

CAMERA prototype implementation Documentation

Draft

Atze Dijkstra

March 24, 1994

Revision: 1.60 ¹

¹Header: /amd/infix/home/projects/camera/d.proto/develop/doc/RCS/doc.tex,v 1.60 1993/10/08
07:58:14 atze Exp atze .

Contents

1	Introduction	3
2	Global model description	3
2.1	Object management systems	3
2.2	Versioning and album	3
2.3	Dependency management	3
3	Global implementation description	4
4	Conventions and notations	4
4.1	Notations	4
4.2	Types	4
4.3	Type checking	4
4.4	Object naming	5
4.5	Routines	5
5	Interfacing to camera oms's	5
5.1	Initialization	5
5.2	Files and UNIX locations	7
5.3	Starting an oms	7
5.4	Starting an interface process	7
5.5	Starting a test environment	7
5.6	Connecting to an oms from an interface process	7
5.7	Stopping an oms	7
5.8	Using objects of an oms	8
5.9	Convenience routines for using an oms	8
5.10	Leaving oms's	9
6	Classes and methods	9
6.1	Object	9
6.2	Class	11
6.3	Environment	13
6.4	File	14
6.5	Method	14
6.6	Piece	15
6.7	Snapshot	16
6.8	System	17
6.9	System-Relation	18
6.10	Album-Piece	19
6.11	Scheme-Src	19
6.12	Unix-Tree	19
6.13	Function	20
6.14	Procedure	20
6.15	Relation	20
6.16	Oms-Src	21
6.17	Computed-Relation	21
7	Relations	21
7.1	Naming of objects	22
8	Extensions of scheme, routines for the implementor	22
8.1	Oms in general	22
8.2	Bytestring and longfield	23

8.3	Miscellaneous	24
8.4	Scheme extensions	25
8.4.1	General utilities	25
8.4.2	Hash tables and sets	26
8.4.3	Comparison, sorting	27
8.4.4	Unix interface	27
8.4.5	Relations	28
8.5	Others	28
9	Changes and bugs	28
9.1	Since version 1.32	28
9.2	Since version 1.41	29
9.3	Since version 1.46	29
9.4	Since version 1.49	29
9.5	Since version 1.50	29
9.6	Since version 1.52	29
9.7	Unsolved bugs, problems and future work	29
A	Example session	30
A.1	Album	30
A.2	Interface	30
B	Demo source 'demo.scm'	35
C	Directory structure	38
D	Class hierarchy	38

1 Introduction

This document describes the functionality of the **camera** prototype system. It can be used as a short introduction into **camera** and as a user/reference manual.

In section 2 the global architecture is described, more information and rationales for design decisions can be found in [...]. Section 3 describes some implementation aspects necessary for understanding the limitations of the system. Section 4 describes the conventions and notations used throughout the description of the system. Sections 5 through 8 contain the descriptions of available routines for using the system as well as extending the system.

2 Global model description

2.1 Object management systems

Camera is a system for maintaining versions of data. Data is modeled by objects, an *object* is an aggregate of *attributes*, an *attribute* is an *elementary value* and an *elementary value* is a value provided by the implementation language. An *attribute* may not be an *object* or reference to an *object*. References between *objects* are implemented by *relations* between *objects*. A *relation* is a set of *relationships* or *tuples*, each *tuple* contains a number of fields which may have a value matching a template. Each *object* belongs to a *class*. *Classes* and *objects* relate to each other as known from other object oriented systems like Smalltalk. A *class* describes the structure of *objects* belonging to that *class* and has *methods* associated with it for performing actions on an *object* of the *class*. *Methods* fall apart in two *classes*, *Procedures* and *Functions*. A *Procedure* is a “classical” *Method* returning a value. A *Function* is a *Method* for which dependency and derivation management is done. *Dependency* management consists of maintaining information about which function results or attribute values are used to calculate other function results. *Derivation* management consists of caching these result values. A set of *objects*, *classes* and *relations* forms an *object management system* or *oms*.

2.2 Versioning and

The unit of version management is a *snapshot*. A *snapshot* is an image of an *oms*. A *snapshot* is passive, if the data of a *snapshot* is manipulated, it is called an *environment*. An *environment* is an *oms* with a *snapshot* as its image. Changes made within an *environment* generate new versions of the *snapshot* used as the initial image. An *environment* has a current *snapshot* associated with it. This administration is maintained by a special *oms*, called the *album-oms*. For the *album-oms* no dependency and derivation management is done. Requests for using *snapshots* must be directed to the *album-oms*.

2.3 Dependency management

A *method* invocation may be either *lazy* or *eager*. Such an invocation is *eager* if its value is always consistent with its input. An invocation is *lazy* if a possibly needed recalculation is postponed until its value is needed. The recalculation may be done immediately when an input changes or more explicitly on user request or after each user command.

3 Global implementation description

The `camera` prototype is implemented on networks of UNIX machines. An `oms` is a UNIX process. `Oms`'s communicate using sockets. `Oms`'s are implemented in `scheme`². *Scheme* is a lisp variant combining lisp syntax and semantics with Algol-like behaviour. Time critical functionality is implemented in C, see also section 8. Using `oms` objects is done by evaluating `scheme` expressions. The elementary types allowed for attributes of objects are the types provided by the implementation language `scheme`. From now on basic knowledge about `scheme` is assumed.

Other implementation issues or implementation dependent behaviour is described where appropriate, especially w.r.t. `classes`.

4 Conventions and notations

4.1 Notations

Tuples are denoted by $\langle \text{field1, field2, \dots, fieldn} \rangle$. Lists with a certain element type are denoted by $\{\text{elt, \dots}\}$. A list with different typed elements is denoted by $\{\text{elt1, elt2, \dots, eltn}\}$. Other notations are self explanatory.

4.2 Types

The elementary types used and supported by `camera` are `boolean`, `list`, `symbol`, `string`, `char`, `number`, `lambda`, `vector`. `Lambda` is a declared procedure or a lambda expression. Other types used solely by the `camera` implementation are `regex`, `bstring`, `relation`, `timestamp`, `lfieldid`, `storage` and `objectid(oid)`. A `timestamp` is derived from GMT.

4.3 Type checking

Type checking and search patterns for relations use `type-lists` for checking actual values. A *type-list* is a list of check values (*type-list-elements*) of which each one must return 'true' (or *match*) when used for checking against a value. The following check values are allowed and used in this order:

****** matches any or no values remaining in the list of actual values.

****:** matches any number of remaining values which match the next type specification in the `type-list`.

***** matches any actual value.

.. specifies that following values are optional.

{type,scheme-types ..} matches if the actual value is of a `scheme-type`. If the list of types `scheme-types` is empty any type except `objectid` is matched.

{class,objectids ..} matches if the actual value is an `objectid` and belongs to one of the classes `objectids`.

{value,scheme-values ..} matches if the actual value equals one of the `scheme-values`.

{list,type-list ..} matches if the actual value is a list matching the `type-list` as described here.

²ELK scheme version 1.3.

`{match,regex}` matches if the regular expression `regex` is matched. The syntax of `emacs` regular expressions is used. Beware of backslashes (`\`) since they have a escape meaning in scheme strings.

`{not,type-list-element}` matches if the actual value does not match `type-list-element`.

`value` matches if the actual value equals the `value`.

`{value, ...}` matches if the actual value is contained in the `{value, ...}`.

`procedure` matches if the call `procedure(actual-value)` returns true.

Checking is done on parameter and return values of methods and on adding tuples to relations. Patterns for query use this checking mechanism too.

4.4 Object naming

An object is identified by its object identifier or oid. An *oid* is a tuple `(internet-id, process-id, sequence-id)` of which the printable and readable notation is `#[objectid internet-id process-id sequence-id]`. The system tries to construct a name for an oid³ and includes this in the printable notation. The parenthesized part of the name denotes the class of the object and the string `<-` means the oid is the oid of a method. In the latter case the printed class is the class for which the object is a method.

Names of classes start with uppercase letters.

Also a mapping from symbols to oids exists, see section 7.1 and appendix C.

4.5 Routines

Routines are described independent of the actual `scheme` syntax needed to invoke routines. Result values consisting of several values are returned as a list. A parameter and type pair is denoted by `parameter: type`. If the type of a parameter or result is left out, arbitrary values are allowed. The words `message` and `method` are used to refer to the same concept, i.e. a method. The words `self` and `receiver` are used to denote the receiving object.

5 Interfacing to camera oms's

5.1 Initialization

Configuration information and locations in the host system are retrieved at initialization from `camerarc` files which are (in the order read): `SysLibDir/camerarc`, `SysLibMachDir/camerarc` and `HOME/.camerarc`, where the capitalized names stand for some system directories. In these files definitions according to the following syntax may be put:

```
<defs>          ::= <def> SEQ OPT .
<def>           ::= <id> '=' <values> '.' .
<values>       ::= <value> SEQ OPT .
<value>        ::= <string> | <quoted-string>
                  | <tilde-string> | <cond-value>
<cond-value>   ::= '(' <value> <rel-op> <value>
```

³Some of the system relation are used to construct such a name.

```

        '?' <value>
        ( ':' <value> ) OPT
.
<quoted-string> ::= '"' <string> '"' .
<tilde-string>  ::= '~' <string> .
<id>           ::= <string> .

```

All fields may be separated by whitespace or comment, which starts with a # and ends with a linefeed. A definition is a series of concatenated string <value>s. A <value> may be a sequence of characters in between quotes. In this case no interpretation on the value is done. If not quoted the value may not contain syntactical characters and is either (in this order)

1. Replaced by its formerly defined value.
2. Interpreted as and replaced by the system (UNIX) environment variable with that name.
3. Left as it is.

A <tilde-string> expands (à la *cs*) to the home directory of the username.

The following names are defined and/or used by the camera system.

SysDir. The directory containing the sources of the camera system.

SysLibDir. The directory containing the library stuff of the camera system.

SysLibMachDir. The directory containing the machine dependent stuff of the camera system.

Machine. The architecture on which the system is running.

StoreDir. The location where snapshots, longfields, etc. are stored. Default is HOME/.camera.

LoadPath. A sequence of directory names separated by space, tab or ':' in which scheme sources are looked up when loaded. Default are the SysDir, SysLibDir and SysLibMachDir.

SysTmpDir. The temporary storage directory.

SysInitFile. The file containing scheme initialization code. Default is StoreDir / "init-a.scm".

OmsInitFile. The file containing scheme initialization code for the camera oms. Default is StoreDir / "init-b.scm".

OmsHeapSize. The size of the heap of (oms) processes started by the album server.

OmsType. The type of scheme/oms running, values are album(for the album server) , oms(for the environment server), itf(for the interface process), store, tst and lfield.

ItfType. The kind of interface used, values are (again) album, env(for the environment server) , itf and tst.

SysScmInterpreter. The scheme interpreter used.

Pid. The current process identifier.

User. The current user.

Host. The host running on.

A value may also be a conditional value <cond-value> following the syntax of the C-language conditional expressions, except for the equality operator which is '='. An empty else part yields an empty string.

5.2 Files and UNIX locations

The UNIX directory `StoreDir` is used to store images of an `album` process and the image of the album server itself. This directory is called the *store directory*. See also section 5.1 and section 8.3.

5.3 Starting an oms

An `album-oms` is started by executing the UNIX command “`camera album socket-nr`” where `socket-nr` is a number over which the `album-oms` will listen as a server accepting requests. An `environment-oms` is started by method invocation of an `environment` object, see class `Environment`. The `albumserver` starts a longfield and a snapshot storage server. The sockets at which these servers are listening are passed to all environment processes started. After starting an `oms`, a connection to it must be made from an interface process.

5.4 Starting an interface process

An interface process is started by executing the UNIX command “`camera`”. A scheme prompt will appear and scheme expressions may be typed in after which they will be evaluated in the context of the interface process.

5.5 Starting a test environment

As a test environment camera can be started by executing the UNIX command “`camera tst`”. The standard scheme interface appears and all `oms` commands, except those related to `album` can be used. This test version creates a new initial `oms` at startup and can thus be used as a test bed for new camera applications not requiring the `album` features. In the future this variation may be removed.

5.6 Connecting to an oms from an interface process

`oms-connect` (`machine-name: string` `socket-nr: integer`) \mapsto (`oms-handle: integer`) `interface`

Description. Open a socket stream connection to the `oms` process on `machine-name` listening on `socket-nr`. The `oms-handle` is a port number used for identifying the connection within the interface process. A shorthand for `oms-connect` is `><`.

`oms-disconnect` (`oms-handle`) \mapsto () `interface`

Description. Close the socket stream connection to the `oms` process connected to over port nr `oms-handle`. A shorthand for `oms-disconnect` is `<>`.

5.7 Stopping an oms

`oms-quit` (`oms-handle`) \mapsto () `interface`

Description. First quits the `oms` then closes the socket stream connection to the `oms` process connected to over port nr `oms-handle`. A shorthand for `oms-disconnect` is `>|`.

5.8 Using objects of an oms

`interface-switch` (oms-handle: integer) \mapsto () interface

Description. Make the interface use the `oms-handle` as the oms where messages with `send` are sent to. A shorthand for the routine is `>>`.

`send` $\left(\begin{array}{l} \text{oid: oid,} \\ \text{selector: symbol,} \\ \text{arguments} \end{array} \right) \mapsto$ (value) interface

Description. Send the message (invoke method for) `selector` to the object `oid` with `arguments` in the current oms. See section 6 for an overview of available methods and their expected arguments and return values. A shorthand for `send` is `<<`.

5.9 Convenience routines for using an oms

These routines are primarily available from the interface. Equivalents with the same name are available for method implementation, except if noted otherwise.

`dir-lookup` $\left(\begin{array}{l} \text{name: symbol,} \\ \text{root: objectid,} \\ \text{error: boolean} \end{array} \right) \mapsto$ (object: oid) interface

Description. Looks up the `oid` of an object associated with `name`. The `root` and `error` parameters are optional. If `root` is and `oid` it specifies a different root. If `error` is false (default true) no error situation is created in case of an error, false is returned instead. The search starts at `self`, unless `name` starts with the separator character `'/'` or `'@'`. In this case the search is started with the `root` as returned by `oms-connect`. `Name` consists of strings separated by the separator character. Each string between separators or the end of the string is used to retrieve an `oid` of a directory entry. It is an error if zero or > 1 entries exist. A shorthand for `dir-lookup` is `@`. Other variants for looking up a class resp. relation are `dir-lookup-class` and `dir-lookup-rel` with their resp. shortcuts `@c` and `@r`. As a `scheme` extension for easier typing the `scheme` read routine uses `#@` as an in-line mapping for names. `read` accepts `#@<name>` which is equivalent to `(dir-lookup '<name>')`. In this case false is returned when an error occurs. If `'@'` is used as a separator instead of `'/'`, the string before the separator need only be a (unique) prefix of the directory entry looked for.

`get-edit-set` $\left(\begin{array}{l} \text{oid,} \\ \text{name: symbol} \end{array} \right) \mapsto$ () interface

Description. Sends the message `'get-name` to `oid`, edits the returned string value and sends the message `'set-name` with the edited string value as a parameter. This interface uses the convention that pairs of methods exist, e.g. `'value` and `'set-value` resp. retrieving and setting a value, thereby using a string representation as an intermediate value.

5.10 Leaving oms's

An interface process is stopped by typing enough EOF's.

Further Ctrl-C (or equivalent) quits the system (in case of emergency).

6 Classes and methods

6.1 Object

Class Object

"Class Object, super class of 'the rest'. Methods for all objects."

Procedure `::edit (display:string) :objectid`

Edit self on display(optional; default from env variable).

Procedure `:change-class (new-class:Class) :objectid`

Changes the class of self. Attributes are added to self if necessary, the relation instance-of is adapted. Return previous class of self.

Note: Proper initialization of new attributes does not take place. Attributes are not deleted when the object is changed to a class without those attributes.

Procedure `:ev (expr:any-scheme-type) :`

Eval arbitrary scheme expression in OMS.

Procedure `:export-to-unix (file:string) :`

Album only. Put an external representation of the Album object self on the unix file.

Function `:get-attribute (name:symbol/string) :`

Get attribute name of self.

Function `:object-to-bstring () :lfieldid/string`

Album only. Return a bstring that represent the object self.

Function `:object-to-list () :list`

Album only. Return a list representation of the object self.

Procedure `:set-attribute (name:symbol/string, val:) :`

Set attribute name of self to val.

Procedure `:ship-to-user (user:symbol/string) :`

Album only. Send an external representation of the object to user.

Function `attribute-names () :list`

Return names of attributes of self.

Function `class () :objectid`

Return the class of self.

Function `class-name () :string`

Return name of the class of self.

Function `classes (all:boolean) :list`

Return list of classes to which self belongs. Return also the superclasses if all(optional; default false) is true.

Procedure `clone () :objectid`

Return copy of self.

Procedure `delete () :`

Delete self. Depending on property on-delete-object references are deleted. ??.

Note: Only the options 'warn and 'delete are implemented.

Procedure `depends-on (selector:symbol) :`

Print values on which method invocation of selector on self depends.

Function `describe (as-list:boolean) :list/string`

Return description of self in a string. If `as-list`(optional; default false) is true the description is returned as a list of strings and lists: {class,name,attr-fields}. The fields is a list of (name, value). If no information is available empty strings are returned.

Procedure `dir-add (name:symbol, oid:objectid) :objectid`

Add name as entry (name, oid) to directory for self. An existing entry with the same name is overwritten.

Function `dir-contents () :list`

Returns list of (sorted) entries of self as (name, oid) pairs (as list of list).

Procedure `dir-delete (name:symbol) :objectid`

Delete name as entry from directory for self.

Function `dir-lookup (name:symbol/string/objectid, root:objectid/list/boolean, :boolean) :objectid`

Lookup in the directory, starting from self, the object name. If `root`(optional; default /) available it is used as the start. If `error`(optional; default true) is false, false will be returned on an error. See `dir-lookup`.

Procedure `dir-make (name:symbol) :objectid`

Create a new object of class `Object` and add it to the directory self under name.

Function `dir-name (restrict?:boolean) :symbol/string`

Return a name of self using the directory for searching. If no name exists a string containing the object identifier is returned. If more than one entry exists in the directory an arbitrary one is chosen. If the `restrict`(optional; default false) is true only a name instead of a complete path is constructed.

Function `dir-parents () :list`

Returns list of (sorted) parent entries of self as (name, oid) pairs (as list of list). Name is the name in the parent directory.

Procedure `fire (trigger-arg:, oid:objectid, sel:symbol, args:list) :`

Default/dummy fire catch.

Procedure `for-attributes (what:compound/primitive) :`

Call what for each attribute. what is called with the name and value of the attribute as its respective parameters.

Function `get-info (how-fmt:symbol) :string/boolean`

Return help string or false about self. The string is formatted according to , `'raw` means no formatting, `'info` means appropriate for display output and `'tex` means for inclusion in `TEX` documents. The formatter actually is a macroprocessor, where macronames are alphabetical strings prefixed with `Macros` have a possibly empty parameter list started with `"` and terminated with `"`, parameters separated with `,`. From within the replacement string of a macro a parameter is referred to by `"0` to `9` and expanding to the `jn`th positional parameter, `0` expanding to the macro name. When formatting approximately the same rules as with `TEX` are used, multiple spaces are converted to 1 space, more than 1 consecutive linefeed means a new paragraph.

The following commands are available (incomplete):

- `varname:`. Reference to variable/parameter name.
- `valval:`. Value.
- `quoteval:`. Quote val.
- `strval:`. String quote val.
- `kbdval:`. Keyboard quote val.
- `symbval:`. Symbol 'val.

Note: For other macros, see implementation (alas, ???).

Procedure `initialise (any:**, :) :objectid`

Default dummy initialise.

Procedure `ls (opts:symbol/string, :, :symbol/string) :`

A listing of information on self is printed. The amount of information is influenced by the options(optional; default empty string) which is a string containing the following letters. Case is insignificant except when denoted otherwise.

- *c: Print the class.*
- *d: Print a description.*
- *o: Print the object identifier.*
- *i: Print the info.*
- *v: Print the value.*
- *V: Pretty print the value.*
- *l: Combines options 'c', 'i', 'o' and 'v'.*
- *r: Recursive one level, directory entries on level deeper printed.*
- *R: Recursive, directory entries are recursively printed.*
- *n: Print available methods of object as class only.*
- *N: Print available methods of object as class, superclasses included.*
- *m: Print available methods of class of object only.*
- *M: Print available methods of class of object, superclasses included.*

Procedure `map-attributes (what:compound/primitive) :list`

Call what for each attribute. what is called with the name and value of the attribute as its respective parameters. Return a list containing the results of each call of what.

Procedure `set-derived-info (sel:symbol, kind:symbol/null/boolean, compare:compound/primitive/null/prio:integer/null/boolean, trigger:objectid/null/boolean, trigger-arg:) :objectid`

Sets information for <self, sel>. Kind(default 'none) specifies the way values are calculated: 'lazy means on demand, 'eager immediately when one of its dependents is changed, 'none means no derivation management is done (except when it was done anyway). Compare(default equal?) specifies the scheme routine used to compare the result values. Prio(default 0) specifies the priority (in the range 0..5) by which values are removed from the cache: 4 means never, 5 means shared, 0 means no caching is done. Trigger(default false) specifies a trigger, i.e. the object to which a will be sent when the value is changed or new. The will be passed its trigger-arg and the complete list of parameters of the actual invocation the trigger is set on. If kind, compare or prio are nil, the default values are used. Only for non album oms's. See dvcache-info.

Note: Not implemented correct.

Procedure `set-function-behaviour (selector:symbol, how:symbol, what:symbol) :objectid`

Set function behaviour of self. If what is neither 'cache or 'depend, the info is removed. If the method is not a function or is not defined for the class Self (in case of 'class), the info is not added. Return self. See :function-info.

Procedure `set-info (info:string) :objectid`

Set help information on self to info. Return self. See get-info.

Procedure `touch () :objectid`

Touch (w.r.t. dependency management) self. Return self.

6.2 Class

Class `Class`

Superclasses `Object`

`template: pair := ()`

"Class Class for classes. Creation of new classes and adding methods to a class. Creation of an object is done by method `new`, which creates an object which is initialised by the class specific method `initialise`. The attribute `template` contains the template for an instance. Its value is a list of tuples `(name, type, initial)`, resp. the name, type and initial value. The template is combined (at creation time) with the templates of super classes."

`Procedure ::edit-method (name:symbol) :`

Edit method selector of class `self`. If the method is non-existent, a template is given and after editing the definition is loaded. Otherwise, the existing method definition is replaced.

`Procedure :construct (args:**, :) :objectid`

Construct a class or extend an existing class with methods. Method `construct` is a wrap around for method `new`, method `add-procedure`, method `add-function` and method `set-info`. Return class. Arguments are commands with parameters. Commands (which are type symbols) with their parameters are:

- *name name .: symbol. Name by which class is known in directory `/class`. See `new`.*
- *super super .: class. A superclass of class. More than 1 is allowed. Default is class `Object`. See `new`.*
- *ivar name init .: symbol value. instancevariable (attribute) with initial value. See `new`.*
- *itypevar name type init .: symbol type value. instancevariable (attribute) with initial value and type.*
- *info info .: string. Information string for class. See `set-info`.*
- *proc interface executable info .: list lambda string. See `add-procedure`.*
- *func interface executable info .: list lambda string. See `add-function`.*
- *dfunc interface executable info .: list lambda string. Same as `func`, function is a function for dependency management is done.*
- *cfunc interface executable info .: list lambda string. Same as `dfunc`, function is a function for which its result is cached.*
- *class class .: class. Use this class to add methods instead of creating new one and adding methods to that one.*

Either the command `class` or a combination of `name`, `super` and `ivar` is used.

`Procedure :def-method (name:symbol, intf:pair, exec:compound, info:string, meth-class:Class) :Method`

Adds/redefines a method to the class `self` which may be invoked by selector `name`. The interface consists of a type `type-list` for the arguments and the result value respectively. `Info` specifies the help information. If a method with the selector `name` exists, its attributes are replaced with the new ones. `Meth-class` specifies to which class the created method will belong. See `def-procedure`.

`Procedure :print-tex () :`

Print class `Self` in `TeX` format. Not ready.

`Procedure :print-tex-sub () :`

Print class `Self` and subclasses in `TeX` format. Not ready.

`Procedure def-cached-function (name:symbol, intf:pair, exec:compound, info:string) :Method`

Adds/redefines a cached function method to the class `self`. See `:def-method`.

`Procedure def-depend-function (name:symbol, intf:pair, exec:compound, info:string) :Method`

Adds/redefines a depend function method to the class self. See :def-method.

Procedure `def-function` (name:symbol, intf:pair, exec:compound, info:string) :Method
Adds/redefines a function method to the class self. See :def-method.

Procedure `def-procedure` (name:symbol, intf:pair, exec:compound, info:string) :Method
Adds/redefines a procedure method to the class self. See :def-method.

Function `describe` (as-list:boolean) :list/string
Return description of self in a string. If the as-list(optional; default false) is true the description is returned as a list of strings and lists: {supers,name,attrs}. Supers and fields are lists of resp. names and tuples (name, type, initial). If no information is available empty strings are returned. Not ready.

Procedure `initialise` (name:symbol, template:list, supers:list) :objectid
Initialise a class named name with template as the description for the attributes and as sub-class of the super-classes. Multiple inheritance is allowed. The class can be found under name in the /class directory using method dir-lookup.

Function `instances` (deep?:boolean, :, :boolean) :list
Return instances of class self as list of oid's. If the deep(optional; default false) is passed also the instances of subclasses are returned.

Function `method` (selector:symbol, all:boolean) :objectid/boolean
Return method for class Self and selector or nil if no method exists. Search also in the super classes if all(optional; default false) is true.

Function `method-definition` (name:symbol) :string
Return a textual representation of method name.

Function `method-template` (name:symbol) :string
Return a template for new method name.

Function `methods` (all:boolean) :list
Return list of (name, method) pairs (as lists) for class Self. Return also the methods of the super classes if all(optional; default false) is true.

Procedure `new` (args:**, :) :objectid
Create a new object of the receiver object and send the method initialise with arguments to the resulting object. An aggregate containing the attributes of all super classes is constructed. Return the created object.

Function `sub-classes` (all:boolean) :list
Return list of sub classes of self. Return also the sub classes of the sub classes if all(optional; default false) is true.

Function `super-classes` (all:boolean) :list
Return list of super classes of self. Return also the super classes of the super classes if all(optional; default false) is true.

6.3 Environment

Class Environment

Superclasses Object

"Album only. Class Environment. Environments are objects which have a current snapshot associated with them. An environment can be used, by several users simultaneously, and works with the current associated snapshot. A default environment can be found in directory entry /env/plain."

Procedure `change-snapshot` (snapshot:Snapshot) :objectid

Album only. Change current snapshot associated with environment self. If an environ-

ment process is running, the new snapshot image is loaded and the old one saved if necessary. The old snapshot is returned.

Function `current-snapshot ()` :Snapshot

Album only. Return current snapshot associated with environment.

Procedure `initialise (snapshot:Snapshot, name:symbol)` :objectid

Album only. Initializes an environment with snapshot and name. The environment can be referred to by directory name /env/name.

Procedure `stop (user-id:symbol)` :objectid

Album only. Stop the use of an environment self under is user-id. The resulting snapshot is returned. The environment-oms process is also killed if no other user is using the environment. See use.

Note: Argument user-id is not used.

Procedure `use (user-id:symbol)` :integer

Album only. Use environment self under id user-id., create a new oms process on the same machine the album process is running. The socket identifier at which the environment process is listening is returned. This identifier can be used as the socket parameter of oms-connect.

Note: Argument user-id is not used.

Note: Every invocation of method stop should follow an invocation of method use since detection of removed processes etc. is not implemented. Removing an environment without invoking a method stop makes the system think the environment still runs.

6.4 File

Class File

Superclasses Object

value: string/lfieldid :=

"Class File for values, string of lfield. Subclasses used for containing scheme sources resp. oms sources are class Scheme-Src and class Oms-Src.

Note: used to be class Edit. The attribute value contains the actual value."

Procedure `::edit (display:string)` :objectid

Edit object by invoking system editor which is editor found in the unix environment variable or vi . Will be removed !!

Function `get-string-value ()` :string/lfieldid

Return value of attribute value as a string.

Function `get-value (as-lfield?:boolean)` :string/lfieldid

Return value of attribute value as a string. If the as-lfield?(optional; default true) is true the returned value may be a lfieldid.

Procedure `initialise (value:string/lfieldid)` :objectid

Initialise value object self with string value(optional; default "").

Procedure `set-value (val:string/lfieldid)` :objectid

Set value attribute of self to val.

6.5 Method

Class Method

Superclasses Object

interface: pair := ()

executable: pair := ()

"Class Method for methods in general. A method is an object which can execute code. A method has no name and is related to objects and classes via relation method-of. The attribute interface is a <arg-list, res-list>, containing type-lists for checking the arguments resp. the result. The attribute executable contains the actual code."

Function `:executable-expr () :string`
Return for method self a description of the executable

Function `:interface-expr () :string`
Return for method self a description of the interface

Procedure `:print-tex () :`
Print method self in $T_{E}X$ format. Not ready.

Function `definition-classes () :list`
Return list of <class, name> pairs for which self is a method.

Function `describe (as-list:boolean) :list/string`
Return description of self in a string. If the as-list(optional; default false) is true the description is returned as a list of strings and lists: {class,method-type,name,arguments,result}. The arguments and result both are lists of pairs (lists): {name,type}. If no information is available empty strings are returned.

Procedure `eval (actuals:**, :) :`
Evaluate function method self with arguments actuals.

Procedure `initialise (interface:pair, executable:compound) :Method`
Create a new method with interface and executable. Return self.

6.6 Piece

Class `Piece`

Superclasses `Object`

"Class Piece describes Pieces on Snapshot-level. An instance of class Piece is an object which contains a set of objects and a set of relationships."

Procedure `add-objects (oids:list/objectid) :objectid`
Adds one or more objects to a piece.
Note: Adds the tuple <oid, self> to the relation part-of. It also adds the following tuples to the relation intern-relationships, using the method add-relationship:

- : <self, instance-of, oid, class-of-oid>
- : <self, part-of, oid, self>

Procedure `add-relationship (relation:System-Relation, tuple:**, :) :objectid`
Adds a relationship to a piece.
Note: Adds the tuple <self, relation, field1, field2, ..> with field_i the value of ith field of tuple to the relation intern-relationships.

Procedure `delete-objects (oids:list/objectid) :objectid`
Deletes an object and corresponding relationships from a piece.
Note: Deletes the tuple <oid, self> from the relation part-of. It also deletes the following tuples from the relation intern-relationships, using the method delete-relationship:

- : <self, instance-of, oid, class-of-oid>
- : <self, part-of, oid, self>

Procedure `delete-piece (:) :`
Deletes the piece self, i.e. self (??), the objects and relationships.

Procedure `delete-relationship (relation:System-Relation, tuple:**, :) :objectid`
Deletes a relationship from a piece.

Note: Deletes the tuple $\langle \text{self}, \text{relation}, \text{field1}, \text{field2}, \dots \rangle$ with field_i the value of i th field of tuple from the relation `intern-relationships`.

Procedure `for-each-object (what:compound/primitive) :`

For each object in piece (recursively) self do what(oid).

Function `map-object (what:compound/primitive) :list`

For each object in piece (recursively) self do what(oid). Return list of function results. Equivalent of map.

Function `objects (all?:boolean) :list`

Return objects in piece self. If all(optional; default false) return objects recursively.

6.7 Snapshot

Class `Snapshot`

Superclasses `Object`

`timestamp: := ()`

Album only. Class `Snapshot`. Snapshots are added under a name derived from the timestamp under the directory `/snapshot`. A default snapshot can be found in directory entry `/snapshot/plain`.

Note: A timestamp is a tuple $\langle \text{time}, \text{count}, \text{pid} \rangle$, which need not necessarily be unique if at the same time, on different machines, by processes with the same pid a timestamp is created."

Procedure `::new-from-file (file:symbol/string) :Snapshot`

Album only. Read the contents from file to create a new snapshot derived from self. Contents of file-name is previously created with method `-i file`. If `keep-objects` is true the old objects from the snapshot are not deleted.

Procedure `::to-file (file:symbol/string, lfields?:boolean) :`

Album only. Dump snapshot of self on file-name. If the `lfields?` (optional; default false) is true, the values of the used longfields are included in the dump.

Function `:object-to-bstring () :lfieldid/string`

Album only. Returns a bstring that represent the snapshot self.

Procedure `contents->bstring () :lfieldid/string`

Album only. Returns the value of the contents of the snapshot self as a bstring

Procedure `contents->list () :list`

Album only. Returns the value of the contents of the snapshot self as a list

Procedure `eval (proc:compound/primitive) :`

Album only. Evaluates lambda-expression proc with no arguments inside context of snapshot self.

Procedure `extract-named-piece (name:symbol/string, piece-name:symbol/string) :Album-Piece`

Album only. Makes an external representation of the piece piece-name in snapshot self. This external representation is an instance of the class `class Album-Piece`, reachable by name

Procedure `extract-piece (name:symbol/string, piece:objectid/list) :Album-Piece`

Album only. Makes an external representation of piece in snapshot self. This external representation is an instance of the class `Album-Piece`.

Note: A longfield is created containing the following set of lists

- *: ('object oid (object-value oid))*
- *: ('lfield fid (lfield-content fid))*
- *: ('relationship oid tuple1 tuple2 .. tuplen)*

for each object in the specified piece and each lfield used as value of one of the objects and all relationships member of the relation intern-relationships.

Procedure `for-each-object` (action:compound/primitive, which:compound/primitive)
:

Album only. Execute action (objectid, value) for each objectid in the list returned by which. Both functions are executed in the context of snapshot self. The function which(optional; default a function returning all objects of the snapshot) determines which objects are iterated on.

Procedure `initialise` (from:**, stamp:**, :) :Snapshot

Album only. Initialize snapshot self. Only for internal use.

Function `lookup-in-snapshot` (obj:symbol) :objectid

Album only. Return object identifier of object with name obj inside the snapshot self

Procedure `piece->list` (piece:objectid/list) :list

Album only. Returns a list that contains a representation of the piece piece in the snapshot self.

Function `timestamp` () :timestamp

Album only. Return timestamp of self.

Procedure `transplant-piece` (piece:objectid) :Snapshot

Album only. Creates a new snapshot by transplanting album-piece in snapshot self using the external representation created by the method `extract-piece`.

6.8 System

Class System

Superclasses Object

"Class System for system dependencies and purposes. A class with one object as it's instance with directory name /system."

Procedure `:def-prop` (key:symbol, access:compound/primitive/symbol, init:) :

Define system property named by key, accessed by access and initial value init. Access is either a symbol or a procedure, in the first case the name of the corresponding scheme variable, in the second case the procedure used for accessing the value. See `set-prop`.

Procedure `:dump-image` (file:symbol/string, lfields?:**, :) :

Dump the current oms on file-name. If lfields?(optional; default false) is true, the values of the used longfields are included in the dump.

Note: method will be removed.

Procedure `:load-image` (file:symbol/string) :

Replace the current oms with the dump (previously created with method `dump-image`) from file-name.

Procedure `:load-scheme` (file:symbol/string) :

Load scheme code from file.

Procedure `:print-tex-system` () :

Print documentation of the system in T_EX format. Not ready.

Procedure `:prop` (key:symbol/boolean/list, value:, set?:boolean/null) :

If set? set (with value), otherwise retrieve property key. If key is nil a list of all (property, value) pairs is returned.

Procedure `apropos` (key:symbol/string, print?:boolean) :list

Return a list of objects for which key can be found in one of the relation `object-info`, `relation method-of` and `relation directory`. If print?(optional; default false) the oid's plus

information is printed.

Procedure `bstring-to-object (lfield:lfieldid/string, new-oid?:boolean) :objectid`
Album only. Uses the external representation created by the method `:object-to-bstring` to create a new object.

Function `get-prop (key:symbol) :`

Get system property named by key. One of its purposes is to maintain (system) properties influencing and/or describing system behaviour. Current system properties are:

- `trans-logging?.`: *boolean, true if transformations, i.e. sends are logged. Default is false.*
- `trans-log.`: *list, list of transformation log. Setting the value sets the list to nil. Default is nil.*
- `logging?.`: *boolean, true if miscellaneous system actions are logged. Default is false.*
- `check-method-type?.`: *boolean, true if argument and return values of method invocations are checked against the required values. Default is true.*
- `check-relation-type?.`: *boolean, true if tuples are checked before added to a relation. Default is true.*
- `when-re-eval.`: *symbol, determines if re-evaluation takes place immediately after setting an attribute value or after evaluation of a user command or explicitly. The alternatives are indicated by the values `'immediate`, `'user-cmd` and `'explicit`. Default is `'explicit`.*
- `on-delete-object.`: *symbol, specifies what to do when an object is deleted. Possible values are `'none` for no action at all, `'warn` for checking – and warning about – consistency, i.e. checking if references to the deleted object from relations still exist. `'error` does the same but creates an error situation instead of an warning only. `'delete` silently deletes all references to the deleted object. Default is `'delete`.*
- `do-time-send.`: *boolean, if true, timing of the top level send will be done.*

Procedure `getenv (key:symbol/string) :string/boolean`

Return the value of the unix environment variable key or, if non existing, false.

Procedure `import-from-unix (file:string) :objectid`

Album only. Load an externally stored Album object from a unix file file.

Procedure `receive-from-unix (file:string, objname:string) :objectid`

Album only. Load an externally stored Album object from a unix file file. Put it in the directory `/inbox` under a given name `objname`.

Procedure `set-prop (key:symbol, value:) :`

Set system property named by key to value. See `get-prop`.

Procedure `unix-getenv (key:symbol/string) :string/boolean`

Album only. Return the value of the unix environment variable key or, if non existing, false.

6.9 System-Relation

Class `System-Relation`

Superclasses `Object`

`value: relation := ()`

”Class for system created/maintained relations. A relation is a set of relationships (or tuples). Each tuple is a `{field,...}`. Each relation is named in the directory `/relation` with the name supplied as a parameter of method `initialise`. The value is kept in attribute `value`.”

Procedure `:print-tex () :`

Print relation self in T_EX format. Not ready.

Function `describe (as-list:boolean) :list/string`

Return description of self in a string. If `as-list`(optional; default false) is true the description is returned as a list of strings and lists: {relation-type,name,fields}. The fields is a list of types. If no information is available empty strings are returned.

Procedure `initialise (name:symbol, relation:relation, info:string) :objectid`

Initialize system relation self by self and scheme relation relation. Used internally only.

Function `query (pattern:pair) :list`

Query (system) relation self by pattern. Returns the tuples found in the relation matching pattern.

Function `template () :list`

Return the template of self used for type checking.

Function `value () :relation`

Return value (= stored tuples) of relation self.

6.10 Album-Piece

Class `Album-Piece`

Superclasses `File`

"Album only. Class Album-Piece. The instances of the class Album-Piece are external representations of pieces occurring in specific snapshots."

6.11 Scheme-Src

Class `Scheme-Src`

Superclasses `File`

"An editable object containing scheme source code."

6.12 Unix-Tree

Class `Unix-Tree`

Superclasses `File`

`status: list := ()`

"Class Unix-Tree for incorporating unix-files in oms. The attribute status contains some status values as returned by file-status."

Procedure `export (unix-dir:symbol, rec:boolean) :`

Export self to the directory unix-dir. If the parameter rec is set true, and if self is represented as a directory, all subdirectories of self are also exported. The fact whether self is a directory is derived from the attribute status. If necessary, the user is asked to confirm overwriting existing files.

Procedure `export-entry (unix-dir:symbol, rec:boolean) :`

Exporting self; recursive slave of 'export.

Procedure `import (unix-dir:symbol, rec:boolean) :`

Importing Unix-files in oms under self. If the value of parameter rec is true, and if the specified path is a directory, all sub-directories under path are included. Under the oms-directory / there is a default directory-entry unix (which is an instance of the class Unix-Tree), representing the root of the oms-file-tree.

Procedure `import-entry (unix-dir:symbol, unix-name:symbol, rec:boolean) :`

Importing a directory-entry under self. Recursive slave of 'import.

Procedure initialise (contents:string/lfieldid, stat:list) :
Initialise for a new instance of this class. The contents of a file is to be stored in attribute value. The status of this file is stored in attribute status. See file-status for a description of attribute status.

Function status () :symbol
Returns de first element of the attribute status of self.

6.13 Function

Class Function

Superclasses Method

"Class Function for methods with functional behaviour. A class Procedure is a class Method, but a class Function is a class Procedure which is not allowed to generate side-effects (i.e. changing attribute values). For functions derivation management is performed and specific values are cached. An instance of class Function may not call an instance of class Procedure."

Function eval (actuals:**, :) :
Evaluate function method self with arguments actuals.

6.14 Procedure

Class Procedure

Superclasses Method

"Class Procedure for methods which may have side effects. Procedure invocations are logged to for use with snapshot transformations. See Function."

6.15 Relation

Class Relation

Superclasses System-Relation

"Class Relation for user created/maintained relations."

Procedure add (tuples:list, :, :list) :Object
Add to relation self tuples.

Procedure delete (patterns:**, :) :Relation
Delete a (set of) tuple(s) matching pattern.

Procedure initialise (name:symbol, template:pair, info:string, mask:list) :objectid
Initialize relation where each tuple has to match template. The list of integers mask(optional; default all fields) specifies on which fields a search index must be made. name is used to create an entry in the relation directory under the entry /relation, info is added to the relation object-info.

Procedure update (pat:list, update:compound/primitive) :Object
Update by replacing every tuple matching pat with the result of calling update with pat.

Procedure update-or-add (pat:list, update:compound/primitive, value:) :Object
Either update by replacing every tuple matching pat with the result of calling update with pat or add to the tuple value.

6.16 Oms-Src

Class `Oms-Src`
Superclasses `Scheme-Src`
"An editable object containing Camera oms scheme source code."
Procedure `load (nil:**, :)` :
Load value of self as scheme oms code

6.17 Computed-Relation

Class `Computed-Relation`
Superclasses `Relation`
"Class Computed-Relation for combination of relation and computed tuples. The value of an instance of class Computed-Relation is the union of its value and the results of the evaluated functions. A recipe is an instance of class Function accepting one parameter: the oid of the instance of class Computed-Relation."
Procedure `add-recipe (recipe:Function) :Computed-Relation`
Add to computed relation self a recipe to compute value.
Procedure `del-recipe (recipe:Function) :Computed-Relation`
Delete from computed relation self the recipe.
Function `stored-query (pattern:pair) :list`
Returns the tuples found in the stored part of relation matching pattern.
Function `stored-value () :relation`
Return stored value (= stored tuples) of relation self.
Function `value () :relation`
Return value (= stored tuples + computed tuples) of relation self.

7 Relations

System-Relation `:derived-value:` `objectid × symbol × list × integer ×`
"Cache for derived values."
System-Relation `:dvcache-info:` `objectid × symbol × symbol × compound/primitive × integer × objectid/boolean ×`
"oid x selector x kind x compare x prio x trigger x trigger-arg: Config info for cached values. Kind may have the values 'none, 'lazy for lazy re-evaluation and 'eager for eager re-evaluation. Compare is the comparison routines used for comparing derived values. Prio will be used to determine the importance of a cached value. Trigger is the object upon which the will be invoked with first arg trigger-arg if the value is new and/or not cached or changed. This will only be done if dependency analysis is performed. The fires will be triggered in an indeterminate order after completion of the top level dependency function invocation. See set-derived-info."
System-Relation `:function-info:` `objectid × symbol × symbol × list`
"object x selector x how x what: Config info for functions w.r.t. dependency management. What may include 'cache and 'depend as values denoting the fact that caching resp. dependency analysis is done. Caching implies dependency analysis. How may be 'class or 'object, the first denoting the fact that the behaviour should be done for all objects of the class object and method selector, the second only for the object itself. The latter overrules the first."
Relation `:system-property:` `symbol × symbol/primitive/compound ×`

"key x access x value. Containing system properties."

System-Relation **depends-on**: `objectid × symbol × list × symbol × symbol ×`
"Dependencies between method invocations/attributes."

System-Relation **directory**: `objectid × symbol × objectid`
"Object x symbol x Object: directory structure."

Relation **env-current-snapshot**: `Environment × Snapshot`
"Album only. Environment x Snapshot: current snapshot of environment."

Relation **env-current-use**: `Environment × symbol × × integer`
"Album only. Environment x symbol x rpcport x integer: user, port, socket for an environment."

System-Relation **instance-of**: `objectid × objectid`
"Object x Class: instance-of hierarchy."

Relation **intern-relationships**: `objectid × System-Relation ×`
*"Object X Relation X **: describes which relationships belong to piece."*

System-Relation **method-of**: `objectid × symbol × objectid`
"Class x symbol x Method: method's for class."

System-Relation **object-info**: `objectid × string`
"Object x string: information about objects (this string for instance)."

Relation **part-of**: `objectid × objectid`
"Object X Object : describes which objects belong to piece"

Relation **recipe-of**: `Computed-Relation × Function`
"Relation: Computed-Relation x Function: functions for a computed relation."

Relation **snapshot-history**: `Snapshot × Snapshot ×`
"Album only. Snapshot x Snapshot x transformation??: Derivation history."

System-Relation **super-class**: `objectid × objectid`
"Class x Class: inheritance hierarchy."

7.1 Naming of objects

See appendix C.

8 Extensions of scheme, routines for the implementor

For implementing methods in scheme extra routines are available.

8.1 Oms in general

`attribute-set (attribute: symbol, value: elementary-type) → (self)` interface

Description. Sets the value of attribute of self to value.

`attribute-get (attribute: symbol) → (value: elementary-type)` interface

Description. Retrieves the value of attribute of self.

self () \rightarrow (object: oid) interface

Description. Returns the current receiver object, i.e. to which object the message was sent or nil if not called from a method.

sender () \rightarrow (object: oid) interface

Description. Returns the current sender object, i.e. which object the message sent or nil if not sender.

msg-class () \rightarrow (object: oid) interface

Description. Returns the oid of the class associated with the current activated method.

8.2 Bytestring and longfield

A *longfield* is an oms type representing a file. Identification is done on the basis of *lfieldid*'s, having the same structure as *objectid*'s. Manipulation of *longfields* is done by a *longfieldserver* which is accessed via UNIX sockets and some *rpc* routines. Use of *longfields* is primarily done via *bstrings*. A *bstring* is either a *string* or a *lfieldid*, depending on the size of the actual data contained in the *bstring*. The following routines are available:

bstring?(value) \rightarrow (boolean) interface

Description. Return true if value is a bstring.

bstring-add! (value: string, old: bstring) \rightarrow (bstring) interface

Description. Add value as a new bstring, where old (optional) is the bstring value originates from. A variant is *bstring-file-add!* (or *file- δ bstring*) which takes the value from value as a file.

bstring-add-exprs! (add: lambda) \rightarrow (bstring) interface

Description. Add scheme expressions in a bstring. Adding is done by add which is passed a routine writing a scheme expression, passed as its only parameter, to the bstring.

bstring-content (value: bstring, start: integer, length: integer) \rightarrow (string) interface

Description. Return the contents of `value` as a string. The optional parameters `start` and `length` specify the start and length of the content returned. Default is 0 resp. the size of `value`.

`bstring-content-exprs` (`bstr: bstring,` `get: lambda`) \mapsto (`value`) interface

Description. Read scheme expressions from a `bstring`. Reading is done by the routine `get` which is passed a parameterless routine returning expressions one by one until end-of-file is encountered and returned. The value returned by `get` is returned.

`bstring-del!` (`bstring`) \mapsto () interface

Description. Delete the `bstring`.

`bstring-length` (`bstring`) \mapsto (`length: integer`) interface

Description. Return the size of the `bstring`.

`bstring-!file` (`value: bstring,` `file: string`) \mapsto () interface

Description. Produce a copy of `value` in the file.

`bstring-equal?` (`b1: bstring,` `b2: bstring`) \mapsto (`boolean`) interface

Description. Return true if both `bstrings` are equal .

8.3 Miscellaneous

`depend-re-eval` () \mapsto () interface

Description. Initiate propagation of (in)consistency information and reevaluate `eager` values. A shorthand is `::`.

`wr` (`values`) \mapsto (`value`) interface

Description. Print values on standard output, return last of values.

`proto-is` (`kind: symbol`) \mapsto (`-: boolean`) interface

Description. Return true if prototype is running a kind oms. kind may take the values album, oms, itf.

proto-itf-is (kind: symbol) → (-: boolean) interface

Description. Return true if prototype is running a kind interface. kind may take the values album, env, itf.

8.4 Scheme extensions

8.4.1 General utilities

first (list) → (value) interface

Description. Return first element of list. Equivalent routines: *second*, *third*, *fourth*, *fifth*, *sixth*, *seventh*, *eighth* and *ninth* exist.

for-each-while (value: list, action: lambda, terminal) → (value) interface

Description. Iterate on the elements of the list value, for each element action is called with the element as its parameter. This process stops when no more elements are found in the list or action returns terminal. In the first case terminal is returned, in the second case the last value returned by action is returned.

list-head (list, nr) → (list) interface

Description. Return the list consisting of the first nr elements of list.

list-set-car! (list, nr, value) → (value) interface

Description. Set the nr element (i.e. car field) of list to value.

strings-join-string (strings: {string, ...}, separator: string) → (string) interface

Description. Return the concatenation of the strings, separated by separator. separator optional, default is " ".

string-join-strings (string, separator: string) → (values: {string, ...}) interface

Description. Return the argument string as a list of strings. Separation is done by a sequence of characters from `separator`. `separator` is optional, default is " \r\n\t".

walk $\left(\begin{array}{l} \text{value,} \\ \text{action: lambda,} \\ \text{recurse-on: \{symbol, \dots\}} \end{array} \right) \mapsto (\quad)$ interface

Description. Iterate on the elements of `value`, for each element `action` is called with the element as its parameter. For values/elements which have a type found in `recurse-on` `action` is not called but the iteration is done recursively on the element.

Implementation. Recursion will only be done for elements of type `vector`, `list`, `string`, `relation`, `hashtable` and `storage`.

8.4.2 Hash tables and sets

The builtin type `hashtable` provides for fast access of $\langle \text{key}, \text{value} \rangle$ pairs. Both the `key` and the `value` may be any scheme value. Sets – using hashtables – are also implemented, set equivalent routines of the hashing routines are mentioned where appropriate.

make-hashtable (size: integer \quad) \mapsto (hashtable \quad) interface

Description. An empty hashtable of size `size` is returned. The corresponding set routine is `make-set`. The size parameter is then optional.

hashtable?(scheme object \quad) \mapsto (boolean \quad) interface

Description. Return true if `scheme object` is a hashtable. The corresponding set routine is `set?`.

hashtable-length (hashtable \quad) \mapsto (integer \quad) interface

Description. Return the number of key/value pairs in the hash table. The corresponding set routine is `set-length`.

hashtable-list (hashtable \quad) \mapsto (list \quad) interface

Description. Return list containing the key/value pairs (in a `car/cdr` pair) of the hashtable. The reverse function is `list-hashtable`.

hash-set! $\left(\begin{array}{l} \text{hashtable,} \\ \text{key,} \\ \text{value} \end{array} \right) \mapsto (\quad)$ interface

Description. Add the key/value combination to the hash table. Old entries are overwritten. The corresponding set routine is `set-add!`. For this routine the `value` must not be supplied, `true` is used instead.

hash-del! $\left(\begin{array}{l} \text{hashtable,} \\ \text{key} \end{array} \right) \mapsto (\quad)$ interface

Description. Remove the entry for `key`. The corresponding set routine is *set-del!*.

hash-ref $\left(\begin{array}{l} \text{hashtable,} \\ \text{key} \end{array} \right) \mapsto (\text{value} \quad)$ interface

Description. Return the `value` belonging to `key` or the empty list if no value is found. The corresponding set routine is *set-has?*.

hash $\left(\begin{array}{l} \text{scheme value,} \\ \text{bound: integer} \end{array} \right) \mapsto (\text{integer} \quad)$ interface

Description. Return the hash value of the `scheme value`. If the optional parameter `bound` is supplied the hash value is restricted to the range $0 \dots \text{bound} - 1$.

hashtable-for-each $\left(\begin{array}{l} \text{lambda,} \\ \text{hashtable} \end{array} \right) \mapsto (\quad)$ interface

Description. Call `lambda` with two parameters `key` and `value` for each of the entries in the hashtable. The corresponding set routine is *set-for-each*. The `value` parameter is then not supplied.

-set! $\left(\begin{array}{l} \text{from: hashtable,} \\ \text{sub: hashtable} \end{array} \right) \mapsto (\quad)$ interface

Description. Subtract `sub` from `from`.

8.4.3 Comparison, sorting

8.4.4 Unix interface

Order of parameters of UNIX system calls is roughly the same as in described in the manual entries for the system calls. In general the error code *errno* (or zero) is returned.

chmod $\left(\begin{array}{l} \text{path,} \\ \text{mode: integer} \end{array} \right) \mapsto (\text{errno: integer} \quad)$ interface

Description. Change the mode of the file.

directory-names $(\text{path} \quad) \mapsto (\{\text{string, ...}\} \quad)$ interface

Description. Return list of names in directory.

file-status (path \rightarrow) \rightarrow ({kind,mode,mode,inode,n)link,uid,gid} interface

Description. Return stat info of file.

link (path1, path2 \rightarrow) \rightarrow (errno: integer) interface

Description. Link two files.

mkdir (path mode: integer \rightarrow) \rightarrow (errno: integer) interface

Description. Create directory with mode.

path-split (path: symbol-or-string \rightarrow) \rightarrow ({directory,name,suffix}) interface

Description. Splits the UNIX pathname path into a directory, name and suffix. Separators are '/' and '.'. Result is returned as a list of three strings.

sleep (period: integer \rightarrow) \rightarrow () interface

Description. Sleep period seconds.

unlink (path, \rightarrow) \rightarrow (errno: integer) interface

Description. Unlink (remove) file.

8.4.5 Relations

8.5 Others

See appendix B, the sources of the prototype or ask the author.

9 Changes and bugs

9.1 Since version 1.32

- Changed interfaces: attribute-get, attribute-set, add-procedure, add-function, initialise for relations
- New features: Deletion of objects and directory entries, help information associated with objects, class constructing, longfields, bytestrings, looking up name of directory, timing of send's, etc.
- Changed behaviour: Relations implemented with hashtables, storage of snapshots.

9.2 Since version 1.41

- Names of classes are capitalized.
- Snapshot storage by means of server. Added functionality for snapshots.
- Rpc mechanism implemented faster.
- System initialization via `camerarc` files.
- New (general) methods: `ls`. Used to be `dir-tree-print`.
- Documentation about hashtable, unix interface.
- Type checking mechanism extended.

9.3 Since version 1.46

- New: editing of methods, reading of unix files/directories, apropos, regular expressions.
- Changed: (part of the) implementation of objects and methods coded in C. Names of object are printed when an objectid is printed.
- Changed interface: `new` for relations.

9.4 Since version 1.49

- New: pieces.
- Changed: interface to oms only via `send`. Edit-like methods replaced with value/set-value pairs.
- Changed interface: `new` for relations.
- Removal of class `Edit`, replaced with class `Value`.

9.5 Since version 1.50

- Doc for classes included in oms.

9.6 Since version 1.52

- Dependency and cache mgt only for special Function classes.

9.7 Unsolved bugs, problems and future work

- After an error in a method for which dependency analysis is done, the analysis is not reset. Typing `(set! depend-method-stack nil)` helps.
- System properties, dependency info and cache values are not stored in snapshots.
- The number of symbols is limited (system restriction). Symbol tables should automatically extend themselves or string should be used instead.
- Temporary objects, i.e. created and deleted between creation of 2 snapshots are marked as deleted in the newer snapshot. This is unnecessary.
- Dependency management must/will be revised.

A Example session

A.1 Album

```
spirit[62]% ./camera album 9903
Camera [pid 4846] type album, interface album, socket 9903.
$Revision: 1.60 $ $Date: 1993/10/08 07:58:14 $
Load: (1.28+0.00=1.28 u) + (0.47+0.00=0.47 s) = 1.75 t
Camera [pid 4847] type lfield, interface lfield, socket 9914.
$Revision: 1.60 $ $Date: 1993/10/08 07:58:14 $
Load: (0.92+0.00=0.92 u) + (0.22+0.00=0.22 s) = 1.13 t
Camera [pid 4848] type store, interface store, socket 9924.
$Revision: 1.60 $ $Date: 1993/10/08 07:58:14 $
Load: (0.87+0.00=0.87 u) + (0.30+0.00=0.30 s) = 1.17 t
No image for #[timestamp 11 11 11], initializing from scratch:
directory-a, method, relation, depend, directory-b, system, value, piece, scmedit, unixtree, snapshot, environment, album-p
took (4.72+0.00=4.72 u) + (0.60+0.00=0.60 s) = 5.32 t
Current stamp = #[timestamp 695298030 0 4848]
Camera [pid 4852] type oms, interface env, socket 9934.
$Revision: 1.60 $ $Date: 1993/10/08 07:58:14 $
Load: (1.35+0.00=1.35 u) + (0.27+0.00=0.27 s) = 1.62 t
Current stamp = #[timestamp 11 11 11]
Camera [pid 4856] type oms, interface env, socket 9944.
$Revision: 1.60 $ $Date: 1993/10/08 07:58:14 $
Load: (1.25+0.00=1.25 u) + (0.43+0.00=0.43 s) = 1.68 t
Current stamp = #[timestamp 695298219 0 4848]
Camera [pid 4859] type oms, interface env, socket 9954.
$Revision: 1.60 $ $Date: 1993/10/08 07:58:14 $
Load: (1.33+0.00=1.33 u) + (0.38+0.00=0.38 s) = 1.72 t
Current stamp = #[timestamp 695298825 0 4848]
spirit[63]%
```

A.2 Interface

```
spirit[43]% ./camera
Camera [pid 4851] type itf, interface itf, socket ().
$Revision: 1.60 $ $Date: 1993/10/08 07:58:14 $
Load: (0.92+0.00=0.92 u) + (0.28+0.00=0.28 s) = 1.20 t
Itf> (<< "localhost" 9903)
0
Itf> (>> 0)
()
Client(0 = [9903@localhost]) > (<< #@/env/plain 'use 'atze)
9934
Client(0 = [9903@localhost]) > (<< "localhost" 9934)
1
Client(0 = [9903@localhost]) > (>> 1)
#[rpcport nr=0 flags=0x0 sock=9903 pid=0 host=localhost info=#[objectid 0.0.0.0 0 3]]
Client(1 = [9934@localhost]) > (<< #@/system 'load-scheme 'demo.scm)

Client(1 = [9934@localhost]) > (send #@/src/hello.c 'value)
main()
{
    printf( "Hello world\n" );
}

Client(1 = [9934@localhost]) > (get-edit-set #@/src/hello.c 'value)
#[objectid 192.87.7.19 4852 5]
Client(1 = [9934@localhost]) > (send #@/src/hello.c 'value)
main()
{
    printf( "Hello worldsssss\n" );
}

Client(1 = [9934@localhost]) > (<< #@/ 'ls 'r)
```

```

#[objectid 0.0.0.0 0 3 (Object)root] root
  #[objectid 192.87.7.19 4852 10 (Object)tex-src] tex-src
    #[objectid 192.87.7.19 4852 13 (Tex-Source)doc.sty] doc.sty
    #[objectid 192.87.7.19 4852 12 (Tex-Source)doc.tex] doc.tex
    #[objectid 192.87.7.19 4852 11 (Tex-Source)hello.tex] hello.tex
  #[objectid 192.87.7.19 4852 4 (Object)src] src
    #[objectid 192.87.7.19 4852 5 (C-Source)hello.c] hello.c
#[objectid 0.0.0.0 0 105 (Unix-Tree)unix] unix
#[objectid 0.0.0.0 0 80 (Object)pieces] pieces
#[objectid 0.0.0.0 0 65 (System)system] system
#[objectid 0.0.0.0 0 2 (Class)Class] class
  #[objectid 192.87.7.19 4852 6 (Class)Tex-Source] Tex-Source
  #[objectid 192.87.7.19 4852 1 (Class)C-Source] C-Source
  #[objectid 0.0.0.0 0 102 (Class)Dvi-File] Dvi-File
  #[objectid 0.0.0.0 0 95 (Class)Unix-Tree] Unix-Tree
  #[objectid 0.0.0.0 0 82 (Class)Oms-Src] Oms-Src
  #[objectid 0.0.0.0 0 81 (Class)Scheme-Src] Scheme-Src
  #[objectid 0.0.0.0 0 71 (Class)Piece] Piece
  #[objectid 0.0.0.0 0 66 (Class)Value] Edit
  #[objectid 0.0.0.0 0 66 (Class)Value] Value
  #[objectid 0.0.0.0 0 57 (Class)System] System
  #[objectid 0.0.0.0 0 34 (Class)Computed-Relation] Computed-Relation
  #[objectid 0.0.0.0 0 30 (Class)Relation] Relation
  #[objectid 0.0.0.0 0 27 (Class)System-Relation] System-Relation
  #[objectid 0.0.0.0 0 7 (Class)Function] Function
  #[objectid 0.0.0.0 0 6 (Class)Procedure] Procedure
  #[objectid 0.0.0.0 0 5 (Class)Method] Method
  #[objectid 0.0.0.0 0 1 (Class)Object] Object
  #[objectid 0.0.0.0 0 2 (Class)Class] Class
#[objectid 0.0.0.0 0 4 (Object)relation] relation
  #[objectid 0.0.0.0 0 79 (Relation)intern-relationships] intern-relationships
  #[objectid 0.0.0.0 0 78 (Relation)part-of] part-of
  #[objectid 0.0.0.0 0 56 (System-Relation)system-property] system-property
  #[objectid 0.0.0.0 0 48 (System-Relation)directory] directory
  #[objectid 0.0.0.0 0 46 (System-Relation)dvcache-info] dvcache-info
  #[objectid 0.0.0.0 0 45 (System-Relation)derived-value] derived-value
  #[objectid 0.0.0.0 0 44 (System-Relation)depends-on] depends-on
  #[objectid 0.0.0.0 0 43 (System-Relation)method-of] method-of
  #[objectid 0.0.0.0 0 42 (System-Relation)super-class] super-class
  #[objectid 0.0.0.0 0 41 (System-Relation)instance-of] instance-of
  #[objectid 0.0.0.0 0 40 (System-Relation)object-info] object-info
  #[objectid 0.0.0.0 0 39 (Relation)recipe-of] recipe-of

```

```

Client(1 = [9934@localhost]) > (>> 0)
#[rpcport nr=1 flags=0x0 sock=9934 pid=0 host=localhost info=#[objectid 0.0.0.0 0 3]]
Client(0 = [9903@localhost]) > (<< #@/env/plain 'stop 'atze)
#[objectid 192.87.7.19 4846 129]
Client(0 = [9903@localhost]) > (>> 1)
[4851] #f: interface error: invalid port
Client(0 = [9903@localhost]) > (<< #@/ 'ls 'r)
#[objectid 0.0.0.0 0 3 (Object)root] root
  #[objectid 0.0.0.0 0 126 (Object)album-pieces] album-pieces
  #[objectid 0.0.0.0 0 119 (Object)env] env
    #[objectid 0.0.0.0 0 124 (Environment)plain] plain
  #[objectid 0.0.0.0 0 118 (Object)snapshot] snapshot
    #[objectid 192.87.7.19 4846 129 (Snapshot)1992:01:13.10:23:39.0.4848] 1992:01:13.10:23:39.0.4848
    #[objectid 0.0.0.0 0 123 (Snapshot)plain] 1970:01:01.00:00:11.11.11
    #[objectid 0.0.0.0 0 123 (Snapshot)plain] plain
  #[objectid 0.0.0.0 0 105 (Unix-Tree)unix] unix
  #[objectid 0.0.0.0 0 80 (Object)pieces] pieces
  #[objectid 0.0.0.0 0 65 (System)system] system
  #[objectid 0.0.0.0 0 2 (Class)Class] class
    #[objectid 0.0.0.0 0 125 (Class)Album-Piece] Album-Piece
    #[objectid 0.0.0.0 0 112 (Class)Environment] Environment
    #[objectid 0.0.0.0 0 106 (Class)Snapshot] Snapshot
    #[objectid 0.0.0.0 0 102 (Class)Dvi-File] Dvi-File

```

```

# [objectid 0.0.0.0 0 95 (Class)Unix-Tree] Unix-Tree
# [objectid 0.0.0.0 0 82 (Class)Oms-Src] Oms-Src
# [objectid 0.0.0.0 0 81 (Class)Scheme-Src] Scheme-Src
# [objectid 0.0.0.0 0 71 (Class)Piece] Piece
# [objectid 0.0.0.0 0 66 (Class)Value] Edit
# [objectid 0.0.0.0 0 66 (Class)Value] Value
# [objectid 0.0.0.0 0 57 (Class)System] System
# [objectid 0.0.0.0 0 34 (Class)Computed-Relation] Computed-Relation
# [objectid 0.0.0.0 0 30 (Class)Relation] Relation
# [objectid 0.0.0.0 0 27 (Class)System-Relation] System-Relation
# [objectid 0.0.0.0 0 7 (Class)Function] Function
# [objectid 0.0.0.0 0 6 (Class)Procedure] Procedure
# [objectid 0.0.0.0 0 5 (Class)Method] Method
# [objectid 0.0.0.0 0 1 (Class)Object] Object
# [objectid 0.0.0.0 0 2 (Class)Class] Class
# [objectid 0.0.0.0 0 4 (Object)relation] relation
# [objectid 0.0.0.0 0 122 (Relation)env-current-use] env-current-use
# [objectid 0.0.0.0 0 121 (Relation)env-current-snapshot] env-current-snapshot
# [objectid 0.0.0.0 0 120 (Relation)snapshot-history] snapshot-history
# [objectid 0.0.0.0 0 79 (Relation)intern-relationships] intern-relationships
# [objectid 0.0.0.0 0 78 (Relation)part-of] part-of
# [objectid 0.0.0.0 0 56 (System-Relation)system-property] system-property
# [objectid 0.0.0.0 0 48 (System-Relation)directory] directory
# [objectid 0.0.0.0 0 46 (System-Relation)dvcache-info] dvcache-info
# [objectid 0.0.0.0 0 45 (System-Relation)derived-value] derived-value
# [objectid 0.0.0.0 0 44 (System-Relation)depends-on] depends-on
# [objectid 0.0.0.0 0 43 (System-Relation)method-of] method-of
# [objectid 0.0.0.0 0 42 (System-Relation)super-class] super-class
# [objectid 0.0.0.0 0 41 (System-Relation)instance-of] instance-of
# [objectid 0.0.0.0 0 40 (System-Relation)object-info] object-info
# [objectid 0.0.0.0 0 39 (Relation)recipe-of] recipe-of

Client(0 = [9903@localhost]) > (<< #@/class 'ls 'r)
# [objectid 0.0.0.0 0 2 (Class)Class] class
# [objectid 0.0.0.0 0 125 (Class)Album-Piece] Album-Piece
# [objectid 0.0.0.0 0 112 (Class)Environment] Environment
# [objectid 0.0.0.0 0 106 (Class)Snapshot] Snapshot
# [objectid 0.0.0.0 0 102 (Class)Dvi-File] Dvi-File
# [objectid 0.0.0.0 0 95 (Class)Unix-Tree] Unix-Tree
# [objectid 0.0.0.0 0 82 (Class)Oms-Src] Oms-Src
# [objectid 0.0.0.0 0 81 (Class)Scheme-Src] Scheme-Src
# [objectid 0.0.0.0 0 71 (Class)Piece] Piece
# [objectid 0.0.0.0 0 66 (Class)Value] Edit
# [objectid 0.0.0.0 0 66 (Class)Value] Value
# [objectid 0.0.0.0 0 57 (Class)System] System
# [objectid 0.0.0.0 0 34 (Class)Computed-Relation] Computed-Relation
# [objectid 0.0.0.0 0 30 (Class)Relation] Relation
# [objectid 0.0.0.0 0 27 (Class)System-Relation] System-Relation
# [objectid 0.0.0.0 0 7 (Class)Function] Function
# [objectid 0.0.0.0 0 6 (Class)Procedure] Procedure
# [objectid 0.0.0.0 0 5 (Class)Method] Method
# [objectid 0.0.0.0 0 1 (Class)Object] Object
# [objectid 0.0.0.0 0 2 (Class)Class] Class

Client(0 = [9903@localhost]) > (<< #@/env/plain 'use 'atze)
9944
Client(0 = [9903@localhost]) > (>< "localhost" 9944)
1
Client(0 = [9903@localhost]) > (>> 1)
# [rpcport nr=0 flags=0x0 sock=9903 pid=0 host=localhost info=#[objectid 0.0.0.0 0 3]]
Client(1 = [9944@localhost]) > (<< #@/src 'ls 'r)
# [objectid 192.87.7.19 4852 4 (Object)src] src
# [objectid 192.87.7.19 4852 5 (C-Source)hello.c] hello.c

```

```

Client(1 = [9944@localhost]) > (<< #%/system 'load-scheme 'test-piece.scm)

Client(1 = [9944@localhost]) > (<< #%/pieces 'ls 'r)
#[objectid 0.0.0.0 80 (Object)pieces] pieces
  #[objectid 192.87.7.19 4856 3 (Piece)test2.pie] test2.pie
  #[objectid 192.87.7.19 4856 2 (Piece)test1.pie] test1.pie
  #[objectid 192.87.7.19 4856 1 (Piece)test.pie] test.pie

Client(1 = [9944@localhost]) > (>> 0)
#[rpcport nr=1 flags=0x0 sock=9944 pid=0 host=localhost info=#[objectid 0.0.0.0 3]]
Client(0 = [9903@localhost]) > (<< #%/env/plain 'stop 'atze)
#[objectid 192.87.7.19 4846 130]
Client(0 = [9903@localhost]) > (<< #%/snap@ 'ls 'r)
#[objectid 0.0.0.0 118 (Object)snapshot] snapshot
  #[objectid 192.87.7.19 4846 130 (Snapshot)1992:01:13.10:29:05.0.4848] 1992:01:13.10:29:05.0.4848
  #[objectid 192.87.7.19 4846 129 (Snapshot)1992:01:13.10:23:39.0.4848] 1992:01:13.10:23:39.0.4848
  #[objectid 0.0.0.0 123 (Snapshot)plain] 1970:01:01.00:00:11.11.11
  #[objectid 0.0.0.0 123 (Snapshot)plain] plain

Client(0 = [9903@localhost]) > (<< #[objectid 192.87.7.19 4846 130] 'extract-piece 'test.pie)
new#[objectid 192.87.7.19 4856 1 (Piece)test.pie]
#[objectid 192.87.7.19 4852 5 (C-Source)hello.c]
#[objectid 192.87.7.19 4852 11 (Tex-Source)hello.tex]

#[objectid 192.87.7.19 4846 131]
Client(0 = [9903@localhost]) > (<< #%/album-pi@ 'ls 'r)
#[objectid 0.0.0.0 126 (Object)album-pieces] album-pieces
  #[objectid 192.87.7.19 4846 131 (Album-Piece)test.pie] test.pie

Client(0 = [9903@localhost]) > (<< #%/snaps@plain 'transplant-piece #%/album-p@test.pie)
(object #[objectid 192.87.7.19 4856 1] ())
(relationship #[objectid 0.0.0.0 41 (System-Relation)instance-of] #[objectid 192.87.7.19 4856 1] ([objectid 192.87.7.19 4852 5] ((value main()
{
  printf( "Hello worldsssss\n" ) ;
}
))))
(object #[objectid 192.87.7.19 4852 11] ((value #[lfieldid 192.87.7.19 4847 1])))
(relationship #[objectid 0.0.0.0 48 (System-Relation)directory] #[objectid 192.87.7.19 4856 1 (Piece)] ([objectid 0.0.0.0 41 (System-Relation)instance-of] #[objectid 192.87.7.19 4852 5] ((value main()
{
  printf( "Hello worldsssss\n" ) ;
}
))))
(relationship #[objectid 0.0.0.0 78 (Relation)part-of] #[objectid 192.87.7.19 4856 1 (Piece)test.pie] ([objectid 192.87.7.19 4852 11] ((value #[lfieldid 192.87.7.19 4847 1]))))
(relationship #[objectid 0.0.0.0 41 (System-Relation)instance-of] #[objectid 192.87.7.19 4856 1 (Piece)test.pie] ([objectid 192.87.7.19 4852 11] ((value #[lfieldid 192.87.7.19 4847 1]))))

#[objectid 192.87.7.19 4846 132]
Client(0 = [9903@localhost]) > (<< #%/snapsh@ 'ls 'r)
#[objectid 0.0.0.0 118 (Object)snapshot] snapshot
  #[objectid 192.87.7.19 4846 132 (Snapshot)1992:01:13.10:33:45.0.4848] 1992:01:13.10:33:45.0.4848
  #[objectid 192.87.7.19 4846 130 (Snapshot)1992:01:13.10:29:05.0.4848] 1992:01:13.10:29:05.0.4848
  #[objectid 192.87.7.19 4846 129 (Snapshot)1992:01:13.10:23:39.0.4848] 1992:01:13.10:23:39.0.4848
  #[objectid 0.0.0.0 123 (Snapshot)plain] 1970:01:01.00:00:11.11.11
  #[objectid 0.0.0.0 123 (Snapshot)plain] plain

Client(0 = [9903@localhost]) > (<< #%/env/plain 'current)
#[objectid 192.87.7.19 4846 130]
Client(0 = [9903@localhost]) > (<< #%/env/plain 'ch-snapshot #%/snaps@1992:01:13.10:33:45.0.4848)
#[objectid 0.0.0.0 124]
Client(0 = [9903@localhost]) > (<< #%/env/plain 'current)
#[objectid 192.87.7.19 4846 132]
Client(0 = [9903@localhost]) > (<< #%/env/plain 'use 'atze)
9954
Client(0 = [9903@localhost]) > (>< "localhost" 9954)
1
Client(0 = [9903@localhost]) > (>> 1)
#[rpcport nr=0 flags=0x0 sock=9903 pid=0 host=localhost info=#[objectid 0.0.0.0 3]]

```

```

Client(1 = [9954@localhost]) > (<< #0/ 'ls 'r)
#[objectid 0.0.0.0 0 3 (Object)root] root
# [objectid 0.0.0.0 0 105 (Unix-Tree)unix] unix
# [objectid 0.0.0.0 0 80 (Object)pieces] pieces
# [objectid 192.87.7.19 4856 1 (Piece)test.pie] test.pie
# [objectid 0.0.0.0 0 65 (System)system] system
# [objectid 0.0.0.0 0 2 (Class)Class] class
# [objectid 0.0.0.0 0 102 (Class)Dvi-File] Dvi-File
# [objectid 0.0.0.0 0 95 (Class)Unix-Tree] Unix-Tree
# [objectid 0.0.0.0 0 82 (Class)Oms-Src] Oms-Src
# [objectid 0.0.0.0 0 81 (Class)Scheme-Src] Scheme-Src
# [objectid 0.0.0.0 0 71 (Class)Piece] Piece
# [objectid 0.0.0.0 0 66 (Class)Value] Edit
# [objectid 0.0.0.0 0 66 (Class)Value] Value
# [objectid 0.0.0.0 0 57 (Class)System] System
# [objectid 0.0.0.0 0 34 (Class)Computed-Relation] Computed-Relation
# [objectid 0.0.0.0 0 30 (Class)Relation] Relation
# [objectid 0.0.0.0 0 27 (Class)System-Relation] System-Relation
# [objectid 0.0.0.0 0 7 (Class)Function] Function
# [objectid 0.0.0.0 0 6 (Class)Procedure] Procedure
# [objectid 0.0.0.0 0 5 (Class)Method] Method
# [objectid 0.0.0.0 0 1 (Class)Object] Object
# [objectid 0.0.0.0 0 2 (Class)Class] Class
#[objectid 0.0.0.0 0 4 (Object)relation] relation
# [objectid 0.0.0.0 0 79 (Relation)intern-relationships] intern-relationships
# [objectid 0.0.0.0 0 78 (Relation)part-of] part-of
# [objectid 0.0.0.0 0 56 (System-Relation)system-property] system-property
# [objectid 0.0.0.0 0 48 (System-Relation)directory] directory
# [objectid 0.0.0.0 0 46 (System-Relation)dvcache-info] dvcache-info
# [objectid 0.0.0.0 0 45 (System-Relation)derived-value] derived-value
# [objectid 0.0.0.0 0 44 (System-Relation)depends-on] depends-on
# [objectid 0.0.0.0 0 43 (System-Relation)method-of] method-of
# [objectid 0.0.0.0 0 42 (System-Relation)super-class] super-class
# [objectid 0.0.0.0 0 41 (System-Relation)instance-of] instance-of
# [objectid 0.0.0.0 0 40 (System-Relation)object-info] object-info
# [objectid 0.0.0.0 0 39 (Relation)recipe-of] recipe-of

```

```

Client(1 = [9954@localhost]) > (>> 0)
#[rpcport nr=1 flags=0x0 sock=9954 pid=0 host=localhost info=#[objectid 0.0.0.0 0 3]]
Client(0 = [9903@localhost]) > (<< #0/env/plain 'stop 'atze)
#[objectid 192.87.7.19 4846 132]
Client(0 = [9903@localhost]) > (>| 0)
()
Itf> ^D
spirit[44]%

```

B Demo source 'demo.scm'

```
;;; -*-scheme*-
;;; $Header: /amd/infix/home/projects/camera/d.proto/develop/doc/RCS/demo.scm.tex,v 1.60 1993/10/08 07:58:20 atze Exp atze

;;; Code for initializing demo.

(define initialise-demo-objects
  (lambda ()
    (let
      ((class-c-source)
       (class-tex-source)
       (a-c-src)
       (a-tex-src))

      (set! class-c-source
            (send (dir-lookup-class 'Class)
                  ':construct
                  'name 'C-Source
                  'super (dir-lookup-class 'File)
                  'dfunc 'compile
                  (list nil (list type-bstring))
                  (lambda ()
                    (let ((c-src-file (string-append
                                       (new-tmp-filename)
                                       ".c"))
                          (c-exec-file (new-tmp-filename))
                          (res ""))
                      (c-exec-file (new-tmp-filename))
                      (res ""))
                    (bstring->file (send (self) 'get-value #t) c-src-file)
                    (system-do "cc"
                              c-src-file
                              "-o" c-exec-file)
                    (if (file-exists? c-exec-file)
                        (set! res (file->bstring c-exec-file)))
                    (system-rm-files c-src-file c-exec-file)
                    res))
                    "Return compilation (cc) of %var{self}."
                  'cfunc 'execute
                  (list nil (list '**))
                  (lambda ()
                    (let* ((compilation (send (self) 'compile))
                          (exec-file (new-tmp-filename))
                          (res nil))
                      (if (and (bstring? compilation)
                              (> (bstring-length compilation) 0))
                          (begin
                            (bstring->file compilation exec-file)
                            (chmod exec-file #o766)
                            (set! res (system-do->string exec-file))
                            (system-rm-files exec-file))
                          res))
                    "Execute compilation of %var{self}."
                  ))

            (send (dir-lookup '/')
                  'dir-make
                  'src)

            (set! a-c-src
                  (send class-c-source
                        'new
                        "main()\n{\n printf( \"Hello world\\n\" ) ;\n}\n"))

            (send (dir-lookup '/src)
                  'dir-add
                  'hello.c
                  a-c-src)
```

```

(set! class-tex-source
  (send (dir-lookup-class 'Class)
    'construct
    'name 'Tex-Source
    'super (dir-lookup-class 'File)
    'cfunc 'compile-with
      (list (list type-bstring) (list '(type list)))
      (lambda (aux)
        (let*
          ((tex-base (new-tmp-filename))
            (tex-src (string-append tex-base ".tex"))
            (tex-log (string-append tex-base ".log"))
            (tex-aux (string-append tex-base ".aux"))
            (tex-dvi (string-append tex-base ".dvi"))
            (res nil))
          (bstring->file (send (self) 'get-value #t) tex-src)
          (bstring->file aux tex-aux)
          (system-do "cd" sys-tmp-dir ";" "latex" tex-src)
          (set! res (list (file->bstring tex-aux)
                        (file->bstring tex-log)
                        (file->bstring tex-dvi)))
          (system-rm-files tex-src tex-log tex-aux tex-dvi)
          res))
        "Return compilation of %var{self} with %var{aux} file."
      'cfunc 'compile
        (list nil (list type-bstring))
        (lambda ()
          (let loop
            ((res nil)
             (aux ""))
            (set! res (send (self) 'compile-with aux))
            (set! res
              (if (and res
                      (not (bstring-equal? aux
                                         (first res))))
                  (loop nil
                        (first res))
                  (third res)))
            res))
          "Return compilation of %var{self}."
        'proc 'view
          (list nil (list '(type objectid)))
          (lambda ()
            (let ((dvi (send (self) 'compile))
                  (dvi-file (string-append
                            (new-tmp-filename)
                            ".dvi")))
              (bstring->file dvi dvi-file)
              (system-do "xdvi" dvi-file)
              (system-rm-files dvi-file)
              (self)))
            "Preview compilation of %var{self}."
          ))))

(send (dir-lookup '/')
  'dir-make
  'tex-src)

(set! a-tex-src
  (send class-tex-source
    'new
    "\\documentstyle{article}\n\\begin{document}\nHello World\n\\end{document}\n"))

(send (dir-lookup '/tex-src)
  'dir-add
  'hello.tex)

```

```

a-tex-src)

(let
  ((add (lambda (file)
           (let ((oid (send class-tex-source
                             'new
                             (bstring-file-add! file))))
             (send (dir-lookup
                    '/tex-src)
                    'dir-add
                    (->symbol file)
                    oid))))))
  ; (add "doc.tex")
  ; (add "doc.sty")
  #v
  )

(send (@ '/src/hello.c) 'set-derived-info 'compile nil nil nil (@ '/tex-src/hello.tex) "dummy-arg" )

(set! initialise-demo-objects nil-proc)
)))

(initialise-demo-objects)

```

C Directory structure

See demo session.

D Class hierarchy

```
#[objectid 0.0.0.0 0 1 object]
  #[objectid 131.211.80.45 19771 57 snapshot]
  #[objectid 131.211.80.45 19771 56 environment]
  #[objectid 0.0.0.0 0 52 edit]
  #[objectid 0.0.0.0 0 46 system]
  #[objectid 0.0.0.0 0 20 system-relation]
    #[objectid 0.0.0.0 0 23 relation]
      #[objectid 0.0.0.0 0 27 computed-relation]
  #[objectid 0.0.0.0 0 5 method]
    #[objectid 0.0.0.0 0 7 function]
    #[objectid 0.0.0.0 0 6 procedure]
  #[objectid 0.0.0.0 0 2 class]
```

Index

-jfile 16
-set! [27](#)

::edit [9](#), [14](#)
::edit-method [12](#)
::new-from-file [16](#)
::to-file [16](#)
:change-class [9](#)
:construct [12](#)
:def-method [12](#), [12](#), [13](#)
:def-prop [17](#)
:dump-image [17](#)
:ev [9](#)
:executable-expr [15](#)
:export-to-unix [9](#)
:function-info 11
:get-attribute [9](#)
:interface-expr [15](#)
:load-image [17](#)
:load-scheme [17](#)
:object-to-bstring [9](#), [16](#), [18](#)
:object-to-list [9](#)
:print-tex [12](#), [15](#), [18](#)
:print-tex-sub [12](#)
:print-tex-system [17](#)
:prop [17](#)
:set-attribute [9](#)
:ship-to-user [9](#)

Access 17
Album-Piece 16, 19
Algol 4
Arguments 12

C 4, 6
Camera [3](#)
Class 3, 12
Compare 11, 21
Computed-Relation 21

Dependency [3](#)
Derivation [3](#)

Edit 14, 29
Environment 7, 13

File 14
Function 3, [3](#), 20–21

HOME/.camera 6
Host [6](#)
How 21

Info 12
lftType [6](#)

Kind 11, 21

Lambda 4
LoadPath [6](#)

Machine [6](#)
Meth-class 12
Method 3, 15, 20

Name 8, 10

Object 9–10, 12
Oms 4
Oms-Src 14
OmsHeapSize [6](#)
OmsInitFile [6](#)
OmsType [6](#)

Pid [6](#)
Piece 15
Prio 11, 21
Procedure 3, [3](#), 20

Relation 20

Scheme [4](#)
Scheme-Src 14
Self 11–13
Snapshot 16
StoreDir [6](#), 7
StoreDir / "init-a.scm" 6
StoreDir / "init-b.scm" 6
Supers 13
SysDir 6, [6](#)
SysInitFile [6](#)
SysLibDir 6, [6](#)
SysLibMachDir 6, [6](#)
SysScmInterpreter [6](#)
SysTmpDir [6](#)
System 17

Trigger 11, 21

UNIX 27
Unix-Tree 19
User [6](#)

Value 29

What 21

#0 [8](#)

access 17
action 17, 25–26
actuals 15, 20
add [20](#), 23
add-function 12, 28
add-objects [15](#)

- add-procedure 12, 28
- add-recipe [21](#)
- add-relationship 15, [15](#)
- album 3, [3](#), 6–7, 25
- album-piece 17
- all 9, 13
- apropos [17](#)
- arguments 8, 13, 15
- as-lfield? 14
- as-list 10, 13, 15, 19
- attribute 3, [3](#), 4, 22
- attribute-get [22](#), 28
- attribute-names [9](#)
- attribute-set [22](#), 28

- backslash 5
- boolean 4
- bound 27
- bstring 4, 23, [23](#), 24
- bstring-¿file [24](#)
- bstring-add! [23](#)
- bstring-add-exprs! [23](#)
- bstring-content [23](#)
- bstring-content-exprs [24](#)
- bstring-del! [24](#)
- bstring-equal? [24](#)
- bstring-file-add! [23](#)
- bstring-length [24](#)
- bstring-to-object [18](#)
- bstring? [23](#)

- camera 3–4
- camerarc [5](#), 29
- car 25
- change-snapshot [13](#)
- char 4
- chmod [27](#)
- class 3, [3](#), 4, [9](#), 12
- class-name [9](#)
- classes [9](#)
- clone [9](#)
- compare 11
- construct 12
- contents->bstring [16](#)
- contents->list [16](#)
- csh 6
- current-snapshot [14](#)

- deep 13
- def-cached-function [12](#)
- def-depend-function [12](#)
- def-function [13](#)
- def-procedure 12, [13](#)
- definition-classes [15](#)
- del-recipe [21](#)
- delete [9](#), [20](#)
- delete-objects [15](#)
- delete-piece [15](#)
- delete-relationship 15, [15](#)

- depend-re-eval [24](#)
- depends-on [9](#)
- describe [10](#), [13](#), [15](#), [19](#)
- dir-add [10](#)
- dir-contents [10](#)
- dir-delete [10](#)
- dir-lookup 8, [8](#), 10, [10](#), 13
- dir-lookup-class [8](#)
- dir-lookup-rel [8](#)
- dir-make [10](#)
- dir-name [10](#)
- dir-parents [10](#)
- dir-tree-print 29
- directory 17, 20
- directory-names [27](#)
- display 9
- dump-image 17
- dvcache-info 11

- eager 3, [3](#), 24
- eighth [25](#)
- elementary value 3, [3](#)
- emacs 5
- end-of-file 24
- env 6, 25
- environment 3, [3](#), 7
- equal? 11
- errno [27](#)
- error 8, 10
- eval [15–16](#), [20](#)
- executable 15
- export [19](#)
- export-entry [19](#)
- extract-named-piece [16](#)
- extract-piece [16](#), 17

- false 8
- fields 10, 13, 19
- fifth [25](#)
- file 9, 17–18, 24
- file-¿bstring [23](#)
- file-name 16–17
- file-status 19–20, [28](#)
- fire [10](#)
- first 25, [25](#)
- for-attributes [10](#)
- for-each-object [16–17](#)
- for-each-while [25](#)
- fourth [25](#)
- from 27

- get 24
- get-edit-set [8](#)
- get-info [10](#), 11
- get-prop 18, [18](#)
- get-string-value [14](#)
- get-value [14](#)
- getenv [18](#)

- hash [27](#)
- hash-del! [27](#)
- hash-ref [27](#)
- hash-set! [26](#)
- hashtable [26](#), [26](#)
- hashtable-*l*list [26](#)
- hashtable-for-each [27](#)
- hashtable-length [26](#)
- hashtable? [26](#)

- import [19](#)
- import-entry [19](#)
- import-from-unix [18](#)
- info 11, 20
- init 17
- initialise [11](#), 12–13, [13–15](#), [17](#), 18, [19–20](#), 28
- instance-of 9
- instances [13](#)
- interface 12, 15
- interface-switch [8](#)
- intern-relationships 15–17
- itf 6, 25
- ivar 12

- keep-objects 16
- key 17–18, [26](#), 27
- kind 11, 25

- lambda 4, 27
- lazy 3, [3](#)
- length 24
- lfield 6
- lfieldid 4, 23, [23](#)
- lfields? 16–17
- link [28](#)
- lisp 4
- list 4, 25–26
- list->hashtable [26](#)
- list-head [25](#)
- list-set-car! [25](#)
- load [21](#)
- longfield 23, [23](#)
- lookup-in-snapshot [17](#)
- ls [11](#), 29

- machine-name 7
- make-hashtable [26](#)
- make-set [26](#)
- map-attributes [11](#)
- map-object [16](#)
- mask 20
- match [4](#)
- message 5
- method 3, 5, [13](#)
- method-definition [13](#)
- method-of 15, 17
- method-template [13](#)
- methods [13](#)
- mkdir [28](#)

- msg-class [23](#)

- name 8–10, 12–14, 16, 20
- new 12, [13](#), 29
- ninth [25](#)
- nr 25
- number 4

- obj 17
- object 3, [3](#), 4, 21
- object-info 17, 20
- object identifier 5
- object management system [3](#)
- objectid 4, 23
- objectids 4
- objects [16](#)
- objname 18
- oid 4–5, [5](#), 8, 21, 23
- old 23
- oms 3, [3](#), 4, 6–8, 21, 25
- oms-connect 7, [7](#), 8, 14
- oms-disconnect 7, [7](#)
- oms-handle 7–8
- oms-quit [7](#)
- on-delete-object 9
- options 11

- part-of 15
- pat 20
- path 19, 28
- path-split [28](#)
- pattern 19–21
- period 28
- piece 16–17
- piece->list [17](#)
- piece-name 16
- print? 17
- prio 11
- proc 16
- procedure 5
- proto-is [24](#)
- proto-itf-is [25](#)

- query 5, [19](#)

- read 8
- rec 19
- receive-from-unix [18](#)
- receiver 5
- recipe 21
- recurse-on 26
- regex 4–5
- relation 3, [3](#), 4, 19, 26, 28
- relationship [3](#)
- restrict 10
- result 15
- root 8, 10
- rpc 23

scheme 4–5, 7–8, 23
 scheme-types 4
 scheme-values 4
 scheme object 26
 scheme value 27
 second 25
 selector 8, 10, 12–13, 21
 self 5, 8–17, 19–22, 23
 send 8, 8, 29
 sender 23
 separator 25–26
 set-add! 26
 set-del! 27
 set-derived-info 11, 21
 set-for-each 27
 set-function-behaviour 11
 set-has?. 27
 set-info 11, 12
 set-length 26
 set-prop 17, 18
 set-value 14
 set? 17
 set? . 26
 seventh 25
 sixth 25
 size 26
 sleep 28
 snapshot 3, 3, 14
 socket 14, 23
 socket-nr 7
 start 24
 status 19–20, 20
 stop 14, 14
 storage 4, 26
 store 6
 store directory 7
 stored-query 21
 stored-value 21
 string 4, 23, 26
 string-¿strings 25
 strings-¿string 25
 sub 27
 sub-classes 13
 super 12
 super-classes 13, 13
 symbol 4–5, 12

 template 12–13, 19, 20
 terminal 25
 third 25
 timestamp 4, 17
 touch 11
 transplant-piece 17
 trigger-arg 11, 21
 true 23–24, 26
 tst 6
 tuple 3, 3, 15–16, 20
 type-list 4, 4, 12
 type-list-element 4, 5

unix-getenv 18
 unlink 28
 update 20, 20
 update-or-add 20
 use 14, 14
 user 9
 user-id 14

 val 9–10, 14
 value 5, 14, 17–18, 19, 20, 21, 22–27
 values 24
 vector 4, 26

 walk 26
 what 10–11, 16
 which 17
 wr 24

 :: 24
 << 8
 <> 7
 >> 7
 >> 8
 >| 7
 @c 8
 @r 8
 @ 8