at int offset;
```

Undocumented.

class tom.Data

Inherits

State supers: MutableByteArray

instance tom.Data

class tom.MutableByteString

Inherits

State supers: ByteString, MutableString, MutableByteArray

instance tom.MutableByteString

methods

```
void
  freeze;
```
Undocumented.

```
ByteString
  frozen;
```
Undocumented.

```
protected id
  init (pointer, int) (p, num);
```
Undocumented.

```
id
  initCopy (pointer, int) (p, num);
```
In addition to what our `super` does, adjust our (new) `capacity` to fit our `length`.

```
void
  set char c
   at int index;
```
Set the byte at `index` to the character `c`, converted to a `byte` according to the `default_encoding`.

```
void
  add char c;
```
Add the byte encoding of the `char c` to this string.

```
ByteString
  substring (int, int) (start, len);
```
Override the `ByteString` implementation of this `substring` method, since that actually employs `ByteSubstring` objects which we can't use.

class tom.MutableCharArray

Inherits

State supers: CharArray, MutableArray

instance tom.MutableCharArray

methods

```
void
  add char c;
```
Undocumented.

```
void
  freeze;
```
Undocumented.

```
id
  initWithCapacity int capacity;
```
Undocumented.

```
protected id
  initWithEnumerator Enumerator e;
```
Undocumented.

```
char
  removeAt int index;
```
Undocumented.

```
void
  removeElements (int, int) (start, length);
```
Undocumented.

```
void
  resize (int, int) (start, num);
```
Undocumented.

```
void
  set char c
   at int index;
```
Undocumented.

```
CharNumber
  removeAt int index;
```

Undocumented.

```
void
  set All object
   at int index;
```

Undocumented.

```
void
  swap (int, int) (i, j);
```

Undocumented.

class tom.MutableCharString

Inherits

State supers: CharString, MutableString, MutableCharArray

instance tom.MutableCharString

methods

```
void
  freeze;
```

Undocumented.

```
protected id
  init (pointer, int) (p, num);
```

Undocumented.

class tom.MutableIntArray

Inherits

State supers: IntArray, MutableArray

instance tom.MutableIntArray

methods

```
void
  add int i;
```
Undocumented.

```
void
  freeze;
```
Undocumented.

```
id
  initWithCapacity int capacity;
```
Undocumented.

```
protected id
  initWithEnumerator Enumerator e;
```
Undocumented.

```
int
  removeAt int index;
```
Undocumented.

```
Any
  removeAt int index;
```
Undocumented.

```
void
  removeElements (int, int) (start, length);
```
Undocumented.

```
void
  resize (int, int) (start, num);
```
Undocumented.

```
void
  set int i
   at int index;
```

Undocumented.

```
void
  set All object
   at int index;
```
Undocumented.

```
void
  swap (int, int) (i, j);
```
Undocumented.

class tom.MutableObjectArray

Inherits

State supers: ObjectArray, MutableArray

methods

```
instance (id)
    with int num
  copies All object;
```

Return a newly allocated `MutableObjectArray` with `num` copies of the `object`.

instance tom.MutableObjectArray

methods

```
void
  freeze;
```

Undocumented.

```
id
  initWithCapacity int capacity;
```

Undocumented.

```
protected id
  initWithEnumerator Enumerator e;
```

Undocumented.

```
void
  insert All object
      at int index;
```

Undocumented.

```
Any
  removeAt int index;
```

Undocumented.

```
void
  removeElements (int, int) (start, length);
```

Undocumented.

```
void
  resize (int, int) (start, num);
```

Undocumented.

```
void
  set All object
   at int index;
```

Undocumented.

```
void
  swap (int, int) (i, j);
```

Undocumented.

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

class tom.MutableString

Inherits

State supers: String, MutableArray

instance tom.MutableString

methods

```
deferred void
  set char c
   at int i;
```

Undocumented.

class tom.Number

Inherits

State supers: State

methods

```
instance (id)
  with byte value;
```
Undocumented.

```
instance (id)
  with char value;
```
Undocumented.

```
instance (id)
  with int value;
```
Undocumented.

```
instance (id)
  with long value;
```
Undocumented.

```
instance (id)
  with float value;
```
Undocumented.

```
instance (id)
  with double value;
```
Undocumented.

instance tom.Number

methods

```
deferred byte
  byteValue;
```
Undocumented.

```
deferred char
  charValue;
```
Undocumented.

```
deferred int
  intValue;
```
Undocumented.

```
deferred long
  longValue;
```
Undocumented.

```
deferred float
  floatValue;
```
Undocumented.

```
deferred double
  doubleValue;
```
Undocumented.

```
int
  hash;
```
Undocumented.

```
boolean
  equal Number n;
```
Undocumented.

```
deferred int
  compare Number n;
```
Undocumented.

```
deferred int
  compare byte v;
```
Undocumented.

```
deferred int
  compare char v;
```
Undocumented.

```
deferred int
  compare int v;
```
Undocumented.

```
deferred int
  compare long v;
```
Undocumented.

```
deferred int
  compare float v;
```

Undocumented.

```
deferred int
  compare double v;
```

Undocumented.

```
deferred protected id
  init byte value;
```

Undocumented.

```
deferred protected id
  init char value;
```

Undocumented.

```
deferred protected id
  init int value;
```

Undocumented.

```
deferred protected id
  init long value;
```

Undocumented.

```
deferred protected id
  init float value;
```

Undocumented.

```
deferred protected id
  init double value;
```

Undocumented.

```
boolean
  dump_simple_p;
```

Undocumented.

class tom.ObjectArray

Inherits

State supers: Array

methods

```
instance (id)
  with dynamic elements;
```

Undocumented.

instance tom.ObjectArray

methods

```
protected id
  initWithEnumerator Enumerator e;
```

Undocumented.

```
Any
  at int index
pre
  index >= 0 && index < length
post
  length == old length;
```

Return the object at `index` in the receiving array.

```
boolean
  dump_self_p;
```

Return `TRUE`.

```
void
  dumpSelf MutableKeyed done
    indent MutableByteString prefix
    simple boolean allow_simple
     level int level
        to OutputStream s;
```

Dump the elements to the stream `s`.

```
int
  elementByteSize;
```

Undocumented.

```
(pointer, int)
  pointerToElements (int, int) (start, len);
```
Undocumented.

```
protected void
  setDuringConstruction (int, All) (index, object)
pre
  index >= 0 && index < length;
```
Set the `object` at the `index` in the receiving array, even if it is not a mutable array. This method must only be used during construction of a constant array object. <h4>Copying</h4>

```
class (State)
  mutableCopyClass;
```
Return the `MutableObjectArray` class.

```
id
    deepen int level
  mutably: boolean mutable_p = NO;
```
Deepen this copy.

```
void
  encodeUsingCoder Encoder coder;
```
<h4>Coding methods</h4>

```
void
  initWithCoder Decoder coder;
```
Undocumented. <h4>Garbage collection</h4>

```
redefine void
  gc_mark_elements;
```
This method is invoked by the garbage collector to have the receiving object mark the elements it references. An `ObjectArray` must reference the objects it holds.

class tom.Pointer

This `Pointer` class is a simple object wrapper around an object value.

Inherits

State supers: State

methods

```
int (result)
  hash pointer p;
```

Hash the pointer `p`. This hashes the pointer in a way similar to the `hashq` method of `All`.

```
instance (id)
  with pointer p;
```

Simple allocator.

instance tom.Pointer

variables

`public pointer value;`
Our value.

methods

```
id (self)
  init pointer p;
```

Designated initializer.

```
boolean
  dump_simple_p;
```

Return `YES`.

```
boolean
  equal Pointer other;
```

Compare our `value` with the `other`'s.

```
int
  hash;
```

Return the hashed pointer `value`.

```
OutputStream
  write OutputStream s;
```

Similar to a `Number`, a `Pointer` simply outputs the `value`.

class tom.PointerDictionary

A `PointerDictionary` maps a `pointer` to an object reference.

Inherits

State supers: EqHashTable, DictionaryContainer

instance tom.PointerDictionary

methods

```
Any
  at pointer key;
```

Undocumented.

class tom.MutablePointerDictionary

Inherits

State supers: PointerDictionary, MutableEqHashTable

instance tom.MutablePointerDictionary

methods

```
void
  remove pointer key;
```

Remove the mapping for the `key`.

```
void
  set All value
   at pointer key
pre
  value != nil;
```

Associate the `value` with the `key`.

class tom.Queue

Inherits

State supers: Indexed

instance tom.Queue

class tom.MutableQueue

Inherits

State supers: Queue, MutableIndexed

instance tom.MutableQueue

class tom.MutableObjectQueue

Inherits

State supers: MutableQueue, Conditions

instance tom.MutableObjectQueue

class tom.Random

Inherits

Behaviour supers: All

instance tom.Random

Inherits

Behaviour supers: All

methods

```
deferred int
  next;
```

Undocumented.

```
int (result)
  next int limit
pre
  limit > 1
post
  result >= 0 && result < limit;
```

Return a number in the range [0, limit).

class tom.MinimalRandom

Inherits

State supers: State, Random

variables

`const ia = 16807;`
Constants needed by the algorithm.

`const im = 2147483647;`

`const iq = 12773;`

`const ir = 2836;`

instance tom.MinimalRandom

variables

`int seed;` The seed from which we feed.

methods

```
id (self)
  init int s
pre
  s > 0;
```

Designated initializer.

```
id
  init;
```

Initialize this instance with a seed derived from the current moment in time and the `hashq` value of `self`.

```
int
  next;
```

Return the next random positive `int` value.

class tom.StandardRandom

Inherits

State supers: MinimalRandom

instance tom.StandardRandom

class tom.Selector

A `Selector` is an object wrapper for a selector.

Inherits

State supers: State

methods

```
instance (id)
  with selector sel;
```

Undocumented.

instance tom.Selector

variables

`selector sel;`
: The selector which we wrap.

methods

```
id (self)
  init selector s;
```

Designated initializer.

```
int
  hash;
```

Return the identity of the selector.

```
boolean
  equal id other;
```

Undocumented.

```
String
  name;
```

Return the name of the selector.

```
selector
  selector;
```

Return the selector we wrap. <h4>Coding methods</h4>

```
void
  encodeUsingCoder Encoder coder;
```

Encode the receiving `Selector` object. This writes the name of the selector as the means of identifying the selector upon decoding. (The selector itself can't be written since encoding a selector actually encodes a `Selector` object.)

```
void
  initWithCoder Decoder coder;
```

Initialize the receiving `Selector` from the `coder`.

class tom.Runtime

The Runtime class provides an interface to the functionality in the runtime library and other process related information.

Most variables of the `Runtime` class are not `public`, since they can be accessed by simply inheriting `Runtime`.

Inherits

State supers: Conditions, Constants, stdio

variables

`static String hostname;`
: The name of the host on which this program is running.

`static public String program_name;`
: The name under which this program was invoked (i.e. the basename of argv[0] in C).

`static String long_program_name;`
: The long program name (i.e. argv[0] in C).

`static Array arguments;`
: The arguments as passed to `main`.

`static Array all_arguments;`
: All the arguments, before any `load` method modified it.

`static Mapped environment;`
: The environment.

`static private ByteString main_resource_dir;`
: The directory holding the main resources, at least including the character encodings.

`static private MutableMapped classes_by_name;`
: The dictionary of classes. Mapping from name to array of classes with that name. Since this is created upon request and it is reset by dynamic loading, it is not publicly available. Access it through the `classes_by_name` method.

`static private int quit_inhibit;`
: Iff !0, quit (`SIGINT`, the user interrupt signal) is inhibited.

`static boolean quit_pending;`
: Iff !0, a `signal-int` will be raised when `quit_inhibit` and `panic_mode` again reach 0.

`static int panic_mode;`
: Iff !0, any signal received (excluding the interrupt signal), other condition signaled or raised, or any object thrown will simply cause an abort. This is used to protect critical sections in the runtime, such as during garbage collection or object allocation. If `panic_mode` is set, `quit_inhibit` is implicitly set too.

`static boolean core_on_fatal;`
: Iff `TRUE`, a corefile will be produced on fatal errors, such as uncaught condition raises.

`static int gc_alloc_since_partial;`
: The number of objects allocated since the last partial garbage collection run. In this respect, a partial run completing a full run is still considered a partial run.

`static int gc_alloc_since_total;`
: The number of objects allocated since the previous completed run. This excludes the objects counted by `gc_alloc_since_partial`; it is adjusted after a run is initiated, before the run is actually started.

`static boolean gc_stat_at_exit;`
: Iff `TRUE` statistics on memory usage and the garbage collector will be emitted upon exit.

`static boolean rt_stat_at_exit;`
: Iff `TRUE` output statistics on the runtime structures at exit.

`static boolean rt_num_inst_at_exit;`
: Iff `TRUE` output the number of live instances of each class at exit.

`static int gc_num_runs;`
: Statistics on the garbage collector and allocator, in order: the number of gc runs; the number of runs which complete a full run; the number of object allocated; the number of objects deallocated; the (real, i.e. elapsed) time spent protecting, marking, and sweeping; and the time spent in all of gc (this is the sum of the previous three, plus overhead).

`static int gc_num_complete;`

`static long gc_num_alloc;`

`static long gc_num_dealloc;`

`static double gc_total_protect;`

`static double gc_total_mark;`

`static double gc_total_sweep;`

`static double gc_total_all;`

`static int malloc_cur_bytes;`
: These numbers are only maintained if the runtime library was not instructed to not do so at compile time.

static int malloc_max_bytes;

static int malloc_cum_bytes;

static int malloc_cur_items;

static int malloc_max_items;

static int malloc_cum_items;

static int gc_inhibit;
: Iff !0, garbage collection won't be run. This is important during, for instance, enumerating a Container, since (most) enumerators can not handle the collection changing while they are enumerating.

static boolean gc_atomic;
: Iff TRUE, garbage collection will run atomically (with all its consequences), irrespective of the time constraint argument to garbageCollect. The default value is FALSE.

static boolean gc_full_at_exit;
: Iff TRUE all garbage will be cleaned upon exit. This is a debugging tool mostly.

static int gc_debug;
: The level of debugging garbage output by the garbage collector. Information is output to stderr stream provided by the C library. No information will be output if gc_debug is 0 or if the runtime was not compiled with the appropriate flags.

static int gc_partial_threshold;
: Threshold for gc_alloc_since_partial before a garbage collection run will be initiated. If gc_partial_threshold is 0 (the default value), garbage collection is never run implicitly.

static double gc_partial_time_limit;
: The time allowed for a partial garbage collection run when initiated by gc_alloc_since_partial exceeding gc_partial_threshold.

static int gc_total_threshold;
: When a partial garbage collection run is initiated and gc_alloc_since_total exceeds gc_total_threshold, the gc_partial_time_limit is ignored and instead the gc_total_time_limit is used. If gc_total_threshold is 0 (the default value), it is ignored.

static double gc_total_time_limit;
: The time limit used in case the condition described for gc_total_threshold applies.

static boolean preconditions_enabled;
: Iff TRUE, preconditions are checked.

static boolean postconditions_enabled;
: Iff TRUE, postconditions are checked.

static boolean rt_print_signals;
: Iff TRUE, unhandled signals are printed on [stdio err]. This is for debugging purposes.

methods

```
int
     start (All, selector) (object, sel)
  arguments Array arguments;
```

This method is invoked by the runtime library. Its main responsibility is to invoke the real main method, which is identified by the `sel` and `object`.

```
void
  exit int rc;
```

Normal level exit. Cleaning up will be performed.

```
void
  fastExit int rc;
```

Low level exit. Usual functionality for cleaning up is avoided.

```
void
  unhandledSignal Condition condition;
```

Output information on the unhandled signal `condition` on `[stdio err]`.

```
void
  willExit int rc;
```

Perform all things necessary for a clean exit. This runs the garbage collector if specified by a `:gc-exit`, dumps gc statistics if specified by `:gc-stat`, number of instances if specified by `:rt-inst`, and memory overhead information if specified by `:rt-stat`.

```
OutputStream
  help OutputStream s
  done MutableKeyed done;
```

Output help information about the facilities (most notably ':' arguments) offered by the receiving class, on the `OutputStream` s.

Any implementation should add itself to the set `done`, and check for presence before outputting anything, to avoid generating the same output for every subclass not overriding this method.

```
void
  load MutableArray arguments;
```

Scan the arguments to the program for something telling us whether or how to do certain things.

See the output of `:help` of any TOM program for short information on the options.

```
void
  preload MutableArray arguments;
```

Invoked by the runtime library before the first load is invoked.

This method is needed for two occasions: first is to check for `:help`. The reason this is not done in `load` is to be able to get some help before any negative side effects of any `load` method. The second reason is for finding `:rt-resource-dir`, which must be done before `ByteString`'s `load` method can play with its encoding.

```
void
  reportNumInstances OutputStream s
      includeZeroes: boolean zeroes = FALSE;
```

Output to the stream `s` the number of live instances for each class. If the optional `zeroes` is `TRUE`, classes with zero instances are included in the report. This includes deferred classes, as they cannot have any instances.

```
protected void
  runtimeStatistics OutputStream s;
```

Output the actual statistics for `:rt-stat` to the stream `s`.

<h4>Classes</h4>

```
Indexed
  classes;
```

Return the array of all class objects.

```
Mapped
  classes_by_name;
```

Return the, possibly created upon request, mapped collection of classes keyed on their name.

```
class (State) (class_object)
  classNamed String name;
```

Return the class with the `name`.

Name may be unqualified, as in `"Runtime"`, which will return the single class with that name, or `nil` in case such a class does not exist, or if more than one class with that name exists in multiple units.

The `name` may be qualified, as in `"tom.Runtime"`, in which case the `Runtime` class of the `tom` unit will be returned, if that unit and class within that unit exist. <h4>Selectors</h4>

```
selector
  selectorNamed String name;
```

Return the existing selector known by the `name`.

```
boolean
  selector selector s1
    equals selector s2;
```

Return `TRUE` iff the selectors `s1` and `s2` denote the same selector.

```
String
  nameOfSelector selector sel;
```

Return the name of the `selector`.

```
selector
  nullSelector;
```
Return the invalid selector. <h4>Garbage collection</h4>

```
void
  garbageCollect;
```
Run the garbage collector to the end of a full garbage collection run.

```
void
  garbageCollect double time
pre
  gc_atomic -> !time;
```
Run the garbage collector for at most `time` seconds.

```
void
  disableGC
pre
  gc_inhibit >= 0
post
  gc_inhibit == old gc_inhibit + 1;
```
Increase the `gc_inhibit`. This invocation should be matched by an invocation of `enableGC`.

```
void
  enableGC
pre
  gc_inhibit > 0
post
  gc_inhibit == old gc_inhibit - 1;
```
Decrease the `gc_inhibit`. <h4>Process information</h4>

```
Mapped
  environment;
```
Return the dictionary holding the process environment. The dictionary is filled upon the first request, thread-safely.

```
ByteString
  main_resource_dir;
```
Return the `main_resource_dir`.

```
void
  setenv (String, String) (environment_variable, value);
```
Set the `value` of the `environment_variable`, thread-safely.

```
String
  hostname;
```

Return the hostname of this machine. If the class variable is not set, it is set once from gethostname(2).

```
String
  tom_prefix;
```

Return the directory in which all TOM stuff has been installed. This returns the value of TOM_PREFIX in the Constants class.

<h4>Signal and condition handler</h4>

```
Any
  perror String prefix
     for All object
   class ConditionClass condition_class
   raise boolean not_signal;
```

Construct a Condition for the object with the condition_class and a message created from the (optional) prefix, plus the information available from the (ANSI C) errno variable. If not_signal is TRUE, the new condition is raised; otherwise it is signaled and the result is returned (if a return is allowed).

```
int
  quit_inhibit;
```

Accessor method for quit_inhibit which is private to the Runtime class to protect it against being mutated by subclasses but which can be freely read, hence this method.

```
void
  quit_disable;
```

Increase the quit_inhibit flag. Any increase should be accompanied later on by the corresponding decrease.

```
void
  quit_enable
pre
  quit_inhibit > 0;
```

Decrease the quit_inhibit flag, raising a postponsed signal-int if indicated by quit_pending.

```
int
  panic_mode;
```

Accessor method for panic_mode which is private to the Runtime class to protect it against being mutated by subclasses but which can be freely read, hence this method.

```
void
  panic_enable;
```

Increase the panic_mode flag. Any increase should be accompanied later on by the corresponding decrease.

```
void
  panic_disable
pre
  panic_mode > 0;
```

Decrease the panic_mode flag, raising a signal-int if requested by quit_pending.

instance tom.Runtime

The Runtime instance is totally empty.

class tom.Set

Inherits

State supers: HashTable, Keyed

instance tom.Set

methods

```
Any
  member All object;
```
Invoke `HashTable`'s implementation.

```
Any
  memq All object;
```
Invoke `HashTable`'s implementation.

```
Enumerator
  enumerator;
```
Return an enumerator on the receiving set.

class tom.MutableSet

Inherits

State supers: Set, MutableHashTable, MutableKeyed, Container

instance tom.MutableSet

methods

```
void
  add All object;
```

Undocumented.

```
void
  remove All object;
```

Remove `elt` from the receiving set, if present.

class tom.SetEnumerator

Inherits

State supers: HashTableEnumerator

instance tom.SetEnumerator

variables

`redeclare BucketSetElement elt;`

methods

```
(boolean, Any) (valid, object)
  next;
```

Undocumented.

class tom.State

Inherits

Behaviour supers: All

variables

`State isa;`
: Our class. The 'State' class is an instance of the State meta-class. The same is true for every other class.

`private int asi;`
: Information used, in cunning ways, by the runtime.

methods

```
instance (id)
  alloc;
```

Return a newly created instance of the receiving class. All values, apart from the `isa` will have been initialized to their default value.

```
boolean
  classp;
```

Return `TRUE`, since we're a class object.

```
boolean
  coding-permanent-object-p;
```

Return `YES`. This should not be changed; this method is used in situations where it is not known whether the object is a class or not. In other situations, where it is known to be a class, this method is not invoked as it is known to return `YES`.

```
boolean
  dump_simple_p;
```

Return `YES`.

```
class (id)
  kind;
```

Return the class of the receiving object, i.e. the value of `isa`.

```
ByteString
  name;
```

Return the name of this class.

```
int (n)
  num_instances;
```

Return the number of currently live direct instances of this class, non-transitively.

```
Unit (unit)
  unit
post
  unit != nil;
```

Return the unit of this class. The postcondition states that such a unit must exit.

```
instance (id)
  new;
```

Return a newly created and initialized instance of the receiving class.

```
OutputStream
  write OutputStream s;
```

Write a description of this class object to the stream `s`.

instance tom.State

Inherits

Behaviour supers: All

variables

`class (id) isa;`
Our class.

`private int asi;`
Information used, in cunning ways, by the runtime.

methods

```
boolean
  classp;
```

Return `FALSE`, as we're an instance, and not a class.

```
protected void
    dump MutableKeyed done
  indent MutableByteString prefix
  simple boolean allow_simple
   level int level
      to OutputStream s;
```

Hard worker for `dump`.

```
id
  init;
```

Designated initializer. Does nothing.

```
class (id)
  kind;
```

Return the class of the receiving object.

```
void
  set_kind class (State) a_class;
```

Change the class of the receiving object (i.e., the `isa`) into the `a_class`. Currently both the original and the new class must carry exactly the same state. Looser restrictions could be implemented...

```
OutputStream
  write OutputStream s;
```

Write the class and address of the receiving object to the stream `s`.

```
OutputStream
  writeFields OutputStream s;
```

Subsidiary for `write` to allow subclasses to write their fields to the stream `s`. The default implementation does nothing. <h4>Garbage collection</h4>

```
void
  dealloc;
```

Invoked by the garbage collector when an object has become garbage. Some important notes apply to this method:

Do not message any other objects from within this method as they might have become garbage too.

Since class objects can not become garbage, it is safe to message class objects.

When overriding this method, it is not necessary to invoke `State`'s implementation.

```
void
  gc_mark_elements;
```

This method is invoked by the garbage collector for instances which employ pointer typed instance variables, to have the receiving object mark the elements it references through said pointers. The default implementation marks the object referencing variables.

class tom.BufferedStream

Inherits

State supers: StreamStream, InputOutputStream, Conditions

variables

`const DEFAULT_BUFFER_SIZE = 8192;`
: The default value of the default size for the buffers of our instances.

`static int default_buffer_size;`
: The default size for the buffers of our instances. If anyone sets this to a negative number, s/he should be ni'd.

methods

```
void
  load MutableArray arguments;
```

Undocumented.

instance tom.BufferedStream

variables

`MutableByteArray buffer;`
: The buffer we use.

`int num;` The number of elements in the `buffer`.

`int next;` The index of the first character not yet handled (i.e. read or written).

methods

```
id
  init Stream s;
```

Initialize the newly allocated instance to buffer the stream `s` with a buffer sized the `default_buffer_size`.

```
id
        init Stream s
  bufferSize int cap;
```

Designated initializer. Initialize the newly allocated instance to buffer the stream `s` with a buffer sized `cap`.

```
int
  peek;
```

Return the value of the next byte to be returned by `read`, or -1 upon an error or end-of-file. This does not actually read the byte.

```
void
  unget byte b;
```

Stuff the byte `b` back (sort-of) into the stream. It will be the next byte to be read. <h4>`OutputStream` methods</h4>

```
id
  flushOutput;
```

Flush any bytes buffered to the stream this instance is buffering.

```
void
  write byte b;
```

Write the byte `b`, raising a `stream-error` on error.

```
int
  write byte b;
```

Write the byte `b`, returning 1 upon success.

```
int
  writeBytes int length
        from pointer address;
```

Write to this stream the `length` bytes residing in memory at `address`. <h4>`InputStream` methods</h4>

```
byte
  read;
```

Return the next byte, raising a `stream-eos` upon an error or end-of-file.

```
int
  read;
```

Return the value of the next byte read, or -1 upon an error or end-of-file.

```
int (num_read)
  readRange (int, int) (start, length)
       into MutableByteArray destination;
```

Read at most `length` bytes into the `destination`, writing them from `start`. Return the number of bytes actually read. <h4>Internal methods</h4>

```
protected int
  readBuffer;
```

Fill the buffer by reading more bytes from the `stream`. Return the number of bytes read.

```
protected int
  writeBuffer;
```

Write any bytes needing to be output to the `stream`. Return the number of bytes written.

class tom.StringStream

A `StringStream` is sort-of an enumerator on a `String`, with a `InputStream` interface.

Inherits

State supers: State, InputStream

methods

```
instance (id)
  with String string;
```

Return a new stream on the `string`.

instance tom.StringStream

variables

`String string;`
: The `String` we're streaming.

`int next;` The index of the next byte to read.

methods

```
protected id
  init String s;
```

Designated initializer.

```
byte
  read;
```

Undocumented.

```
int
  read;
```

Undocumented.

```
int
  readRange (int, int) (start, num)
       into MutableByteArray buffer;
```

Read the range `(start, num)` from the string into the `buffer`.

class tom.String

Inherits

State supers: Indexed, Comparable

instance tom.String

methods

```
deferred redeclare String
  frozen;
```
We'll return a `String` when frozen.

```
boolean
  dump_simple_p;
```
Return `YES`.

```
OutputStream
  dump_simple OutputStream s;
```
Print the receiving string, quoted.

```
deferred redeclare boolean
  equal String other;
```
Compare the receiving `String` with the other `String`.

```
deferred boolean
  equalByteString ByteString other;
```
Compare the receiving `String` with the other `ByteString`.

```
deferred boolean
  equalCharString CharString other;
```
Compare the receiving `String` with the other `CharString`.

```
deferred boolean
  equalUniqueString UniqueString other;
```
Compare the receiving `String` with the other `UniqueString`.

```
boolean
  equalModuloCase String other;
```
Compare the receiving `String` with the other `String`, ignoring case differences.

```
int
  compare id other;
```

Compare the receiving `String` with the other.

```
(int, int)
  rangeOfString String string
         range: (int, int) (start, len) = (0, -1);
```

Return the range of the occurrence of the `string` in the receiving string. Return a negative length in case it could not be found. The optional `start` and `length` can be specified to restrict the searching within the receiving string.

```
MutableArray
  componentsSeparatedBy char c
                 limit: int limit = -1
          excludeEmpty: boolean excl = NO
     substringSelector: selector sel = selector (String substring (int, int));
```

Return a (mutable) `Array` of strings, taken from the receiving string by splitting it at characters with the indicated `char` value. Thus, splitting '/usr/tmp' at each '/' returns an array holding the empty string, 'usr', and 'tmp'.

The optional argument `limit` specifies the maximum number of items in which the caller is interested, or -1 for all items. For example, if '/usr/foo/bar' is split on '/' in 3 items, the array returned contains '', 'usr', and 'foo/bar'.

The optional argument `excl`, if YES specifies that zero-length substrings are not to be included in the result. Thus, splitting '/aap/noot/mies/wim' in 3 items, ignoring empty items, returns an array containing 'aap', 'noot', and 'mies/wim'.

The optional selector `sel` specifies the method to be called to extract the substrings from the receiving string. The default selector is `"r_substring_(ii)"`. To retrieve mutable substrings, the selector `"r_mutableSubstring_(ii)"` could be used.

```
deferred MutableString
  mutableSubstring (int, int) (start, len)
pre
  start >= 0 && len >= -1;
```

Return a `MutableString` holding the characters from the receiving `String` in the (clipped) range (`start`, `len`).

```
deferred String
  substring (int, int) (start, len)
pre
  start >= 0 && len >= -1;
```

Return a constant `String` holding the characters from the receiving `String` in the (clipped) range (`start`, `len`).

```
deferred UniqueString
  uniqueString;
```

Return a unique version of the receiving string. Do not use this method to create unique strings; use `[UniqueString with my_string]` instead. (This method only creates strings which think they are unique; the `UniqueString` class ensures they actually are.)

```
id
  downcase;
```

Undocumented.

```
id
  upcase;
```

Undocumented. <h4>Conversions</h4>

```
double (value)
  doubleValue;
```

Return the double value at the start of the string.

```
(int, boolean, int) (value, full_range, actual_length)
    integerValue (int, int) (start, len)
    defaultBase: int base = 10
      allowSign: boolean signs = YES
    allowCBases: boolean c_bases = YES
  baseSeparator: byte base_separator = '_'
    decimalBase: boolean decimal_base = YES;
```

Convert the number contained in the receiving string from index `start`, running for `len` bytes (which -1 for unlimited length).

The value returned is a tuple (extracted value, occupied full range, actual length). If the actual length is 0, the extracted number will be 0.

The `base` defaults to 10, but can be any number. If it is larger than 10, alpha characters encountered have the value of 11 + the offset from the alpha character to the start of its range. Thus, 'a' is 10, 'z' is 35.

Iff `signs`, a leading '+' or '-' sign is accepted.

Iff `c_bases`, C-style base indicators may be used: a number starting with a '0' denotes an octal number; a number starting with '0x' or '0X' is a hexadecimal number.

Iff the `base_separator` is not 0, a number can be prefixed with a base indication followed by the base separator to specify the base of the actual number to follow. The base is read using the a decimal `base`, unless `decimal_base` is `FALSE`, in which case the base is read in the default `base`. Thus, '10_10', with '_' as a base separator, returns `base` if `decimal_base` is `FALSE`, and 10 if it was `TRUE`.

```
int
  intValue;
```

Simple front-end for `integerValue` (with default arguments).

```
int
  unsignedIntValue;
```

Simple front-end for `integerValue`, similar to `intValue`, but not allowing a negative value. For a negative value entered (due to `integerValue` not doing overflow checking), 0 is returned.

```
boolean
  isAlpha char c;
```
Return TRUE iff the character c denotes a letter.

```
boolean
  isDigit char c;
```
Return TRUE iff the character c is a digit.

```
boolean
  isLower char c;
```
Return TRUE iff the character c is in lower-case.

```
boolean
  isPunct char c;
```
Return TRUE iff the character c is a punctuation character.

```
boolean
  isSpace char c;
```
Return TRUE iff the character c is a space character.

```
boolean
  isUpper char c;
```
Return TRUE iff the character c is in upper-case.

```
char
  toLower char c;
```
Undocumented.

```
char
  toTitle char c;
```
Undocumented.

```
char
  toUpper char c;
```
Undocumented.

```
int
  digitValue char c;
```
Undocumented.

```
int
  alphaValue char c;
```
Undocumented.

```
id
  stringByDecoding String encoding_name;
```

Return a string by decoding it assuming it was encoded using the encoding named by `encoding_name`. The default implementation simply returns `self`.

class tom.Thread

The `Thread` class provides an abstraction to the multi-threading facilities provided by the underlying operating system.

A new thread is started by the `performInThread with` method provided by the instance `All`. The value returned by that method is the `Thread` object of the newly created thread.

Every thread has an id. The id of the current thread is available from the `Thread` class as `current_id`. The main thread (which every program has, even when running single-threaded) has id 0. Due to the differences in target implementations, exiting the main thread, by invoking that `Thread`'s `exit` method, is not guaranteed not to exit the program.

Multi-threading need not be available on all TOM targets. The `functioning` method returns `FALSE` on those targets on which multi-threading is not available. However, a TOM programmer can assume multi-threading to always be available.

Inherits

State supers: State

variables

`static MutableEqSet threads;`
: The currently existing threads.

`local static public instance (id) current;`
: The current thread.

`local static public int current_id;`
: The id of the current thread.

methods

```
boolean
  functioning;
```

Return `TRUE` iff we can run multiple threads on this target.

```
Set
  threads;
```

Return the currently existing threads.

instance tom.Thread

variables

`public int thread_id;`
: The TOM thread id of this thread.

methods

```
void
  exit int rc
pre
  self == [#<ClassReference 002aba88 Thread.t:68 type=Thread> current];
```

Exit the receiving thread, which must be the current thread.

```
protected id (self)
  init int th_id;
```

Designated initializer.

class tom.Trie

The `Trie` is a class providing the mechanism to store information in a trie on `char` strings. It does not by itself store any information, subclasses should be created to hold the information. The `Trie` is accompanied by its subclass `ObjectTrie` which can store objects in a trie.

Inherits

State supers: State, Constants

instance tom.Trie

variables

`int start;`
: The offset to the first element in `next`, i.e. the element with numeric value `start` resides at index 0 in `next`.

`int beyond;`
: The value of the first element beyond the last element in `next`.

`Any next;`
: If `start == beyond`, this is the suffix `String` which leads up to the value this node holds (if any). Otherwise, `start > beyond` and this is a `MutableObjectArray` pointing to the next nodes, and which is to be indexed with offset `start`.

methods

```
deferred boolean
  isEmpty;
```

Return `YES` iff we can hold a value, i.e. if we do not yet hold a value.

```
protected id
  createNode String str
       start int s
         end int e
     options int options;
```

Create the node for that part of the string `str` starting at `s`, and ending at `e`. The `TRIE_LOOKUP_PREFIX` option is ignored. If this node already exists, it is returned.

When the `options` include `TRIE_FOLD_CASE`, the `str` is inserted in lower case.

```
protected id
  findNode String str
     start int s
       end int e
   options int options;
```

Find the node for that part of the string `str` starting at `s`, and ending at `e`. Iff a prefix match is desired, the node returned is the longest prefix match.

```
protected void
  pushSuffix int options;
```

Push our suffix one node down.

```
OutputStream
  write OutputStream s;
```

Undocumented.

```
deferred OutputStream
  writeValue OutputStream s;
```

Undocumented.

class tom.ObjectTrie

An `ObjectTrie` is a `Trie` which can hold an object.

Inherits

State supers: Trie

instance tom.ObjectTrie

variables

`public mutable Any value;`
Our value.

methods

```
boolean
  isEmpty;
```

Undocumented.

```
void
  pushSuffix int options;
```

Move our value with the suffix.

```
Any
       at String key
  options int options;
```

Undocumented.

```
void
      set All object
       at String key
  options int options;
```

Undocumented.

```
OutputStream
  writeValue OutputStream s;
```

Undocumented.

```
Any
  at String key;
```

<h4>MutableMapped collection</h4>

```
void
  set All object
   at String key;
```

Undocumented.

class tom.TypeDescription

Inherits

State supers: State

variables

`static MutablePointerDictionary descriptions;`
A container holding the mapping from `struct trtd_selector_args` to a `TypeDescription` instance.

methods

```
instance (id) (result)
  for pointer args;
```

Designated allocator, using a cache.

instance tom.TypeDescription

variables

`public pointer types_description;`
The internal runtime structure to the type description.

methods

```
boolean
  equal id other;
```

Undocumented.

```
protected id (self)
  init pointer args;
```

Designated initializer.

```
int
  length;
```

Return the number of elements.

```
int
  component int n;
```

Describe the element at `n`, indexed 0. This returns one of the `TYPEDESC_* Constants`.

```
  OutputStream
    writeFields OutputStream s;
```

Describe the component types.

class tom.Unicoding

The `Unicoding` class object maintains information on the `Unicode` character coding.

Inherits

Behaviour supers: All

variables

`static ByteArray is_digit;`
Bitmap for digit predicate.

`static ByteArray is_letter;`
Bitmap for letter predicate.

`static ByteArray is_lower;`
Bitmap for lower predicate.

`static ByteArray is_punct;`
Bitmap for punctuation predicate.

`static ByteArray is_space;`
Bitmap for space predicate.

`static ByteArray is_upper;`
Bitmap for upper predicate.

methods

```
protected ByteArray
  loadPredicateSet String predicate
       alternative selector alt_sel;
```

Load and return the predicate set for the `predicate` on Unicode characters. If it can not be located, the `alt_sel` is used to extract part of the information needed from the `USASCIIEncoding`.

```
boolean
  isAlpha char c;
```

Undocumented.

```
boolean
  isDigit char c;
```

Return `TRUE` iff the character `c` is a digit according to the encoding of the receiving string.

```
boolean
  isLower char c;
```

Undocumented.

```
boolean
  isPunct char c;
```
Undocumented.

```
boolean
  isSpace char c;
```
Undocumented.

```
boolean
  isUpper char c;
```
Undocumented.

```
char
  toLower char c;
```
Undocumented.

```
char
  toTitle char c;
```
Undocumented.

```
char
  toUpper char c;
```
Undocumented.

```
int
  digitValue char c;
```
Undocumented.

```
int
  alphaValue char c;
```
Undocumented.

instance tom.Unicoding

Inherits

Behaviour supers: All

class tom.Unit

Inherits

State supers: State

variables

`static MutableDictionary units;`
: All units known.

methods

```
instance (id)
  named String name;
```

Return the `Unit` with the `name`, or `nil` if said unit does not currently exist.

```
protected void
  fillUnits
pre
  !units;
```

Create the `units` dictionary and fill it with the currently known units.

```
Mapped
  units;
```

Return the collection of units, keyed on their name.

instance tom.Unit

variables

`public String name;`
: The name of this unit.

`Dictionary classes;`
: The classes in this unit, keyed on their unqualified name.

methods

```
protected id (self)
  initWithName String n
       classes Dictionary c;
```

Designated initializer.

```
class (State)
  classNamed String name;
```

Return the class with the given unqualified `name`, or `nil` if a class with that name does not exist in this unit.

class tom.XLTokens

The tokens available from XL.

variables

```
const XLT_PAR_CLOSE = -9;
const XLT_PAR_OPEN = -8;
const XLT_DOUBLE = -7;
const XLT_FLOAT = -6;
const XLT_LONG = -5;
const XLT_INT = -4;
const XLT_SYMBOL = -3;
const XLT_EPSILON = -2;
const XLT_EOF = -1;
```

instance tom.XLTokens

class tom.XL

Inherits

State supers: State, XLTokens

variables

`const XLS_SYMBOL = 0;`
: Different states of the lexer state machine. Basically, these states are the states of reading a floating point number, with a prefix for an integer, and an escape for a non-numeric input.

`const XLS_SIGN = 1;`

`const XLS_INT = 2;`

`const XLS_DOT = 3;`

`const XLS_FRAC = 4;`

`const XLS_EXP_E = 5;`

`const XLS_EXP_SIGN = 6;`

`const XLS_EXP = 7;`

instance tom.XL

variables

`public InputStream stream;`
: The stream being lexed.

`MutableByteString buffer;`
: The buffer used for building the text of the token.

`public long int_value;`
: The most recent integer value retrieved.

`public double float_value;`
: The most recent floating value retrieved.

`public int current_line;`
: The current line.

`public int token;`
: The current token.

`int next_char;`
: The next character, i.e. the first character of the next token. This is `XLT_EOF` for end of stream, or `XLT_EPSILON` if this should be considered invalid (and read before starting the next token).

methods

```
id
  initWithStream InputStream s;
```

Designated initializer.

```
int
  intValue;
```

Return the `int_value` as an int. Any loss of bits is not remarked.

```
MutableString
  matched;
```

Return the matched text.

```
int
  nextToken;
```

Skip space and return the next token.

```
(pointer, int) (contents, length)
  readBytes int expected_length
post
  length == expected_length;
```

Skip whitespace, read a quoted string of bytes (`"quoting \\like\\ \"this\""`) and return it. The length should match the expected length. Anything unexpected results in the return of a `NULL` pointer.

class tom.StreamEncoder

Inherits

State supers: Encoder

instance tom.StreamEncoder

variables

`OutputStream stream;`

The stream to which we write.

methods

```
id
  initWithStream OutputStream s;
```

Undocumented.

```
protected State
  replacementObjectFor State object;
```

Undocumented.

class tom.StreamDecoder

Inherits

State supers: Decoder

instance tom.StreamDecoder

variables

`InputStream stream;`
The stream from which we read.

methods

```
id
  initWithStream InputStream s;
```

Designated initializer.

class tom.BinaryStreamEncoder

Inherits

State supers: StreamEncoder, BinaryEncoder

instance tom.BinaryStreamEncoder

methods

```
id
  initWithStream OutputStream s;
```
Designated initializer.

```
void
  finishEncodingRoot All object;
```
Finish the graph.

```
protected void
  writeByte byte b;
```
Undocumented.

```
protected void
  writeBytes (int, int) (start, length)
        from ByteArray r;
```
Undocumented.

```
protected void
  writeBytes (pointer, int) (address, length);
```
Undocumented.

class tom.BinaryStreamDecoder

Inherits

State supers: StreamDecoder, BinaryDecoder, C

instance tom.BinaryStreamDecoder

methods

```
id
  initWithStream InputStream s;
```

Designated initializer.

```
protected byte
  readByte;
```

Undocumented.

```
protected void
  readBytes int num
         to pointer address;
```

Undocumented.

class tom.TextStreamEncoder

Inherits

State supers: StreamEncoder

instance tom.TextStreamEncoder

methods

```
void
  startEncodingRoot All object;
```
Output the top of the graph.

```
void
  finishEncodingRoot All object;
```
Finish the graph.

```
class (State)
  startEncoding State object;
```
Output the start of the `object`.

```
void
  finishEncoding State object;
```
Finish the output of the `object`.

```
protected int
  identityForClass class (State) a_class;
```
Identify this class on the output `stream`, reporting its coding version.

```
void
  encodeNilObject;
```
Output `()`, which is the notation for the `nil` object.

```
void
  encodeReference int v;
```
Undocumented.

```
void
  encode boolean v;
```
Undocumented.

```
void
  encode byte v;
```

Undocumented.

```
void
  encode char v;
```
Undocumented.

```
void
  encode int v;
```
Undocumented.

```
void
  encode long v;
```
Undocumented.

```
void
  encode float v;
```
Undocumented.

```
void
  encode double v;
```
Undocumented.

```
void
  encode selector v;
```
Undocumented.

```
void
  encodeBytes (int, int) (start, length)
         from ByteArray r;
```
Undocumented.

class tom.TextStreamDecoder

This class is unimplemented.

Inherits

State supers: StreamDecoder, XLTokens

instance tom.TextStreamDecoder

variables

`XL lexer;` The lexer actually doing the reading from our `stream`.

`int token;`
: The current token, cached so we know when we are starting up, in which case the token is `XLT_EPSILON`.

methods

```
id
  initWithStream InputStream s;
```

Designated initializer of our super.

```
id
  initWithLexer XL l;
```

Designated initializer.

```
Any
  decode;
```

Undocumented.

```
byte
  decode;
```

Undocumented.

```
boolean
  decode;
```

Undocumented.

```
char
  decode;
```

Undocumented.

```
int
  decode;
```

Undocumented.

```
long
  decode;
```
Undocumented.

```
float
  decode;
```
Undocumented.

```
double
  decode;
```
Undocumented.

```
(pointer, int) (contents, length)
  decodeBytes;
```
Undocumented.

```
protected void
  declareClass;
```
Undocumented.

```
protected int
  nextToken;
```
Undocumented.

```
protected Any
  readReference;
```
Undocumented.

```
protected void
  skipList;
```
Read tokens up to and including the first top-level close parenthesis.

```
protected void
  termSymbol String name;
```
Undocumented.

```
protected void
  termToken int t;
```
Undocumented.

class tom.Comparable

Inherits

Behaviour supers: All

instance tom.Comparable

Inherits

Behaviour supers: All

methods

```
deferred int
  compare id other
pre
  other != nil;
```

Return 0 if the `other` is considered equal by the receiving object. 1 if the receiver considers himself larger, and -1 when smaller.

class tom.Container

A container is an object which gets to mark its elements after normal marking has been done. This is very usable for unique string tables, DO proxies, etc; actually: all cases where an object having become garbage implying it should be removed from its container, and the container itself is not allowed to reference the object in a normal way (since then it would never become garbage).

In short, a the combo of garbage collector and container implements weak referencing.

instance tom.Container

methods

```
deferred void
  gc_container_mark_elements;
```

The container mark method.

```
boolean
  isContainer;
```

Return `TRUE` iff the receiving object is a container.

```
void
  setIsContainer boolean container_p;
```

Set this object to be a container, or not, depending on `container_p`.

class tom.Copying

The `Copying` class defines an interface to copying objects.

`Copying` inherits from `State` since class objects should not be copyable. Inheriting from `State` ensures that the `Copying` instance methods can not be inherited by class objects.

Inherits

State supers: State

instance tom.Copying

methods

```
id
  copy;
```
Return a shallow copy of the receiving object.

```
id
  deepCopy;
```
Return a deep copy of the receiving object.

```
id
     deepen int level
  mutably: boolean mutable_p = NO;
```
Intended to be called on a recently acquired copy of an object, `deepen mutable:` deepens the copy. Iff the optional `mutable_p` is `TRUE`, the deepened copies will also be mutable. The default implementation does nothing.

The `level` should be less than 0 for an infinite deepen. `length == 0` is a nop; iff `length > 0`, every element of the copy is copied and deepened with `level - 1`.

The value returned is `self`.

```
id
  initCopy;
```
Initialize the receiver just after it has been created as the result of a `copy`. The default implementation does nothing but return `self`.

```
id
  initAsCopyOf All other;
```
Initialize the receiver just after it has been created as the result of a `mutableCopy` of the `other` object. The default implementation does nothing but returning `self`.

```
Any
  mutableCopy;
```

Return a mutable (shallow) copy of the receiving object. For objects which do not discern between mutable and immutable variants, the default implementation returns `[self copy]`.

Mutable copying asks the receiving object for its `mutableCopyClass`. If this class is `isa`, `self` is sent a `copy`. Otherwise, an instance of the class is allocated and sent an `initAsCopyOf`.

```
class (State)
  mutableCopyClass;
```

Return the class of the object resulting from a mutable copy of this object. The default implementation simply returns `isa`.

class tom.Enumerable

Inherits

State supers: State

methods

```
instance (id)
  withEnumerable Enumerable other;
```

Invoke `self`'s `withEnumerator` with an `enumerator` from the `other`.

```
instance (id)
  withEnumerator Enumerator e;
```

Return a newly allocated instance of the receiving class, filled with the elements from the `Enumerator e`.

instance tom.Enumerable

methods

```
deferred protected id
  initWithEnumerator Enumerator e;
```

Initialize with the elements from the `Enumerator e`.

```
deferred Enumerator
  enumerator;
```

Return an `Enumerator` on the receiving object.

class tom.Enumerator

instance tom.Enumerator

methods

```
deferred (boolean, Any)
  next;
```

Return a tuple containing the next object, preceded by a boolean value indicating whether the end of the enumerable has been reached; if the boolean is `TRUE`, the end has not yet been reached.

```
(boolean, byte)
  next;
```

Default implementations for direct value retrieving enumerators.

```
(boolean, char)
  next;
```

Undocumented.

```
(boolean, int)
  next;
```

Undocumented.

```
(boolean, long)
  next;
```

Undocumented.

```
(boolean, float)
  next;
```

Undocumented.

```
(boolean, double)
  next;
```

Undocumented. This extension of `State` defines the functionality for encoding and decoding objects. To be able to encode an object, it must at least properly implement `encodeUsingCoder`. Similarly, to be decodable, it must implement `initWithCoder`.

The 'unit of archiving' is a class, not an extension. This means that if an extension adds state information which needs to be archived (or encoded onto a `too.PortCoder`), the extension must re-implement the coding methods.

class tom.State (Coding)

Return the current version of the class `cls`. This is the version that will be written when coding instances of this class or a subclass thereof. The default version is 0.

A version should only be returned if `self` is identical to the class containing the method definition, i.e. the method is not inherited. Otherwise, the two are unequal, and the version of a subclass is requested that does not implement this method, and hence should return version 0.

methods

```
int
  version;
```

Undocumented.

```
boolean
  persistent-coding-p;
```

Return `YES`. <h4>Encoding methods</h4>

instance tom.State (Coding)

methods

```
class (State)
  classForCoder Encoder coder;
```

Return the class to be put in the coded stream as the class of this object. The default implementation simply returns `isa`, which is the receiving object's class.

```
void
  encodeUsingCoder Encoder coder;
```

Encode the receiving object to the target `coder`. Every object should first invoke this method of all its direct superclasses before encoding its instance variables, but only if `hasBeenCodedFor` for the class implementing the method returns `FALSE`.

```
boolean
  persistent-coding-p;
```

Return `NO`.

```
Any (self)
  replacementForStreamCoder StreamEncoder coder;
```

Return the object to be encoded on the `StreamEncoder coder` (i.e. archived or wired) instead of the receiving object. The default implementation simply returns `self`. <h4>Decoding methods</h4>

```
void
  initWithCoder Decoder coder;
```

Initialize the receiving object from the `coder`. After verifying that this method implementation has not yet been invoked (using `hasBeenCodedFor`), this method should invoke the implementation of this method by the superclasses, followed the fields that were encoded by this class. Decoding must be done in the same order as encoding.

Note that this method returns `void`. An object can change the actual object returned from decoding by implementing `awakeAfterUsingCoder`.

```
id (self)
  awakeAfterUsingCoder Decoder coder;
```

Return the object to be the object retrieved from decoding instead of the receiving object. The default implementation returns `self`.

Objects can use this method to return their administered counterpart, like `UniqueString` objects do.

Note that if an object is referenced during its decoding (i.e. object A is referenced by an object B which is decoded because B is (indirectly) referenced by A), it must not return a different object from `awakeAfterUsingCoder`. If it does, a `coding-condition` is raised.

class tom.Coder

Inherits

State supers: State, Conditions

methods

```
int
  version;
```

The version of the coding scheme used. The current version is 0.

instance tom.Coder

class tom.BinaryCoder

The BinaryCoder classes BinaryEncoder and BinaryDecoder can archive dearchive a graph of objects in a binary form onto/from a stream. The format is rather simple: Every item stored is preceded by a tag byte indicating what the next item is. There are a few secondary tags to introduce classes, etc.

Every instance or class written is internally numbered in the order the objects are written. References to these objects are encoded in the number of bytes necessary for the number of currently known objects. The secondary tags 2 and 4 switch to 2 and 4 byte reference encoding, respectively.

The nil object is denoted by the 0 tag.

Selectors are encoded as a tag S, followed by the assigned selector number (which is an int, starting at 1) and the corresponding Selector object. Selectors already encoded are denoted by a tag s and the int selector number. The invalid selector (the default value of selector typed variables, also available as [Runtime nullSelector]) is identified by the tag s followed by 0 as the selector number.

Inherits

State supers: Coder

instance tom.BinaryCoder

variables

`int reference_size;`

The number of bytes issued for a reference. This starts with 1 (a byte), and can become 2 (a char) or 4 (an int).

methods

```
id
  init;
```

Undocumented.

class tom.Encoder

Inherits

State supers: Coder

instance tom.Encoder

variables

`MutableEqDictionary tmp_objects_done;`
: Keyed on the objects already encoded, the value is the identifier (which is an `IntNumber`) used for this object. This dict only contains temporary objects, i.e. objects that can be forgotten about after each `encodeRoot`.

`MutableEqDictionary perm_objects_done;`
: Similar, the non-temporary objects. This includes class objects and `Selector` objects.

`MutableEqSet objects_skipped;`
: The set of conditional objects that were skipped.

`MutableEqSet coded_classes;`
: The classes which, for the current object, have already done their part in the coding.

`int last_object_id;`
: The most recently issued object identifier.

methods

```
void
  encodeRoot State object;
```

The main entry `Encoder` method: encode the `object` and the whole object graph implied by it. This method is not reentrant.

```
id
  init;
```

Designated initializer.

```
boolean
  hasBeenCodedFor class (State) the_class;
```

Return `NO` if the object currently being encoded on this coder has not yet been encoded for `the_class`. Return `YES` otherwise. While coding an object, only the first invocation for a certain `the_class` will return `YES`; subsequent invocations will return `NO`. <h4>Encoding methods</h4>

```
void
  encode State object;
```

Encode the `object`, unconditionally.

```
void
  encodeConditionally State object;
```

Encode the `object`, but only if it already is part of the output graph. If this is not the case, `nil` is encoded, and if later on in the coding process the object previously encoded as nil is encountered (unconditionally), a `program-condition` will be raised to flag the inconsistency.

```
deferred void
  encode boolean v;
```

Encode the `boolean` `v`.

```
deferred void
  encode byte v;
```

Encode the `byte` `v`.

```
deferred void
  encode char v;
```

Encode the `char` `v`.

```
deferred void
  encode int v;
```

Encode the `int` `v`.

```
deferred void
  encode long v;
```

Encode the `long` `v`.

```
deferred void
  encode float v;
```

Encode the `float` `v`.

```
deferred void
  encode double v;
```

Encode the `double` `v`.

```
deferred void
  encode selector v;
```

Encode the `selector` `v`.

```
deferred void
  encodeBytes (int, int) (start, length)
         from ByteArray r
pre
  start >= 0 && length >= -1;
```

Encode the bytes in the range (`start, length`) from the array `r`.

```
deferred void
  encodeBytes (pointer, int) (address, length);
```

Encode the `length` bytes of which the first one resides at the `address`. <h4>Internal methods</h4>

```
deferred protected State
  replacementObjectFor State object;
```

Return the object to be encoded to this coder instead of the `object`. This method is implemented by subclasses to retrieve the actual object from the `object` itself, for instance by asking for it `replacementForStreamCoder` or `replacementForPortCoder`.

```
deferred protected void
  encodeNilObject;
```

Encode the `nil` reference.

```
deferred protected void
  encodeReference int v;
```

Encode a reference to the object known as `v`.

```
protected int
  identityFor All object;
```

Return the identity to be used for the non-class `object`. This returns the next value of `last_object_id`.

```
protected int
  identityForClass class (State) a_class;
```

Return the identity to be used for the class object `a_class`. This returns the next value of `last_object_id`.

```
class (State)
  startEncoding State object;
```

Undocumented.

```
protected void
  finishEncoding All object;
```

Invoked when the `object` has been encoded. Default does nothing.

```
protected void
  startEncodingRoot All object;
```

Invoked when coding starts with the root `object`. Default does nothing.

```
protected void
  finishEncodingRoot All object;
```

Invoked when coding the root `object` has finished. Default does nothing.

class tom.BinaryEncoder

Inherits

State supers: BinaryCoder, Encoder

instance tom.BinaryEncoder

variables

`MutableDictionary selectors;`
The selector dictionary, from `Selector` to `IntNumber`.

methods

```
id
  init;
```

Designated initializer.

```
class (State)
  startEncoding State object;
```

Undocumented.

```
protected void
  finishEncoding All object;
```

Invoked when the `object` has been encoded. Emit a close paren.

```
protected void
  updateReferenceSize;
```

Undocumented.

```
protected int
  identityFor All object;
```

Undocumented.

```
protected int
  identityForClass class (State) a_class;
```

Identify this class on the output `stream`, reporting its coding version.

```
protected void
  encodeNilObject;
```

Undocumented.

```
protected void
  encodeReference int v;
```
Undocumented.

```
void
  encode boolean v;
```
Undocumented.

```
void
  encode byte v;
```
Undocumented.

```
void
  encode char v;
```
Undocumented.

```
void
  encode int v;
```
Undocumented.

```
void
  encode long v;
```
Undocumented.

```
void
  encode float v;
```
Undocumented.

```
void
  encode double v;
```
Undocumented.

```
void
  encode selector v;
```
Undocumented.

```
void
  encodeBytes (pointer, int) (address, length);
```
Undocumented.

```
void
  encodeBytes (int, int) (start, length)
         from ByteArray r;
```
Undocumented.

```
protected void
  writeReference int r
pre
  reference_size == 1 || reference_size == 2 || reference_size == 4;
```
Undocumented.

```
deferred protected void
  writeByte byte b;
```
Undocumented.

```
deferred protected void
  writeBytes (int, int) (start, length)
        from ByteArray r;
```
Undocumented.

```
deferred protected void
  writeBytes (pointer, int) (address, length);
```
Undocumented.

```
protected void
  writeChar char c;
```
Undocumented.

```
protected void
  writeInt int i;
```
Undocumented.

```
protected void
  writeLong long l;
```
Undocumented.

class tom.Decoder

The Decoder class defines the interface to all decoder classes, be it binary or textual, stream or port base.

Inherits

State supers: Coder

instance tom.Decoder

variables

MutableIntDictionary tmp_objects_done;
: Objects, indexed on their identity, as retrieved from this coder.

MutableIntDictionary perm_objects_done;

IntegerRangeSet objects_referenced;
: The identity of the objects that have been referenced while being decoded.

MutableEqDictionary class_versions;
: Mapping from a class to the decoding version of that class.

MutableEqSet coded_classes;
: The classes which, for the current object, have already done their part in the coding.

methods

```
id
  init;
```

Designated initializer.

```
Any
  decodeRoot;
```

This is the entry point for the user of this decoder. The user invokes decodeRoot to retrieve an object, plus its underlying graph, from this decoder. The object is returned.

```
boolean
  hasBeenCodedFor class (State) the_class;
```

Return NO if the object currently being decoded on this coder has not yet been decoded for the_class. Return YES otherwise. While coding an object, only the first invocation for a certain the_class will return YES; subsequent invocations will return NO.

```
int
  versionOfClass class (State) cls
pre
  class_versions[cls] != nil;
```

Return the version of the class `cls` as encountered by this coder. The version can only be retrieved of classes already encountered curing the decoding process. <h4>Decoding methods</h4>

```
deferred Any
  decode;
```

Retrieve an object from this decoder and return it.

```
deferred boolean
  decode;
```

Undocumented.

```
deferred byte
  decode;
```

Undocumented.

```
deferred char
  decode;
```

Undocumented.

```
deferred int
  decode;
```

Undocumented.

```
deferred long
  decode;
```

Undocumented.

```
deferred float
  decode;
```

Undocumented.

```
deferred double
  decode;
```

Undocumented.

```
deferred selector
  decode;
```

Undocumented.

```
deferred (pointer, int)
  decodeBytes;
```

Decode a sequence of bytes from the coder to newly allocated memory space. Return the address and the length.

```
deferred void
  decodeBytes int num
           to pointer address;
```

Decode the num bytes from the coder to the address.

```
protected Any
  decodeObject class (State) cls
            as int ref;
```

<h4>Private methods</h4>

```
protected void
  finishDecoding All o;
```

Invoked by decodeObject as, after having invoked initWithCoder, but before awakeAfterUsingCoder. The default implementation does nothing.

```
protected Any
  reference int i;
```

Return the object referenced as the number i.

class tom.BinaryDecoder

The `BinaryDecoder` is an abstract decoding class which can decode binary encoded objects. It serves as the decoding engine for the `BinaryStreamDecoder` and `too.PortDecoder`.

Inherits

State supers: BinaryCoder, Decoder, C

instance tom.BinaryDecoder

variables

`MutableArray selectors;`
The selectors encountered so far, indexed on their identity.

methods

```
id
  init;
```

Designated initializer.

```
Any
  decode;
```

Decode and return an object.

```
Any
  decode byte b;
```

Decode and return an object, announced by the tag `b`.

```
boolean
  decode;
```

Undocumented.

```
byte
  decode;
```

Undocumented.

```
char
  decode;
```

Undocumented.

```
int
  decode;
```

Undocumented.

```
long
  decode;
```

Undocumented.

```
float
  decode;
```

Undocumented.

```
double
  decode;
```

Undocumented.

```
selector (result)
  decode;
```

Undocumented.

```
(pointer, int)
  decodeBytes;
```

Undocumented.

```
void
  decodeBytes int length
            to pointer address;
```

Undocumented. <h4>Internal methods</h4>

```
protected void
  finishDecoding All o;
```

Invoked when the object `o` has been decoded. Read a close paren.

```
protected byte
  nextPrimary;
```

Return the next primary tag byte, handling secondary tags such as reference size changes and class declarations.

If an unknown class is encountered, a `unknown-class-condition` is signaled. A handler may return a replacement class to be used instead. Failure to do so will later on result in a `nil-receiver` condition or a failed precondition.

```
protected byte
  nextPrimary byte expected;
```

Return the next primary tag byte, which must match `expected`. If it doesn't, a `program-condition` is raised.

```
protected int
  readReference
pre
  reference_size == 1 || reference_size == 2 || reference_size == 4;
```

Read an object reference from this decoder. Depending on the `reference_size` this read 1, 2, or 4 bytes.

```
protected Any
  readReference;
```

Read an object reference from this decoder and return the object referenced. This raises a `coding-condition` in case not a proper reference is encountered, or if the referenced object is unknown.

```
deferred protected byte
  readByte;
```

Return the next single `byte`.

```
deferred protected void
  readBytes int num
         to pointer address;
```

Undocumented.

```
protected char
  readChar;
```

Return the next two bytes as a `char`.

```
protected int
  readInt;
```

Return the next four bytes as an `int`.

```
protected long
  readLong;
```

Return the next 8 bytes as a `long`.

class tom.Collection

Inherits

State supers: State, Conditions, Copying, Enumerable

instance tom.Collection

methods

```
boolean
  dump_simple_p;
```

Return YES.

```
boolean
  equal id other;
```

Two collections consider themselves equal if they are the same object or when their elements are equal.

```
void
  freeze;
```

Make the receiving collection immutable. This is irreversible. It is a no-op for immutable collections.

```
Collection
  frozen;
```

Return self if this is a non-mutable Collection. Otherwise, return a non-mutable collection with the same contents.

```
Any
  member All object;
```

Return the element contained in this collection, which is equal to the object. The default implementation by Collection visits the elements using an enumerator.

```
Any
  memq All object;
```

Like member, but the element is identified on reference equality.

```
deferred int
  length;
```

Return the number of elements in this Collection.

```
void
  makeElementsPerform Invocation invocation;
```

Fire the `invocation` at the elements contained in the receiving collection.

```
void
  makeElementsPerform selector message;
```

Send the argumentless `message` to the elements contained in the receiving collection.

```
void
  makeElementsPerform selector message
                with All argument;
```

Send the `message` with the object `argument` to the elements contained in the receiving collection.

```
boolean
  mutable;
```

A `Collection` is not mutable.

```
void
  passElementsTo Invocation inv;
```

Fire the invocation repeatedly, each time with the next object from the collection completing the invocation.

```
OutputStream
  write OutputStream s;
```

Undocumented.

class tom.MutableCollection

Inherits

State supers: Collection

instance tom.MutableCollection

methods

```
deferred void
  add All object;
```
Undocumented.

```
void
  addElementsFrom Enumerable other;
```
Undocumented.

```
void
  addElementsFromEnumerator Enumerator e;
```
Undocumented.

```
deferred void
  empty;
```
Remove all elements from the receiving collection.

```
deferred void
  freeze;
```
Force `freeze` to be undefined since each particular subclass must itself implement it.

```
id (self)
  initWithEnumerator Enumerator e;
```
Initialize by feeding `self` the elements from the `enumerator`.

```
boolean
  mutable;
```
A `MutableCollection` is mutable.

class tom.Keyed

A `Keyed Collection` stores elements on a key.

Inherits

State supers: Collection

instance tom.Keyed

methods

```
deferred Any
  at All key
pre
  key != nil;
```

Undocumented.

class tom.MutableKeyed

Inherits

State supers: Keyed, MutableCollection

instance tom.MutableKeyed

methods

```
deferred void
  add All object;
```

Add the object.

```
deferred void
  remove All object;
```

Remove the object.

class tom.Mapped

A `Mapped Collection` is a `Keyed` collection which stores (key, value) associations.

Inherits

State supers: Keyed

instance tom.Mapped

methods

```
boolean
  equal id other;
```

Check not only the values, but also the keys.

```
deferred Enumerator
  keyEnumerator;
```

Return an enumerator on the keys of this mapped collection.

class tom.MutableMapped

Inherits

State supers: Mapped, MutableKeyed

instance tom.MutableMapped

methods

```
void
  add All value;
```

Undocumented.

```
deferred void
  set All value
   at All key
pre
  key != nil && value != nil;
```

Undocumented.

class tom.Ordered

A `Ordered Collection` maintains its elements in a specific order, though the time complexity or retrieving the nth object not necessarily independent of n.

Inherits

State supers: Collection

instance tom.Ordered

class tom.MutableOrdered

Inherits

State supers: Ordered, MutableCollection

instance tom.MutableOrdered

class tom.Indexed

An `Indexed Collection` maintains an association between integer indices and the objects it contains, with the promise that retrieving an object through the index is O(1) in time complexity.

Inherits

State supers: Ordered

instance tom.Indexed

methods

```
(int, int)
  adjustRange (int, int) (start, len);
```

Adjust the range (`start`, `len`) to fit the length of the receiving `Indexed` collection.

```
deferred Any
  at int index;
```

Return the element at `index`. If the receiving collection stores unboxed values, such as integers, the value returned is the element boxed.

```
byte
  at int index;
```

Return the byte value of the element at `index`. If the receiving collection stores objects, the `byteValue` of the element retrieved is actually returned.

```
char
  at int index;
```

The following all follow the <basic type> `at` <index> stanza.

```
int
  at int index;
```

Undocumented.

```
long
  at int index;
```

Undocumented.

```
float
  at int index;
```

Undocumented.

```
double
  at int index;
```

Undocumented.

```
dynamic
  elements;
```

Extract the elements from the receiving collection, as indicated by the return type. The number of elements in the collection must match the number of expected elements.

```
dynamic
  elements (int, int) (start, num);
```

Like `elements`, but extract only the `num` elements starting at index `start`.

```
Enumerator
  enumerator;
```

Return an `Enumerator` on this `Indexed`.

```
boolean
  equal id other;
```

A faster implementation than the one inherited from `Collection`.

```
int
  indexOf All element;
```

Return the index of the first `element`, or -1 if it could not be found.

```
int
  indexOfIdentical All element;
```

Return the index of the first identical `element`, or -1 if it could not be found. <h4>Performing</h4>

```
void
  makeElementsPerform Invocation inv;
```

A faster implementation than the one inherited from `Collection`, without using an `Enumerator`.

```
void
  makeElementsPerform selector message;
```

Likewise.

```
void
  makeElementsPerform selector message
                  with All argument;
```

Likewise.

```
void
  passElementsTo Invocation inv;
```

Likewise.

class tom.MutableIndexed

Inherits

State supers: Indexed, MutableOrdered

instance tom.MutableIndexed

methods

```
deferred void
  set All object
   at int index;
```

Store the `object` at `index` in the receiving collection.

```
deferred void
  swap (int, int) (i, j);
```

Swap the elements at the indices `i` and `j`.

```
void
  reverse (int, int) (start, len);
```

Reverse the elements in the range starting at `start`, with length `len`.

```
deferred void
  removeElementAt int index;
```

Remove the element at `index`, decreasing by 1 the indices of the elements further in the collection.

```
void
  removeElement All element;
```

Remove the first occurence of `element`.

```
void
  removeIdenticalElement All element;
```

Remove the first occurence of the identical `element`.

class tom.IndexedEnumerator

The `IndexedEnumerator` enumerates any `Indexed` collection, returning the elements boxed.

Inherits

State supers: State, Enumerator

methods

```
instance (id)
  with Indexed indexed;
```

Undocumented.

instance tom.IndexedEnumerator

variables

`int next;` The index of the next element to be returned.

`int num;` The index one beyond the last element to be returned.

`Indexed indexed;`
The actual indexed collection.

methods

```
id (self)
    init Indexed a
  start: int start = 0
  length int length;
```

Designated initializer.

```
(boolean, Any)
  next;
```

Undocumented.

```
(boolean, byte)
  next;
```

All these `next` methods are not really necessary, as they are provided by the `Enumerator` behaviour. However, binding them here directly greatly enhances speed and reduces memory requirements for non-object indexeds, which now also do not need their own enumerator to obtain speed.

```
(boolean, char)
  next;
```

Undocumented.

```
(boolean, int)
  next;
```

Undocumented.

```
(boolean, long)
  next;
```

Undocumented.

```
(boolean, float)
  next;
```

Undocumented.

```
(boolean, double)
  next;
```

Undocumented.

class tom.Constants (config)

variables

```
const TOM_RESOURCES = "/usr/local/lib/tom/charmaps/";
const TOM_PREFIX = "/usr/local/";
```

instance tom.Constants (config)

class tom.Sink

Inherits

State supers: State, OutputStream

instance tom.Sink

methods

```
void
  close;
```

Undocumented.

```
void
  write byte b;
```

<h4>OutputStream</h4>

```
int
  write byte b;
```

Undocumented.

```
int
  writeRange (int, int) (start, length)
        from ByteArray buffer;
```

Undocumented.

```
id
  print boolean b;
```

Undocumented.

```
id
  print byte b;
```

Undocumented.

```
id
  print char c;
```

Undocumented.

```
id
  print int i;
```

Undocumented.

```
id
  print long l;
```
Undocumented.

```
id
  print float f;
```
Undocumented.

```
id
  print double d;
```
Undocumented.

```
id
  print pointer addr;
```
Undocumented.

```
id
  print dynamic x;
```
Undocumented.

class tom.ByteNumber

Inherits

State supers: Number

instance tom.ByteNumber

variables

`byte value;`

methods

```
byte
  byteValue;
```
Undocumented.

```
char
  charValue;
```
Undocumented.

```
double
  doubleValue;
```
Undocumented.

```
float
  floatValue;
```
Undocumented.

```
int
  intValue;
```
Undocumented.

```
long
  longValue;
```
Undocumented.

```
int
  compare Number n;
```
Undocumented.

```
int
  compare byte v;
```

Undocumented.

```
int
  compare char v;
```

Undocumented.

```
int
  compare int v;
```

Undocumented.

```
int
  compare long v;
```

Undocumented.

```
int
  compare float v;
```

Undocumented.

```
int
  compare double v;
```

Undocumented.

```
id
  init byte value;
```

Undocumented.

```
id
  init char value;
```

Undocumented.

```
id
  init int value;
```

Undocumented.

```
id
  init long value;
```

Undocumented.

```
id
  init float value;
```

Undocumented.

```
id
  init double value;
```

Undocumented.

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
OutputStream
  write OutputStream s;
```

Undocumented.

class tom.CharNumber

Inherits

State supers: Number

instance tom.CharNumber

variables

`char value;`

methods

```
byte
  byteValue;
```
Undocumented.

```
char
  charValue;
```
Undocumented.

```
double
  doubleValue;
```
Undocumented.

```
float
  floatValue;
```
Undocumented.

```
int
  intValue;
```
Undocumented.

```
long
  longValue;
```
Undocumented.

```
int
  compare Number n;
```
Undocumented.

```
int
  compare byte v;
```

Undocumented.

```
int
  compare char v;
```

Undocumented.

```
int
  compare int v;
```

Undocumented.

```
int
  compare long v;
```

Undocumented.

```
int
  compare float v;
```

Undocumented.

```
int
  compare double v;
```

Undocumented.

```
id
  init byte value;
```

Undocumented.

```
id
  init char value;
```

Undocumented.

```
id
  init int value;
```

Undocumented.

```
id
  init long value;
```

Undocumented.

```
id
  init float value;
```

Undocumented.

```
id
  init double value;
```

Undocumented.

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
OutputStream
  write OutputStream s;
```

Undocumented.

class tom.IntNumber

Inherits

State supers: Number

instance tom.IntNumber

variables

`int value;`

methods

```
     byte
       byteValue;
```
Undocumented.

```
     char
       charValue;
```
Undocumented.

```
     double
       doubleValue;
```
Undocumented.

```
     float
       floatValue;
```
Undocumented.

```
     int
       intValue;
```
Undocumented.

```
     long
       longValue;
```
Undocumented.

```
     int
       compare Number n;
```
Undocumented.

```
     int
       compare byte v;
```

Undocumented.

```
int
  compare char v;
```

Undocumented.

```
int
  compare int v;
```

Undocumented.

```
int
  compare long v;
```

Undocumented.

```
int
  compare float v;
```

Undocumented.

```
int
  compare double v;
```

Undocumented.

```
id
  init byte value;
```

Undocumented.

```
id
  init char value;
```

Undocumented.

```
id
  init int value;
```

Undocumented.

```
id
  init long value;
```

Undocumented.

```
id
  init float value;
```

Undocumented.

```
id
  init double value;
```

Undocumented.

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
OutputStream
  write OutputStream s;
```

Undocumented.

class tom.LongNumber

Inherits

State supers: Number

instance tom.LongNumber

variables

`long value;`

methods

```
byte
  byteValue;
```
Undocumented.

```
char
  charValue;
```
Undocumented.

```
double
  doubleValue;
```
Undocumented.

```
float
  floatValue;
```
Undocumented.

```
int
  intValue;
```
Undocumented.

```
long
  longValue;
```
Undocumented.

```
int
  compare Number n;
```
Undocumented.

```
int
  compare byte v;
```

Undocumented.

```
int
  compare char v;
```

Undocumented.

```
int
  compare int v;
```

Undocumented.

```
int
  compare long v;
```

Undocumented.

```
int
  compare float v;
```

Undocumented.

```
int
  compare double v;
```

Undocumented.

```
id
  init byte value;
```

Undocumented.

```
id
  init char value;
```

Undocumented.

```
id
  init int value;
```

Undocumented.

```
id
  init long value;
```

Undocumented.

```
id
  init float value;
```

Undocumented.

```
id
  init double value;
```

Undocumented.

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
OutputStream
  write OutputStream s;
```

Undocumented.

class tom.FloatNumber

Inherits

State supers: Number

instance tom.FloatNumber

variables

`float value;`

methods

```
byte
  byteValue;
```
Undocumented.

```
char
  charValue;
```
Undocumented.

```
double
  doubleValue;
```
Undocumented.

```
float
  floatValue;
```
Undocumented.

```
int
  intValue;
```
Undocumented.

```
long
  longValue;
```
Undocumented.

```
int
  compare Number n;
```
Undocumented.

```
int
  compare byte v;
```

Undocumented.

```
int
  compare char v;
```

Undocumented.

```
int
  compare int v;
```

Undocumented.

```
int
  compare long v;
```

Undocumented.

```
int
  compare float v;
```

Undocumented.

```
int
  compare double v;
```

Undocumented.

```
id
  init byte value;
```

Undocumented.

```
id
  init char value;
```

Undocumented.

```
id
  init int value;
```

Undocumented.

```
id
  init long value;
```

Undocumented.

```
id
  init float value;
```

Undocumented.

```
id
  init double value;
```

Undocumented.

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
OutputStream
  write OutputStream s;
```

Undocumented.

class tom.DoubleNumber

Inherits

State supers: Number

instance tom.DoubleNumber

variables

`double value;`

methods

```
byte
  byteValue;
```
Undocumented.

```
char
  charValue;
```
Undocumented.

```
double
  doubleValue;
```
Undocumented.

```
float
  floatValue;
```
Undocumented.

```
int
  intValue;
```
Undocumented.

```
long
  longValue;
```
Undocumented.

```
int
  compare Number n;
```
Undocumented.

```
int
  compare byte v;
```

Undocumented.

```
int
  compare char v;
```

Undocumented.

```
int
  compare int v;
```

Undocumented.

```
int
  compare long v;
```

Undocumented.

```
int
  compare float v;
```

Undocumented.

```
int
  compare double v;
```

Undocumented.

```
id
  init byte value;
```

Undocumented.

```
id
  init char value;
```

Undocumented.

```
id
  init int value;
```

Undocumented.

```
id
  init long value;
```

Undocumented.

```
id
  init float value;
```

Undocumented.

```
id
  init double value;
```

Undocumented.

```
void
  encodeUsingCoder Encoder coder;
```

Undocumented.

```
void
  initWithCoder Decoder coder;
```

Undocumented.

```
OutputStream
  write OutputStream s;
```

Undocumented.

class tom.Stream

Inherits

State supers: Conditions
Behaviour supers: All

instance tom.Stream

Inherits

State supers: Conditions
Behaviour supers: All

methods

```
deferred void
  close;
```

Undocumented.

class tom.InputStream

Inherits

State supers: Stream

instance tom.InputStream

methods

```
id
  flushInput;
```

Discard any unread input. The default implementation does nothing.

```
deferred byte
  read;
```

Read a byte from the receiving stream. This raises a `stream-eos` condition upon end-of-stream.

```
deferred int
  read;
```

Read a byte from the receiving stream. Return -1 for end-of-stream.

```
int
  readBytes int num
       into MutableByteArray buffer;
```

Read at most `num` bytes from the stream into the `buffer`, starting to add bytes at its current length. Return the number of bytes successfully read, which is 0 for end-of-stream.

```
deferred int
  readRange (int, int) (start, num)
       into MutableByteArray buffer;
```

Read at most `num` bytes from the stream into the `buffer`, by writing in it from position `start`. Return the number of bytes successfully read, which is 0 for end-of-stream.

```
MutableByteString
  readLine;
```

Read a '\n' terminated sequence of bytes and return them (without the '\n' at the end). Return `nil` upon end of file (if no characters have been collected).

```
MutableByteArray
  readLineInto MutableByteArray buf
     truncate: boolean trunc = YES;
```

Read a '\n' terminated sequence of bytes and return them (without the '\n' at the end) in `buf`. Return `nil` upon end of file (if no characters have been collected). If the optional `truncate` is not NO, the buffer is truncated before use.

class tom.OutputStream

Inherits

State supers: Stream

variables

`local static MutableByteString print_buffer;`
  The buffer used by `print base:....`

instance tom.OutputStream

methods

```
id
  flushOutput;
```

Write out any unwritten (buffered) output. The default implementation does nothing.

```
id
  nl;
```

Output a new line to the receiving stream. An interactive stream should override this to also flush its output if it desires line based buffering.

```
private MutableByteString
  print_buffer;
```

Return the truncated `print_buffer`, creating it iff necessary.

```
id
  print boolean b;
```

Undocumented.

```
id
  print byte b;
```

Undocumented.

```
id
  print char c;
```

Undocumented.

```
id
  print int i;
```

Undocumented.

```
id
  print long l;
```

Undocumented.

```
id
  print float f;
```

Undocumented.

```
id
  print double d;
```

Undocumented.

```
id
  print pointer addr;
```

Undocumented.

```
id
  print All object;
```

Undocumented.

```
id
  print dynamic x;
```

Send `print` to `self` for each element of the tuple `x`.

```
id
      print int value
       base: int base = 10
      space: int space = 0
      flush: int flush = -1
     signed: int how_signed = -1
      range: char digit_10 = char ('a');
```

Output the `value` to the receiving stream.

The optional `base` dictates the base of the representation, which defaults to 10.

If the optional `space` is not 0, it is the number of positions the representation must at least occupy.

If `space` is not 0, the optional `flush` dictates how the representation is to be flushed. A negative value means left, 0 for center, and a positive value dictates a right shift. The absolute value of `flush` indicates the amount of whitespace which must be available at the other end.

The optional `signed` should be 0 for unsigned, or 1 for signed. If it is -1 (the default) the value is assumed unsigned, unless `base` has its default value, 10.

The optional `digit_10` sets the value to use for the decimal value 10 when using a `base` exceeding that value.

```
deferred void
  write byte b;
```

Write the byte `b`, signaling a condition upon eof.

```
deferred int
  write byte b;
```

Write the byte `b` and return the number of bytes actually written.

```
deferred int
  writeBytes int num
        from pointer address;
```

The lowest level multiple-byte writing method: Write the `num` bytes from `address` to the stream, and return the number of bytes written.

```
int
  writeRange (int, int) (start, length)
        from ByteArray buffer;
```

Undocumented.

```
int
  writeBytes ByteArray buffer;
```

Undocumented.

class tom.InputOutputStream

Inherits

State supers: InputStream, OutputStream

instance tom.InputOutputStream

class tom.SeekableStream

Inherits

State supers: Stream, Constants

instance tom.SeekableStream

methods

```
deferred long
  position;
```

Return the current position.

```
deferred void
       seek long offset
  relative: int whence = STREAM_SEEK_SET;
```

Set the position. Any following operation will operate from the new position. The optional argument `whence` defaults to `STREAM_SEEK_SET`, for absolute positioning. Possible other values are `STREAM_SEEK_CUR`, for positioning relative to the current position, and `STREAM_SEEK_END`, to work relative to the end.

class tom.StreamStream

Inherits

State supers: Stream, State

methods

```
instance (id)
  with Stream s;
```

Undocumented.

instance tom.StreamStream

variables

`public InputOutputStream stream;`
The stream to which we output and/or from which we input.

methods

```
void
  close;
```

Undocumented.

```
id
  init Stream s;
```

Undocumented.

class tom.stdio

Inherits

Behaviour supers: All

variables

`static public InputStream in;`
The stream connected to descriptor 0, known as `stdin` in C.

`static public OutputStream out;`
The stream connected to descriptor 1, C's `stdout`.

`static public InputOutputStream err;`
The stream connected to descriptor 2, C's `stderr`. Like `out`, this stream is buffered.

methods

```
void
  close int descriptor;
```

Close the `descriptor`. Raises a `stream-error` on failure.

instance tom.stdio

Inherits

Behaviour supers: All

class tom.UniqueString

Inherits

State supers: String

variables

`static MutableSet strings;`
: The `Container` of all unique strings. The container mechanism will ensure only those strings are kept that are still needed.

methods

```
void
  load Array arguments;
```

Initialize the `strings` `Container`.

```
instance (id)
  with String s;
```

Return the `UniqueString` containing the same information as the `String s`. This is the only method to be used to create unique strings.

```
protected instance (id)
  awake instance (id) us
  equal selector cmp;
```

Find the string `us` in the `strings`, comparing them using the selector `cmp`. If the string is found, the old string is returned. Otherwise, `us` is added to the `strings` and it is returned.

instance tom.UniqueString

variables

`int hash;` The cached hash value of this string.

methods

```
boolean
  equal String other;
```

Tell the `other` to compare itself to this `UniqueString`.

```
boolean
  equalUniqueString UniqueString other;
```

Return `TRUE`. This is simple pointer comparison.

```
int
  compare id other;
```

Override compare, to return fast upon equality.

class tom.UniqueByteString

Inherits

State supers: UniqueString, ByteString

instance tom.UniqueByteString

methods

```
boolean
  equal String other;
```

This definition is only here because we can not direct the implementation of this selector to the right class.

```
boolean
  equalUniqueString UniqueString other;
```

The same is true for this one.

```
int
  hash;
```

When called for the first time, hash the string, and remember the value. Every next time, return the cached value. Obviously, this loses for strings with a zero hash value (which keep being hashed).

```
id
  awakeAfterUsingCoder Decoder coder;
```

Iff a unique string like the receiving one already exists, return the already existing one. Otherwise, add the receiving string to the known unique strings.

class tom.UniqueCharString

Inherits

State supers: UniqueString, CharString

instance tom.UniqueCharString

methods

```
boolean
  equal String other;
```

Redefinition since we can not redirect to the right (`UniqueString`) super.

```
boolean
  equalUniqueString UniqueString other;
```

Redefinition since we can not redirect to the right (`UniqueString`) super.

```
int
  hash;
```

Cached hashing, just like `UniqueByteString`.

```
id
  awakeAfterUsingCoder Decoder coder;
```

Iff a unique string like the receiving one already exists, return the already existing one. Otherwise, add the receiving string to the known unique strings.

Table of Contents