vlab

The Virtual Laboratory

The *vlab* Framework
Reference Manual

Last updated: December 11, 2022

Contents

1 Introduction

The Virtual Laboratory (*vlab*) is an interactive environment for creating and conducting simulated experiments: a playground for experimentation. It is a framework consisting of:

- a file structure, *oofs* (object-oriented file system), for storing models (*objects*);
- a *browser* to graphically navigate through the file structure and select objects for experimentation;
- an *object manager* to move objects to a temporary location, the *lab table*, and apply *tools* (programs) to them;
- a utility, *snapicon*, to create a unique icon for each object;
- a daemon, *vlabd*, used to communicate between the browser and object manager windows; and
- a remote access server, *raserver*, providing access to remote *oofs* file structures.

The basis of the virtual laboratory is the communication between collaborating programs (tools) through the sharing of the data files comprising the object. For example, a plant may be visualized using a simulator such as *lpfg*, and manipulated by editing parameters in one or more data files. The files may be changed using a text editor, a control panel, or a specialized tool. The simulator can utilize file monitoring to update the model each time a change is made. The ability of the tools and simulators to share files and, in some cases, update the model continuously, allows the user to seamlessly experiment with an object.

The content of a virtual laboratory (the objects and tools) are domain-specific. This release of *vlab* is focused on graphical applications of L-systems, with an emphasis on the generation of fractals and the modeling of plants. The user can expand the laboratory by adding new objects and incorporating new tools.

2 The *oofs* structure

Each *vlab* object is a directory in which its associated files are found. The directory structure is hierarchical, allowing the user to create a *prototype* object, and then experiment and save a hierarchy of *extension* objects (Figure 1). *Vlab* uses this hierarchical structure to provide an inheritance mechanism: an object contains only those files that are different from the corresponding files in its parent; files that are identical are symbolics links to the parent object (Figure 2). This approach saves space, facilitates creation of objects similar to the prototype, and allows a single change in the prototype to propagate through all extensions.

2.1 Hyperlinks

It is also possible to link to an object in another part of the hierarchy. This keeps the object at its original location, but allows the user to see it as an extension of an object in a different part of the *oofs* structure as well.

When a hyperlink is created, the original object is assigned a unique object ID, which is stored with the object (in a .uuid file). This same ID is also stored with the hyperlink (in a node file). The link between an object's ID and its location is stored in a file at the root of the *oofs* structure (.uuids). When an object is moved, its ID is moved with it and its location is updated in the database. When copying, there is an option to keep the ID with the original object or move it to the copied object (see Move h-links in Section 3.4).

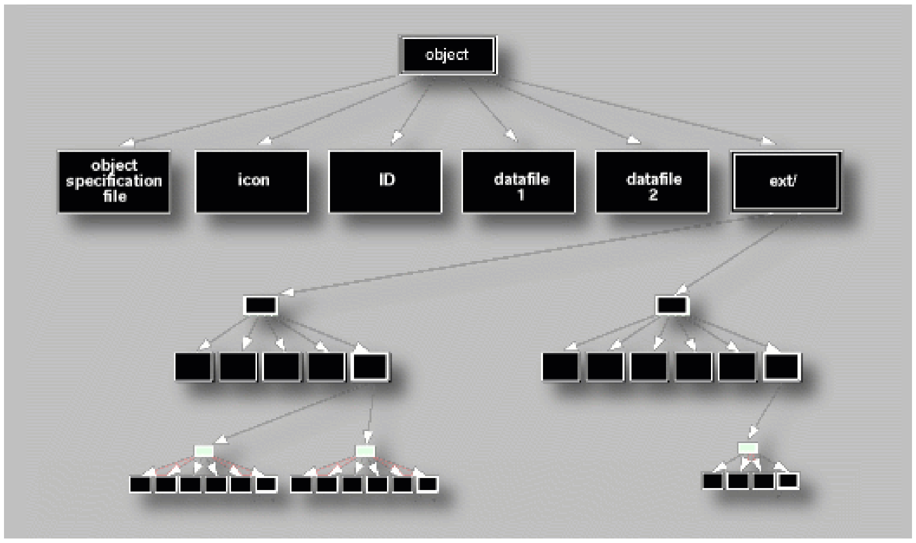


Figure 1: An example of an *oofs* hierarchy, beginning with a single object that has two extensions, one of which has two extensions of its own, and the other has a single extension.

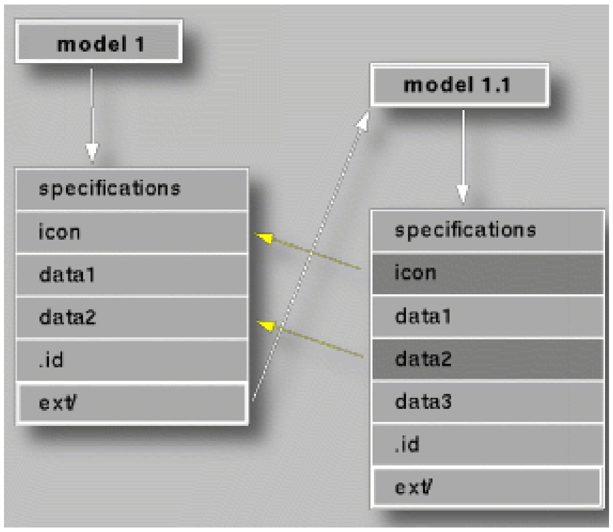


Figure 2: An example of inheritance: model 1.1 is an extension of model 1. Two files in model 1.1 are inherited from model 1: icon and data2. If changes are made to these two files in model 1, the changes will also affect model 1.1.

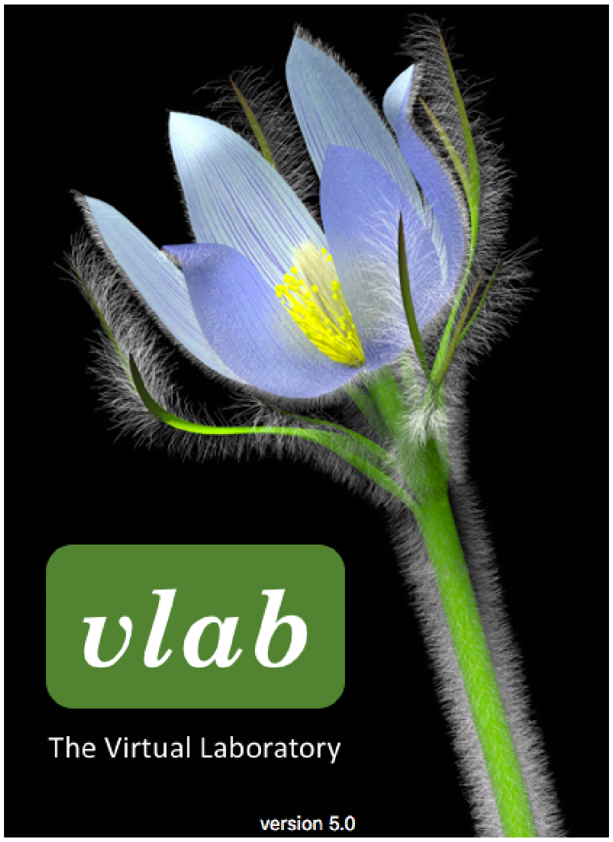


Figure 3: The splash screen for the current release of *vlab*.

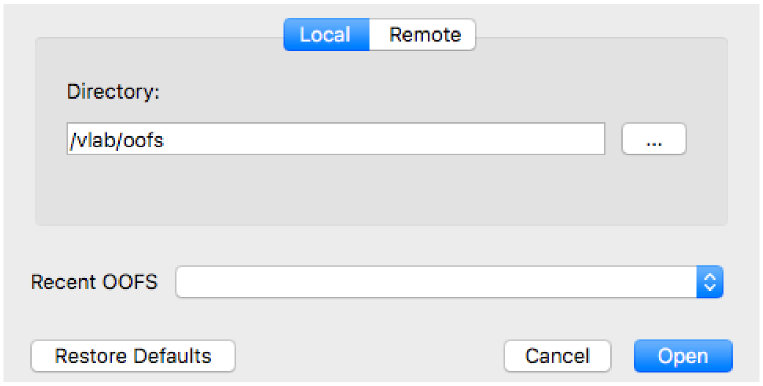


Figure 4: The initial Browser dialog box for entering the location of the *oofs* directory.

3 The Browser

When the browser application (*browser*) is first opened, it displays the splash screen (Figure 3), followed by a dialog box to define the location of the *oofs* structure (Figure 4). The initial screen is used to enter your personal ("local") *oofs* directory. See Section 6 for information on how to set up and access a shared *oofs* structure. Enter, or verify, the location of your *oofs* directory, and click the Open button to display the main *browser* window (Figure 5).

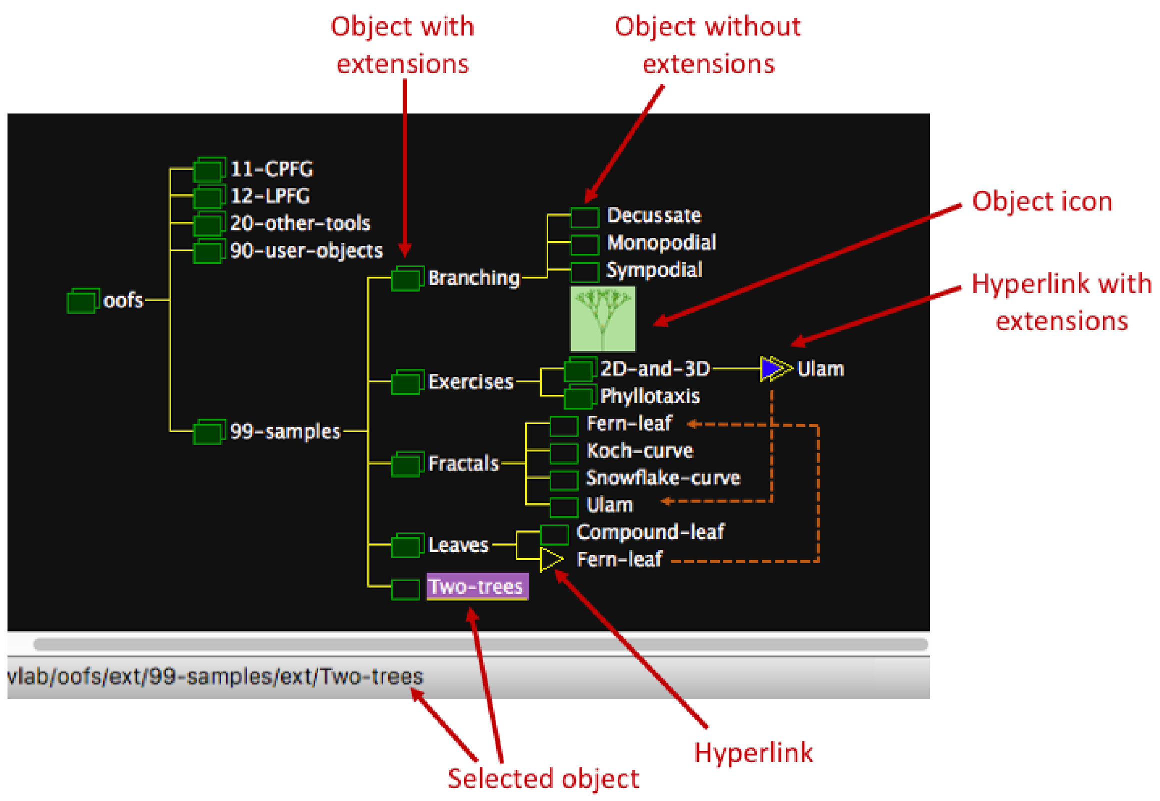


Figure 5: An example of the *browser* window. The dotted lines indicate the object linked to.

The *browser* provides a visual interface for navigating through the *oofs* hierarchy, and for manipulating objects as a whole (e.g. move, copy, delete, or rename an object). Each level of the hierarchy is represented by a browser symbol, a name, and an optional icon. The browser symbols are:

- a single box, representing an object with no extensions;
- double box, representing an object with extensions;
- an arrow, representing a hyperlink (an object linked from another part of the *oofs* hierarchy) with no extensions; and
- double arrows, representing a hyperlink with extensions.

The basic functionality of *browser*, to navigate *oofs* and select an object for experimentation, can be accomplished using the mouse. Additional functionality for maintaining the objects and the *oofs* structure uses a combination of menu items and mouse operations.

It is also possible to have multiple *browser* windows open, displaying (different parts of) the same *oofs* structure, or a different *oofs* structure altogether. Objects can be copied and pasted between *browser* windows.

3.1 Mouse operations

The following operations can be done with the mouse. In all but one case a corresponding menu item also exists.

Function	Mouse action	Menu item
Select an object	Click on the object's browser symbol, name, or icon.	
Open the Object Manager for an object	Double click on the object's browser symbol.	Object > Get
Display extensions of the object	Double click on an object's name.	View > Show extensions
Hide the entire subtree of extensions	Double click on an object's name again.	View > Hide extensions
Display/hide an object's icon	Right click on an object's name.	View > Show icon View > Hide icon
Copy a single object (not its extensions) to another location	Right click and drag the object to the location of the new parent object.	Object > Copy object

3.2 The File menu

The File menu contains general operations:

Menu item	Description
New browser	Open another *browser* window. A dialog box (Figure 4) is displayed to enter the location of the *oofs* structure.
Open shell	Open a Terminal window in the storage directory of the selected object. This is the same as the Object Manager menu item, Utilities > Shells > Storage (see Section 4.3). This function is not available when accessing a remote *oofs* database.
Open file	Open a Mac Finder window in the storage directory of the selected object. This is the same as the Object Manager menu item, Utilities > Shells > Storage (Finder) (see Section 4.3). This function is not available when accessing an *oofs* file structure using *raserver*.
Open console	Open the operating system error log. This may be useful for debugging purposes.
Import	Import an external *oofs* object or subtree as an extension of the selected object. See Export for formats.
Export	Export the selected object or subtree in either Windows or Mac/Linux format, using the same directory structure as in *oofs*, or into a .zip or .tgz file.

3.3 The View menu

Most of the items on View menu act on the currently selected object, and will be grey if no object is selected. Several of the menu items have mouse equivalents as seen in Section 3.1.

Menu item	Description
Show extensions Hide extensions	Display the extensions of the selected object / hide the entire subtree of extensions.
Show all extensions	Display the entire subtree of extensions of the selected object.
Show icon Hide icon	Display/hide the icon associated with the selected object.
Hide all icons	Hide the icons for the selected object and all its extensions.
Show all icons	Display the icons for the selected object and all its extensions.
Show hyperlink target	Locate the object associated with the currently selected hyperlink, display it in the window, and select it.

Menu item	**Description**
Begin tree here	Hide all ancestors of the selected object, making it the visible root of the tree. Only this object and its subtree can be seen. Reverse this operation using the Begin tree from root menu item.
Begin tree from root	Display the *oofs* structure beginning at its root, regardless of the currently selected object.
Enter Full Screen Exit Full Screen	Display the *browser* window in full screen mode / return to original size.

3.4 The Object menu

Functionality for organizing objects within the hierarchy is found on the Object menu. The Cut, Copy and Paste functions use a Browser-specific clipboard.

Menu item	**Description**
New object	Add a new object as an extension of the selected object. This is used to begin a new subtree that does not inherit any information from its parent.
Get	Opens the selected object on the lab table, the same as double-clicking on the object's browser symbol.
Rename	Change the name of the selected object. A dialog box will be displayed to update the current name.
Cut	Remove the selected object and all its extensions (the entire subtree) from its current location to the clipboard.
Copy object	Copy the selected object only (not its extensions) to the clipboard. This can also be done as a "drag & drop" action, using the right mouse button.
Hypercopy object	Copy the selected object's link to the clipboard.
Copy subtree	Copy the selected object and all its extensions (the entire subtree) to the clipboard.
Paste	Add the object, subtree, or link currently on the clipboard as an extension to the currently selected object.
Delete	Delete the selected object and its entire subtree from *oofs*. Note that deleting a hyperlink to an object does not delete the object itself.
Move h-links	Toggle this option on or off. When *on*, and an object is copied, the object's ID is moved to the new location. The default is *on*, so that copying an object and then deleting the original object (rather than moving it) will not break any links to it.

3.5 The Search menu

The Find item on the Search menu can be used to locate an object within the entire *oofs* structure given a part of its name. The subtree containing the object will be displayed, and the object will be selected.

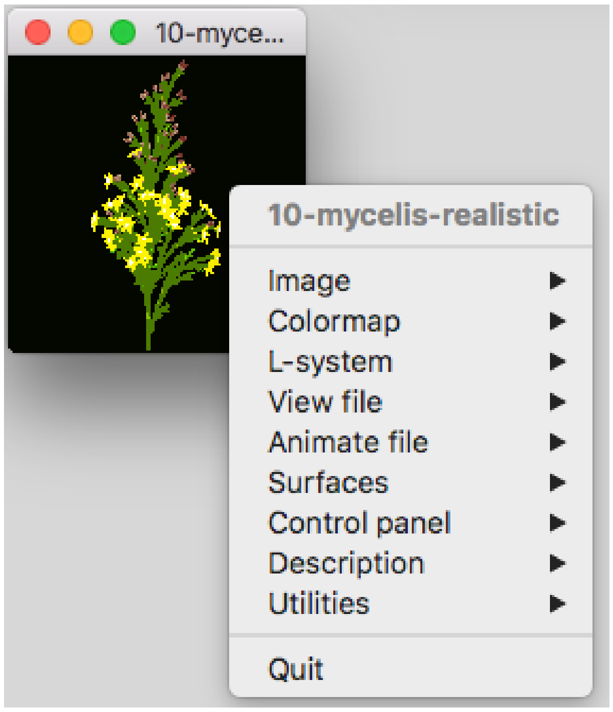


Figure 6: An example of the Object Manager. It displays an icon representing the object, and provides a menu of *tools* for manipulating it.

4 The Object Manager

The Object Manager (*object*) allows the user to experiment with an object by dynamically applying tools (programs). It is invoked from the Browser by double-clicking on the object's browser symbol in the *browser* window, or by selecting the object and using the menu item Object > Get.

4.1 Components of the Object Manager

4.1.1 The object icon

When the Object Manager is invoked, it opens a window displaying the object's icon, a small image representing the object. Right-clicking on the icon displays the menu of tools that can be used to manipulate the object (Figure 6).

4.1.2 The lab table

The Object Manager creates a copy of the object's data files on the *lab table*: a temporary directory within the user's directory structure.[1] The user can then manipulate the object on the lab table without disturbing the stored object. When done, the user may save the current state of the object on the lab table by overwriting the stored data files, or by creating an *extension* (see Section 4.3). In the latter case, only the data files that have been added or changed are explicitly stored; unchanged data files are represented as symbolic links to the parent object. If the changes were only temporary, the user may quit the Object Manager without saving the current state.

[1] Use the Lab Table shell on the Utilities menu (Section 4.3) to find the exact location of the lab table directory for an opened object. The base location of the lab table is set to /usr/.vlab/tmp when the browser is first invoked. If there is a machine crash, look here for open objects.

4.1.3 The object menus

Right-clicking on the object's icon opens its menu of tools and utilities. The tools included on the menu are defined specifically for this object using a *specifications file* (Section 4.2). Standard utilities are always included on the menu, under the Utilities menu item (Section 4.3). In addition, there is an Object Manager menu bar at the top of the screen. Object > Preferences opens the global preference file for all objects (Section 4.4). The File menu has two functions:

Menu item	**Description**
Export	Export the object in either Windows or Mac/Linux format, as a directory containing all data files (replacing symbolic links with the actual files from the parent object), or as a .zip or .tgz file. This is the same as File > Export on the *browser* menu, and can be used in conjunction with File > Import to copy an object to another *oofs* hierarchy.
Load all files	Copy ALL files from storage to the lab table, not only the ones listed in the specifications file. This is especially helpful if the object is missing its specifications file.

4.2 The specifications file

Every object must include a file called *specifications* that:

- lists the data files that constitute the object and should be moved to and from the lab table;
- lists the temporary files that should be disregarded when saving changes;
- specifies the refresh mode for the object as a whole;
- describes the object's menu hierarchy; and
- defines how each tool is applied to the object.

The *specifications* file itself is also moved to the lab table and may be edited by the user to update any of the above components (see Section 4.3).

The file begins with the list of data files to be retrieved from storage and put on the lab table, each on a separate line. This is optionally followed by two statements, and then ends with a line with a star (`*`) only. The optional statements are:

Statement	**Description**
`rmode:` *value*	Set the refresh mode for all tools that support this feature. The *value* can be: `trig` / `triggered`; `cont` / `continuous`; or `expl` / `explicit`. The default mode, if this statement is not present, is `explicit`. The refresh mode can also be set in individual tools.
`ignore:` *filename1* *filename2* ...	Ignore the listed files: do not save them to storage. The files are listed one per line below the `ignore:` statement. An entire group of files can be specified using the wildcard symbol, `*` (e.g. `*.swp` or `gal.*`).

Below the star is a hierarchy of menu items, where each level of the hierarchy is indented using a tab. The menu item name ends with a colon (`:`). Note that colons should not be used in any other context within the *specifications* file.

The lowest level defines the program (tool) to be used. It has the syntax of a MacOS/Linux command line and generally includes one or more of the data files listed above. For example, the following *specifications* file defines data files `fractal.l` through `fractal.map`, a description file, the statements `rmode` and `ignore`, and the menu hierarchy:

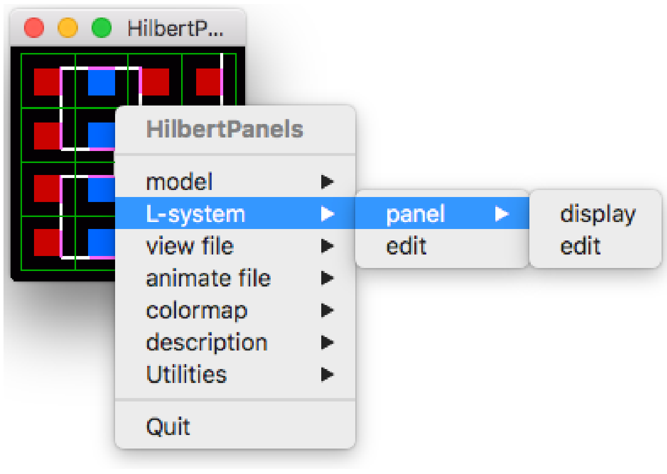
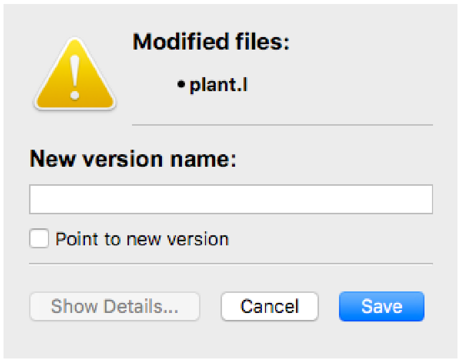


Figure 7: The menu associated with the *specifications* file example. It is displayed by right clicking on the *object* window.

```
fractal.l
fractal.v
fractal.a
panel.l
fractal.map
description.txt
rmode: cont
ignore:
*
model:
     generate:
          cpfg -m fractal.map fractal.l fractal.v fractal.a

L-system:
     panel:
          display:
               panel panel.l | awkped fractal.l
               EDIT panel.l
     EDIT fractal.l

view file:
     EDIT fractal.v

animate file:
     EDIT fractal.a

colormap:
     palette fractal.map

description:
     EDIT description.txt
```


Modified files:

• plant.l

New version name:

Point to new version

Figure 8: An example of the dialog box that is displayed when Utilities > New version is selected from the menu. The Show Details button is used when the list of modified files is too long to display in the dialog box.

Uppercase tool names indicate programs that are globally defined in the Object Manager's Preferences (Section 4.4). In the example *specifications* file above, EDIT is used to indicate the user's preferred text editor, which is globally defined in the preferences.

4.3 Utilities

The bottom entry on the main level of the object's menu, Utilities, is not defined in the specification file; it is standard for all objects. This menu consists of the following tools for managing the object:

Menu item	Next level	Description
Icon	Snap	Create a new icon for the object's window (see Section 5).
	Reread	Update the icon in the object's window.
Specifications	Edit	Open the *specifications* file in a text editor.
	Reread	Update the object's menu from the *specifications* file.
Shells	Lab table	Open a Mac Terminal window in the lab table directory.
	Lab table (Finder)	Open a Mac Finder window in the lab table directory.
	Storage	Open a Mac Terminal window in the *oofs* directory where the original object is stored. This option is not available for remote *oofs*.
	Storage (Finder)	Open a Mac Finder window in the *oofs* directory where the original object is stored. This option is not available for remote *oofs*.
Save changes**		Copy changed data files from the lab table back to the original object in storage. A dialog box will be displayed listing the files that have been modified or the message "Nothing to be saved".
New version**		Create an extension of the stored object and save the data files from the lab table to this new version. Unchanged data files are saved as symbolic links to the parent object. A dialog box will be displayed indicating the files that have been modified, and providing a field to enter the name of the new version (Figure 8). Use the Point to new version checkbox to set the object on the lab table to the new version, and select it in the *browser* window.
Position object		Select this object, and reposition the *oofs* hierarchy in the *browser* window to display it.

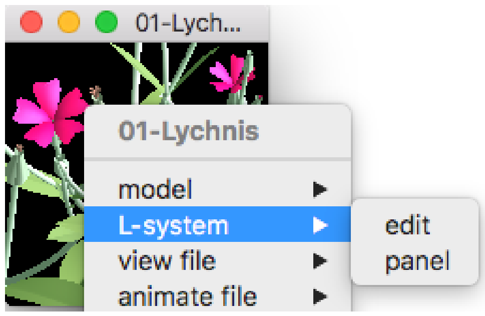


Figure 9: The edit menu item, under L-system, created with the EDIT macro in the object's preferences.

**Only files listed in the *specifications* are saved with the Save changes, New Version and Quit menu items. If new files are added to the object on the lab table, it is important to Edit the *specifications*, add the new files, and then Reread the *specifications*.

4.4 Preferences

Global object preferences can be customized by selecting Object > Preferences on the object's menu bar. This opens a text editor on the preferences file, `~/.vlab/object`. The file may include the following elements:

- define statements located at the beginning of the file;
- macros (named menu item hierarchies) for use in *specifications* files; and
- comments, in lines beginning with a semicolon (`;`), which may be placed anywhere in the file.

By convention the names of macros are in uppercase. For example, *vlab* comes with a basic text editor (see the ***vlab* Tools** manual) which can be defined and referred to as follows:

```
;Standard text editor
#define OBJED vlabTextEdit

EDIT
edit:
          OBJED
```

The `EDIT` macro can then be used as follows in the *specifications* file::

```
L-system:
          EDIT plant.l
          panel:
                    panel panel.l | awkped plant.l
```

This would define two menu items under L-system on the object's pop-up menu for editing the L-system file (Figure 9): the *vlab* text editor or a panel. The text after the `EDIT` macro is appended to the command line when it is invoked:

```
vlabTextEdit plant.l
```

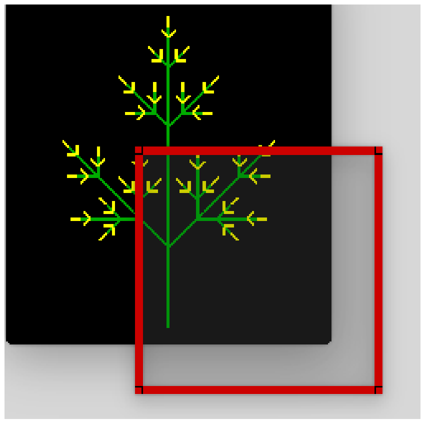


Figure 10: The *snapicon* window overlaying an object image. The window can be moved and resized to capture the image.

5 Snapicon

The *snapicon* tool captures a selected area of the screen, creating a thumbnail image which will be used to represent the object in both the *object* and *browser* windows. The image is stored in PNG format in the file `icon` on the lab table, replacing the current content of this file, if it exists.

The tool is called from the object's menu: Utilities > Icon > Snap. This opens a window outlined in red (Figure 10) that can be moved by clicking and holding the left mouse button to position it on the area of the screen to be captured.

The *snapicon* functions can be accessed from the menu bar, or by right-clicking on the window to open the pop-up menu. In both cases the following menu items are available:

Menu item	Description
Size	Select the size of the window to snap. See the options below.
Snap	Capture the content of the *snapicon* window.
Preview	Display a preview of the image captured by Snap, reduced to the size of the *object* window if a larger size was used.
Save icon	Save the image captured by Snap to the file `icon`. The image will be reduced to the size of the *object* window if a larger size was used.
Save and exit	Save the image captured by Snap to the file *icon*, and exit the program.
Snap, save and exit	Capture and save the content of the *snapicon* window to the file `icon`, and exit the program.
Save as...	Save the image captured by Snap to another file - NOT `icon`. This allows *snapicon* to capture and save images for other purposes.

The Size options are:

- Actual - Set the image size to the actual size of the *object* window.
- Double / Quadruple - Set the image size to double / quadruple the size of the *object* window.
- Aspect - Set the image size using the mouse, but keeping the same aspect as the *object* window.
- Free - Set the image size using the mouse, adjusting as required in all directions.

The final icon will be scaled to the size of the object window when it is saved, except in the case of Free sizing, which will alter the *object* window. Note that this rescaling also occurs when using Save as, unless Size = Free.

To display the new icon in the *object* window, use the menu item: Utilities > Icon > Reread.

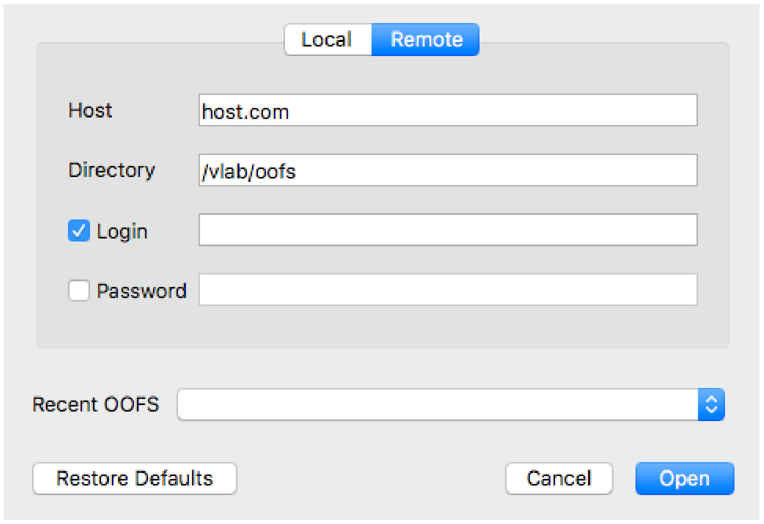


Figure 11: The dialog box for entering details for a remote access server.

6 Remote access server

When the Browser is first invoked, or when the File > New Browser menu item is selected, a dialog box is displayed to enter the location of the *oofs* structure. This may be a personal ("local") hierarchy as described in Section 3, or a shared ("remote") *oofs* structure.

Access to a shared *oofs* structure is accomplished using a remote access server, *raserver*, which runs as a daemon and performs operations on behalf of the Browser. The **owner** of the shared *oofs* structure assigns users a name and password, and defines their read or read/write access to specific subtrees (objects and their child objects) within the *oofs* structure.

The *raserver* is designed to run as a background process on any machine within a network.

WARNING: The remote access server is not secure. Only use it within a network properly secured by other means (e.g. a firewall).

The Remote button at the top of the Browser dialog box is used to display the required fields for access to the *raserver* (Figure 11).

Most operations available locally can also be performed on shared *oofs* structures, provided proper permissions have been granted. For example, the user may browse the hierarchy of objects, experiment with an object (which is transferred to the local *lab table* for fast access), and copy objects and object subtrees between browsers.

6.1 Setting up *raserver*

The owner must first set up the users who will have access to the shared *oofs* structure by running *raserver* in setup mode:

```
raserver -pe
```

A command line interface is presented with the text:

```
For a list of available commands type 'help'.
raserver>
```

The list of commands are:

Command	**Description**
`ls`	List the current users
`add` *loginname password* [*logincopy*]	Add a new user with the specified *loginname* and *password*. Optionally, also copy the permissions from user, *logincopy*, to this new user.
`del` *loginname*	Delete the user named *loginname*
`chlog` *oldname newname*	Change a user's login name from *oldname* to *newname*
`chpass` *loginname newpassword*	Change a user's password to *newpassword*
`edit` *loginname*	Edit the directory permissions for *loginname*. See below.
`quit`	Exit the program
`help`	List these commands

The `edit` command is used to set up access within the *oofs* structure. When it is invoked, the following set of commands can be used to update the specified user:

Command	**Description**
`ls`	List the current permission rules for this user
`add` *permission path*	Add a new rule for this user, where *permission* can be one of the values listed below, and *path* is the full file structure pathname to the required node in the *oofs* hierarchy.
`del` *rnum*	Delete rule number *rnum* for this user
`chmod` *permission rnum*	Change the permissions associated with rule number *rnum* to one of the values listed below.
`quit`	Exit the permissions editor, and return to the main *raserver* prompt.
`help`	List these commands

The *permission* assigned in the `add` and `chmod` rules may be specified using a character or number:

Character	**Number**	**Description**
`-`	`0`	No permission
`r`	`4`	Read access only
`w`	`2`	Write access only
`rw`	`6`	Read and write access

Permissions are granted within the actual file structure, and must contain the complete path. For example, to grant read access to the entire *oofs* structure, but read/write access only to a subtree beginning with an object three levels down, two rules would be used:

```
add r /vlab/oofs
add rw /vlab/oofs/ext/level1/ext/level2/ext/level3
```

and would be displayed (using the `ls` command) as:

```
##  I  R/W   Path
-----------------------------------------------------------------------------
0      R/-   /vlab/oofs
1      R/W   /vlab/oofs/ext/level1/ext/level2/ext/level3
```

Note that rules are ordered by a number, but the number is not assigned specifically to a rule. For example, if three rules are created they will be numbered 0, 1, and 2. However, if the middle rule (1) is deleted, the remaining rules will be numbered 0 and 1 (i.e. rule 2 will become rule 1).

6.2 Daemon mode

The owner provides access to the shared *oofs* structure, using the permissions set up above, by running the remote access server in daemon mode. This is done by simply running the command `raserver`. Follow it with an ampersand (`&`) to run it in the background:

```
raserver &
```

The server will respond with:

```
raserver:  running
```

and will listen for communications from Browser programs.

7 Credits

Precursor ideas were introduced by Przemyslaw Prusinkiewicz and Jim Hanan in [1]. The original version of *vlab* was designed and implemented by Lynn Mercer [2; 3].

Hyperobjects were introduced by Pavol Federl. The graphical interface for the browser was prototyped using Tcl/Tk by Earl Lowe [4], implemented using Motif and OpenGL by Pavol Federl [5; 6], and ported to Qt by Jin Xiao. The *vlab* daemon was designed and implemented by Earle Lowe [4], and enhanced by Pavol Federl. The remote access server was designed and implemented by Pavol Federl and extended by Pierre Barbier de Reuille and Pascal Ferraro. File monitoring and refresh modes were prototyped by Cordell Bloor; the current implementation is by Pascal Ferraro. The *snapicon* tool was designed and implemented by Jin Xiao and Pascal Ferraro. The *vlab* splash screen art is by Martin Fuhrer [7].

All components of the *vlab* framework have been enhanced and maintained by Pascal Ferraro.

8 Document revision history

Date	Description	By
1996	The first version of this documentation in HTML.	Istvan Hernadi
1997 - 2020	Updates made to the HTML documentation.	Pavol Federl Pascal Ferraro
2022	Rewritten to include all features of the current implementation, and reformatted in LaTex.	Lynn Mercer

References

[1] Przemyslaw Prusinkiewicz and Jim Hanan. A hypertext environment for unix. In *Proceedings of Graphics Interface '88*, pages 50–55, 1988.

[2] Lynn Mercer, Przemyslaw Prusinkiewicz, and Jim Hanan. The concept and design of a virtual laboratory. *Proceedings of Graphics Interface '90*, pages 149–155, 1990.

[3] Lynn Mercer. The virtual laboratory. Master's thesis, University of Regina, 1991.

[4] Earl Lowe. Extensions to the virtual laboratory. Master's thesis, University of Calgary, 1995.

[5] Pavol Federl. Design and implementation of a global virtual laboratory - a network-accessibile simulation environment. Master's thesis, University of Calgary, 1997.

[6] Pavol Federl and Przemyslaw Prusinkiewicz. Virtual laboratory: an interative software environment for computer graphics. *Proceedings of Computer Graphics International*, pages 93–100, 1999.

[7] Martin Fuhrer, Henrik Wann Jensen, and Przemyslaw Prusinkiewicz. Modeling hairy plants. In *12th Pacific Conference on Computer Graphics and Applications*, pages 217–226, 2004.