ViTables Users' Guide Documentation

Release 2.1

Vicent Mas

February 15, 2011

CONTENTS

Vicent Mas

ViTables 2.1 Users' Guide version 1.0

February 15, 2011

CHAPTER

ONE

INTRODUCTION

1.1 Overview

ViTables is a member of the PyTables family. It's a graphical tool for browsing and editing files in both PyTables and HDF5 formats. With *ViTables* you can easily navigate through data hierarchies, request metadata, view real data and much more.

ViTables is being developed using Python and PyQt, the bindings of Qt libraries, so it can run on any platform that supports these components (which includes Windows, Mac OS X, Linux and many other Unices). The interface and features will remain the same on all platforms.

Efficiency and low memory requirements are guaranteed by the fact that data is loaded only when the object that contains it is opened and by the use of data buffers for dealing with large datasets.

1.2 Capabilities

The current release provides browsing, displaying, editing and querying capabilities. Some of them are listed below. Details are discussed in the related chapters.

1.2.1 Browsing and Displaying Capabilities

- Display data hierarchy as a fully browsable object tree.
- Open several files simultaneously.
- Open files in write mode as well as in read-only mode, disabling all editing functions.
- Display file information (path, size, number of nodes…).
- Display node (group or leaf) properties, including metadata and attributes.
- Display numerical arrays, i.e. homogeneous tables.
- Display heterogeneous table entities, i.e. records.
- Display multidimensional table cells.
- Unlimited zoom into the inner dimensions of multidimensional table cells.

1.2.2 Editing Capabilities

These editing features have been implemented for the object tree [1]:

- File creation and renaming.

[1] Dataset editing capabilities have not yet been implemented.

- Node creation (only for groups), renaming and deletion.
- Ability to copy and move nodes from their location to a different one, even in different files.
- Attribute creation, renaming and deletion.

All these changes automatically update the database (i.e. the file) to which the nodes belong.

1.2.3 Other

Other nice features include:

- *Ability to smoothly navigate really large datasets*.
- Support for doing complex table queries with a low memory footprint.
- Flexible plugins framework. A bunch of useful plugins are already included, see the *Appendix A* for details.
- Configurable look and feel.
- A logger area, where errors and warnings (if any) are printed.
- Several levels of help are available: users guide, context help, tooltips and status bar.

We have paid special attention to usability issues so making use of these features is intuitive and pleasant. Nevertheless, and just in case, we are providing this guide :-).

1.3 System Requirements

To run *ViTables* you need to install Python 2.6 or Python 2.7, PyTables >= 2.2 (so you have to fulfil its own requirements) and PyQt4 >= 4.8.3.

At the moment, *ViTables* has been fully tested on Linux and Windows XP and Vista platforms. Other Unices should run just fine when using the Linux version because all the software that *ViTables* relies on (i.e. Python, Qt, PyQt, HDF5 and PyTables) is known to run fine on many Unix platforms as well.

1.4 Installation

1.4.1 Unix

The Distutils module (part of the standard Python distribution) has been used to prepare an installer for *ViTables*. It makes easy to get the application up and running.

At the moment no binary versions of the installer exist, so the software has to be installed from sources.

Provided that your system fulfills the requirements listed in the above sections, installing the package is really easy. Just uncompress the package, change to the distribution directory and execute

```
$ python setup.py install
```

By default *ViTables* will be installed in the system-protected area where your system installs third party Python packages, so you will need superuser privileges. If you prefer to install the package in a different location (for instance, your home directory, so that you can complete the installation as a non-privileged user), you can do it using the –prefix tag:

```
$ python setup.py install --prefix=/home/myuser/mystuff
```

Please remember that installing Python modules in non-standard locations makes it necessary to set the `PYTHONPATH` environment variable properly so that the Python interpreter can find the installed modules.

If you need further customizations, please have a look at the output of the command

```
$python setup.py install --help
```

to see the available options. Complete information about these options can be found in the Distutils documentation.

1.4.2 Windows Binary Installers

A binary installer is available for Windows platforms. Just double click the installer icon and follow the wizard instructions. *ViTables* will be installed in a few clicks.

Beware that the installer is not a superpackage containing all *ViTables* requirements. You need PyTables and PyQt4 already installed on your system (excellent installers for both packages are available) in order to install *ViTables*.

1.4.3 Mac OS X Binary Installers

You can use the general Unix procedure to install *ViTables* on Mac OS X, but generating a double-clickable application bundle is recommended. Simply untar the source package, change to the distribution directory and execute

```
$ cd macosxapp
$ ./make.sh
```

If you have problems with this please, refer to the FAQ page in the *ViTables* website.

1.4.4 Further Reading

General information about PyTables can be found at the project site <www.pytables.com>. For more information on HDF5, please visit its web site <www.hdfgroup.org/HDF5>. Information about *ViTables* is available at <www.vitables.org>.

Questions and feedback can be mailed to the developers.

CHAPTER

TWO

FIRST STEPS

In this chapter we are going to describe briefly the main elements that you will meet throughout your working sessions.

2.1 How to Start

Usually *ViTables* is started by running the **vitables** program under X (from a terminal emulator or directly from your desktop). If you are using a terminal then you can give some arguments to the command line

Currently command line arguments are available only in Linux platforms. You can get the available arguments by issuing the command:

```
$ vitables --help
$ usage: vitables \[options] \[h5file]
options:
--version              show program's version number and exit
-h, --help             show this help message and exit
-mMODE, --mode=MODE    mode access for a database
-dh5list, --dblist=h5list
a file with the list of databases to be open
```

Basically you can specify a file to open or a file containing a list of files to open. For example:

```
$ vitables myh5file
```

will start *ViTables* and open the file myh5file in read-write mode. If you want to open it in read-only mode then execute the command:

```
$ vitables -m r myh5file
```

In order to open a set of files at once put them in a list file with the syntax

```
mode path
```

(one pair per line) and execute:

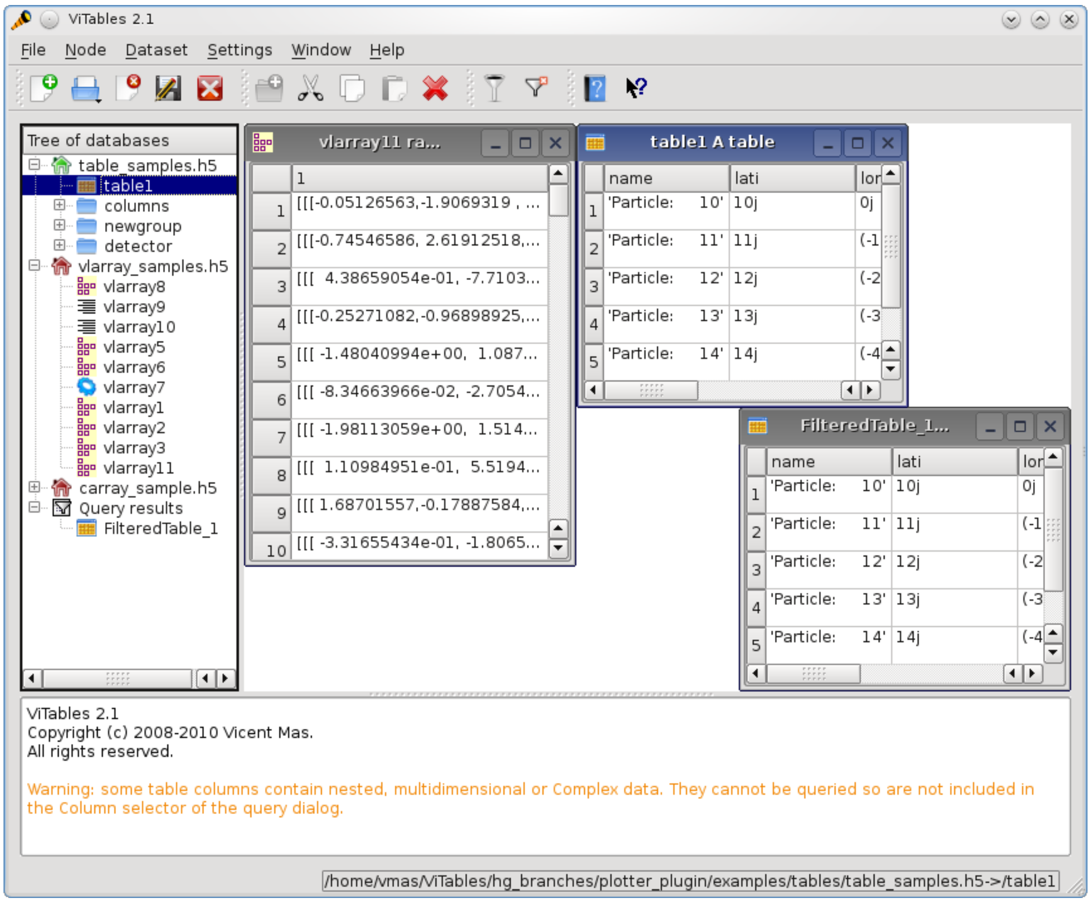
```
$ vitables -d h5list
```

Once the application is running the *main window* appears. It consists of a *menu bar*, a set of tool bars, a *viewing area* and a status bar.

The viewing area of the window is divided into three parts. The *tree of databases viewer* is the narrow region placed at top left side. It will display a tree representation of the data hierarchies we want to access. The big panel next to the tree viewer is called the *workspace*, and will display the real data contained in a given node of the data hierarchy. Finally, the bottom region is the *logger*, a kind of text non interactive console where information about your requested operations will be shown.

As usual, you can launch commands from the menu bar, from context menus or, if a shortcut button is available, from a toolbar. Also keyboard shortcuts are available for most commands.

After starting your session, you are likely to open some files. Just drag the file(s) you want to open into the tree of databases viewer and they will be opened in read-write mode. Opening can be done from the file manager dialog too; simply issue an open command, *File* → *Open File* (`Ctrl-O`) and choose a file.



	1
1	[[[-0.05126563,-1.9069319 , ...
2	[[[-0.74546586, 2.61912518,...
3	[[[4.38659054e-01, -7.7103...
4	[[[-0.25271082,-0.96898925,...
5	[[[-1.48040994e+00, 1.087...
6	[[[-8.34663966e-02, -2.7054...
7	[[[-1.98113059e+00, 1.514...
8	[[[1.10984951e-01, 5.5194...
9	[[[1.68701557,-0.17887584,...
10	[[[-3.31655434e-01, -1.8065...

	name	lati	lor
1	'Particle: 10'	10j	0j
2	'Particle: 11'	11j	(-1
3	'Particle: 12'	12j	(-2
4	'Particle: 13'	13j	(-3
5	'Particle: 14'	14j	(-4

	name	lati	lor
1	'Particle: 10'	10j	0j
2	'Particle: 11'	11j	(-1
3	'Particle: 12'	12j	(-2
4	'Particle: 13'	13j	(-3
5	'Particle: 14'	14j	(-4

ViTables 2.1
Copyright (c) 2008-2010 Vicent Mas.
All rights reserved.

Warning: some table columns contain nested, multidimensional or Complex data. They cannot be queried so are not included in the Column selector of the query dialog.

/home/vmas/ViTables/hg_branches/plotter_plugin/examples/tables/table_samples.h5->/table1

Figure 2.1: The main window

2.2 The Menu Bar

The menu bar is placed at top of the main window. It is composed of six pulldown menus.

File menu

This menu contains commands to manipulate files in several ways: open, close, create, save and so on. It also gives to you quick access to the most recently opened files.

Node menu

The *Node* menu contains commands to manipulate the nodes displayed in the tree of databases viewer. From this menu you can edit nodes in a variety of ways as well as access their properties.

Dataset menu

With this menu you can make selects in any table (the result of your selects will be available under the Query Results node in the tree pane). The number of entries for this menu depends on the list of enabled plugins.

Settings menu

This is the menu from which the application can be customized. Customization includes startup behavior, look and feel and plugins management. You can add/remove paths for loading plugins, enable and disable plugins. Changes in the enabled/disabled status of a given plugin take effect after restarting *ViTables*. See *the ViTables Configuration chapter* for more information on this subject.

Also from this menu you can show, hide and line up the application toolbars. At the moment four toolbars are available, *File*, *Node*, *Query* and *Help*.

Window menu

The *Window* menu can be used to change the arrangement of the workspace contents, sorting the open windows as a cascade or as a tile. By selecting a window name from this menu, you can raise (bring to the front) that window. Any open window can be closed from this menu.

Help menu

The *Help* menu displays this User's Guide in HTML4 format and a couple of *About* boxes, one for the *ViTables* itself and one for the underlying Qt libraries. The *Show Versions* entry shows the version numbers of the libraries being used by *ViTables* (Qt, PyQt, PyTables and PyTables related libraries, like Zlib or LZO). Finally, from this menu you can enter the *What's This* mode which will show context help for the components of the viewing area (the databases tree viewer, the workspace and the logger).

2.3 The Viewing Area

As mentioned before, the viewing area is divided into three regions: the databases tree viewer (also called tree pane), the workspace and the logger. Now we are going to describe these regions in more detail.

2.3.1 The Databases Tree Viewer

Due to the hierarchical model of the underlying HDF5 library, PyTables files store their data on disk in a tree-like structure. Every time you open a PyTables file, its so-called object tree (a representation of the data hierarchy) is dynamically created and added to the tree of databases viewer, at the top left side of the viewing area (see *the main window Figure*).

Note: since PyTables-1.2 the object tree of an opened file is made on demand: nodes are added to the tree when they are accessed. *ViTables* makes use of this feature, which results in stunningly fast opening times for files with a large number of nodes.

Any object tree is made of nodes which can be classified as follows:

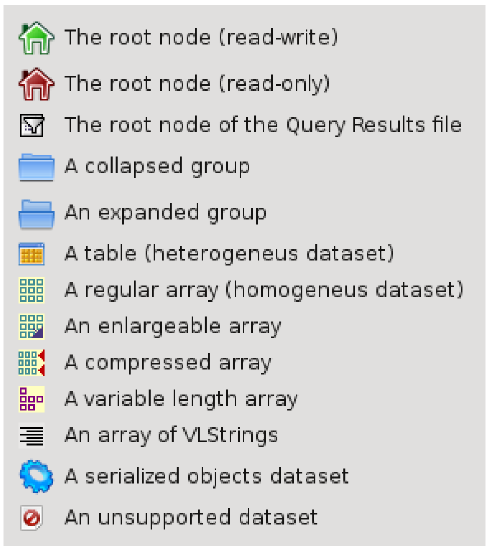
Root node It is the node from which all other nodes hang.

Groups Groups are nodes that can contain other nodes.

Leaves Leaves are nodes that contain real data. They can be tables or arrays.

Working with object trees is really easy. By double-clicking on it, a root node is opened, and the tree structure below it is displayed. Groups are presented as folders. They can be expanded with a double-click, giving you immediate access to their contents. A group can contain groups and/or leaves (or may be empty). A double click on a leaf will display its content on the workspace. You can access the available options for a given node just with a right mouse click on it. A context menu will appear from which commands can be launched. The contents of the menu depend on the kind of node being clicked (root nodes, groups, tables and arrays have all of them their own context menu). Alternatively you can select the node with a single mouse click and choose a command from the *Node* menu. There is also a context menu for the tree pane itself that will pop up by right-clicking any empty area of the tree viewer. Last but not least, the object tree can be navigated with the keyboard too. Pressing the Enter key the selected node will be expanded (if it is a group) or opened (if it is a leaf). The + and - keys expand an collapse groups.

Every node in a given object tree has an associated icon that allows you to identify its type quickly. The following icons are available:



The root node (read-write)
The root node (read-only)
The root node of the Query Results file
A collapsed group
An expanded group
A table (heterogeneus dataset)
A regular array (homogeneus dataset)
An enlargeable array
A compressed array
A variable length array
An array of VLStrings
A serialized objects dataset
An unsupported dataset

Figure 2.2: The node symbols

2.3.2 The Workspace

At this point you should have one or more files opened, and their object trees displayed in the databases tree viewer. Your next step will be to select a leaf and display its data. Remember that the object tree imitates the structure on disk, which makes it very easy to browse the hierarchy of the file and locate the leaf you want to open.

A double-click on a leaf of your choice will open it and display its contents in a window (a *view* in the *ViTables* jargon) placed in the workspace, the big panel at the top right side of the viewing area (see *the main window Figure*).

Note that the databases tree viewer and the workspace are always synchronized: if you select a node in the tree viewer and that node has a view, then that view becomes the active view on the workspace. The opposite is also true, click on a view on the workspace and its node will be automatically selected on the databases tree viewer.

The *Window* pulldown menu provides some additional commands that will help you to manage your views. From this menu you can, for instance, rearrange views, see the list of views (which is particularly useful when the workspace is cluttered with so many views that it's difficult to find the one you want) or close all the views at once.

There is also a context menu for the workspace. It can be used to change the workspace view mode: you can display views as regular windows (default behavior) or with tabs in a tab bar. In addition it give you access to the *Window* pulldown menu.

2.3.3 The Logger

The logger is a read-only (i.e. non interactive) console placed at the bottom of the viewing area (see *the main window Figure*). It is an info panel where *ViTables* reports the result of requested operations (namely if they were not successful). Also runtime errors are caught and reported to you through the logger (so you can mail the error to *ViTables* developers and help to improve the quality of the package :-). Errors and warning messages are highlighted in red and orange respectively.

Of course there is also a context menu for the logger that provides you with some handy operations, like to copy selected text or to empty the logger.

CHAPTER

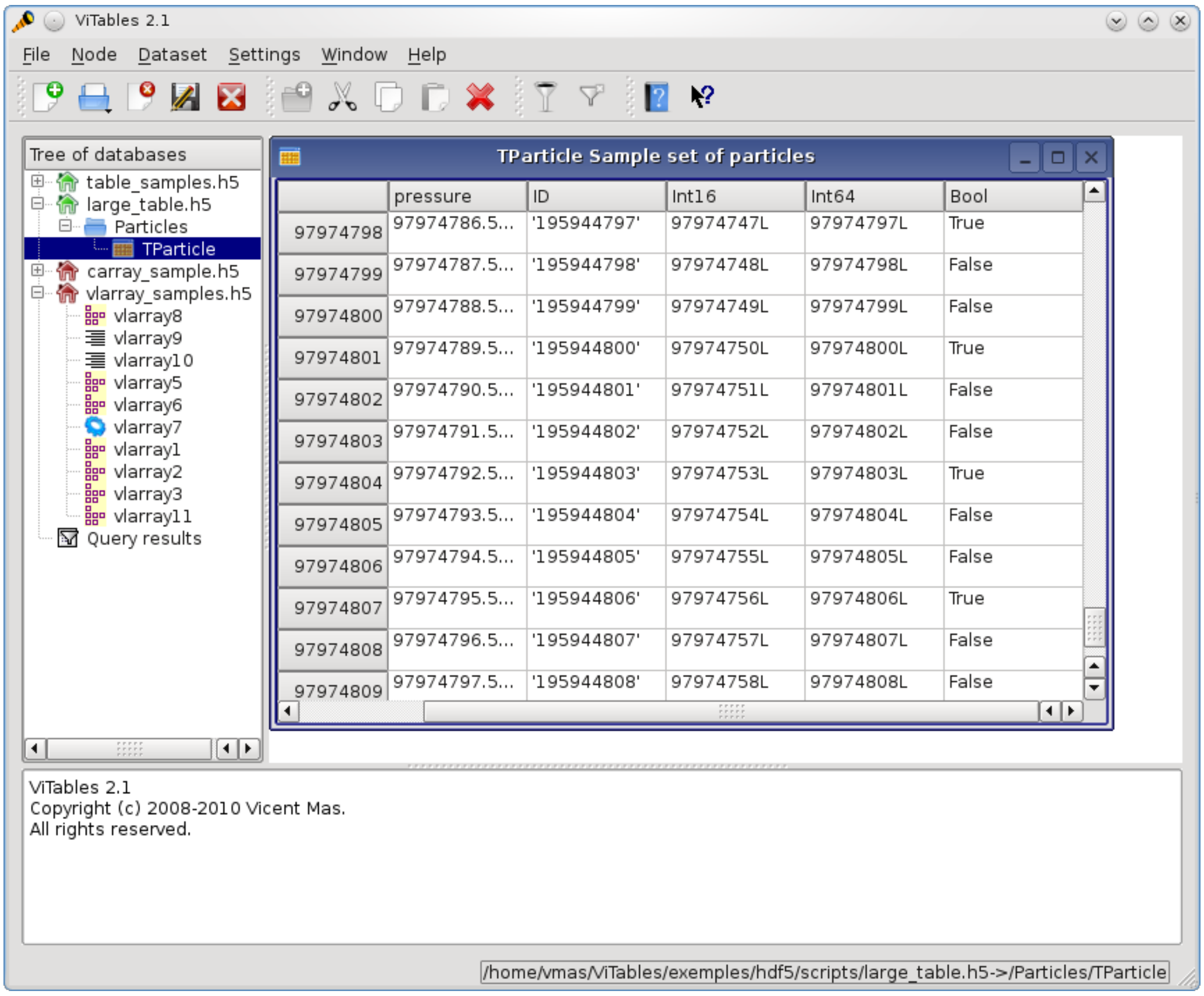
THREE

BROWSING AND QUERYING DATASETS

In this chapter we are going to describe how the information contained in a dataset can be navigated and filtered.

3.1 Browsing Datasets

A noticeable aspect of views is the visualization speed. Views show data nearly as quickly as PyTables accesses them. As a consequence, very large datasets (with up to 2^64 rows) can be browsed stunningly fast.



	pressure	ID	Int16	Int64	Bool
97974798	97974786.5...	'195944797'	97974747L	97974797L	True
97974799	97974787.5...	'195944798'	97974748L	97974798L	False
97974800	97974788.5...	'195944799'	97974749L	97974799L	False
97974801	97974789.5...	'195944800'	97974750L	97974800L	True
97974802	97974790.5...	'195944801'	97974751L	97974801L	False
97974803	97974791.5...	'195944802'	97974752L	97974802L	False
97974804	97974792.5...	'195944803'	97974753L	97974803L	True
97974805	97974793.5...	'195944804'	97974754L	97974804L	False
97974806	97974794.5...	'195944805'	97974755L	97974805L	False
97974807	97974795.5...	'195944806'	97974756L	97974806L	True
97974808	97974796.5...	'195944807'	97974757L	97974807L	False
97974809	97974797.5...	'195944808'	97974758L	97974808L	False

```
ViTables 2.1
Copyright (c) 2008-2010 Vicent Mas.
All rights reserved.
```

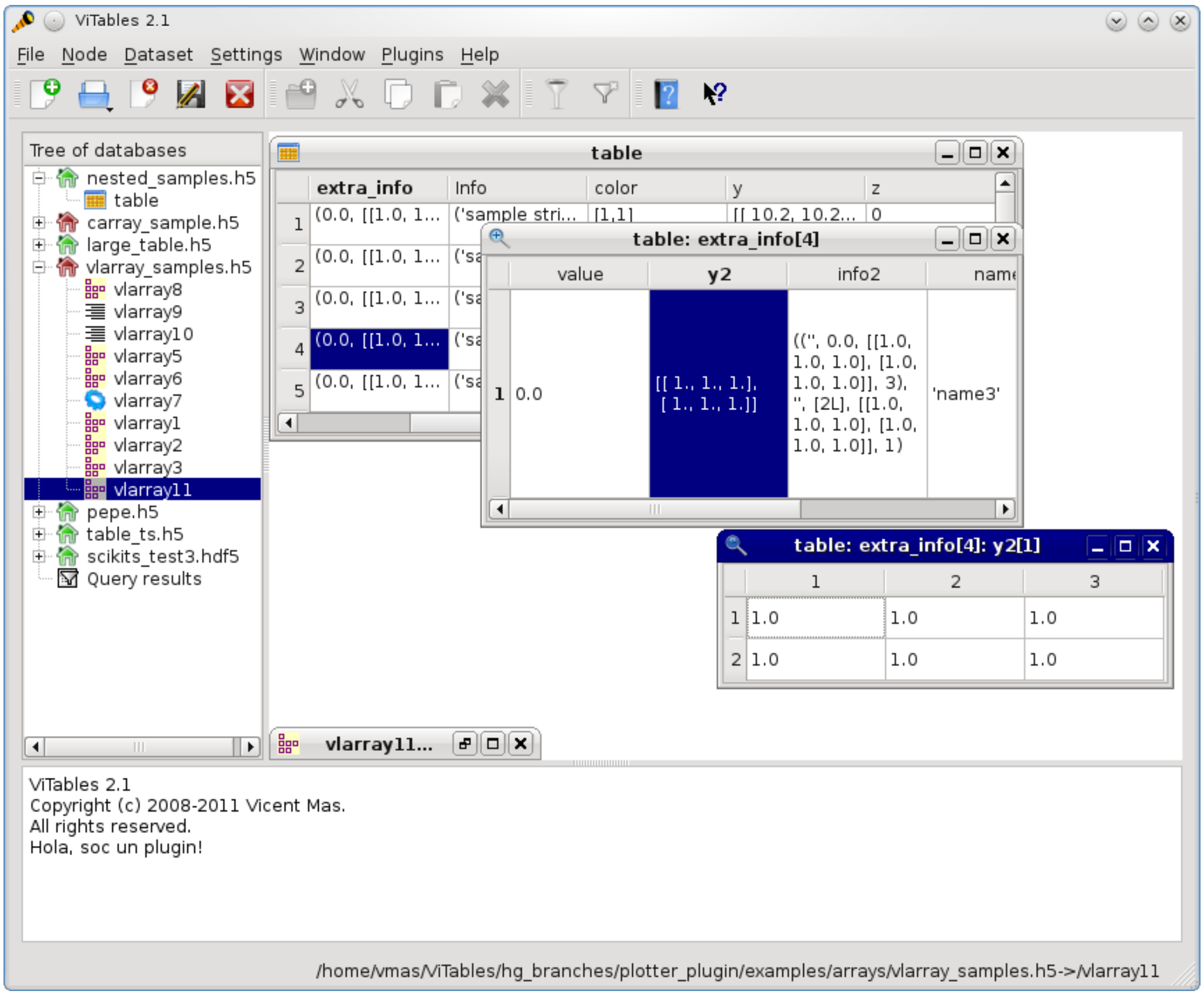
/home/vmas/ViTables/exemples/hdf5/scripts/large_table.h5->/Particles/TParticle

Figure 3.1: Browsing a large dataset

As said in the previous chapter, datasets are displayed in views. They are spreadsheet-like windows so can be navigated as you would expect: via scrollbar, keyboard or wheel of mouse.

Note: what makes *ViTables* views interesting is that *the navigation speed is independent of the view size*: a table with several thousand million rows is navigated as quickly as a table with just a few dozens rows.

Another interesting feature of views is the ability to zoom in on cells that contain multidimensional data. When you double-click in a cell, it is displayed in its own view, reducing by two the number of dimensions of the displayed data. For instance, a cell containing a vector is displayed as a one column table of scalars. A cell that contains two-dimensional data will be shown as a bidimensional table of scalars. And so on. In general, a cell containing N-dimensional data will be displayed as a table of N-2 dimensions data. Zoom can be applied as many times as needed, so that multi-dimensional cells can be inspected until you get a table of scalars. Finally, if you try to zoom in on a cell that contains a scalar value, this value will be presented alone in a view; this can be useful to visualize large scalar values (for example, large strings) that doesn't fit on regular columns.



	extra_info	Info	color	y	z
1	(0.0, [[1.0, 1...	('sample stri...	[1,1]	[[10.2, 10.2...	0
2	(0.0, [[1.0, 1...	('sa			
3	(0.0, [[1.0, 1...	('sa			
4	(0.0, [[1.0, 1...	('sa			
5	(0.0, [[1.0, 1...	('sa			

	value	y2	info2	nam
1	0.0	[[1., 1., 1.], [1., 1., 1.]]	(('', 0.0, [[1.0, 1.0, 1.0], [1.0, 1.0, 1.0]], 3), '', [2L], [[1.0, 1.0, 1.0], [1.0, 1.0, 1.0]], 1)	'name3'

	1	2	3
1	1.0	1.0	1.0
2	1.0	1.0	1.0

```
ViTables 2.1
Copyright (c) 2008-2011 Vicent Mas.
All rights reserved.
Hola, soc un plugin!
```

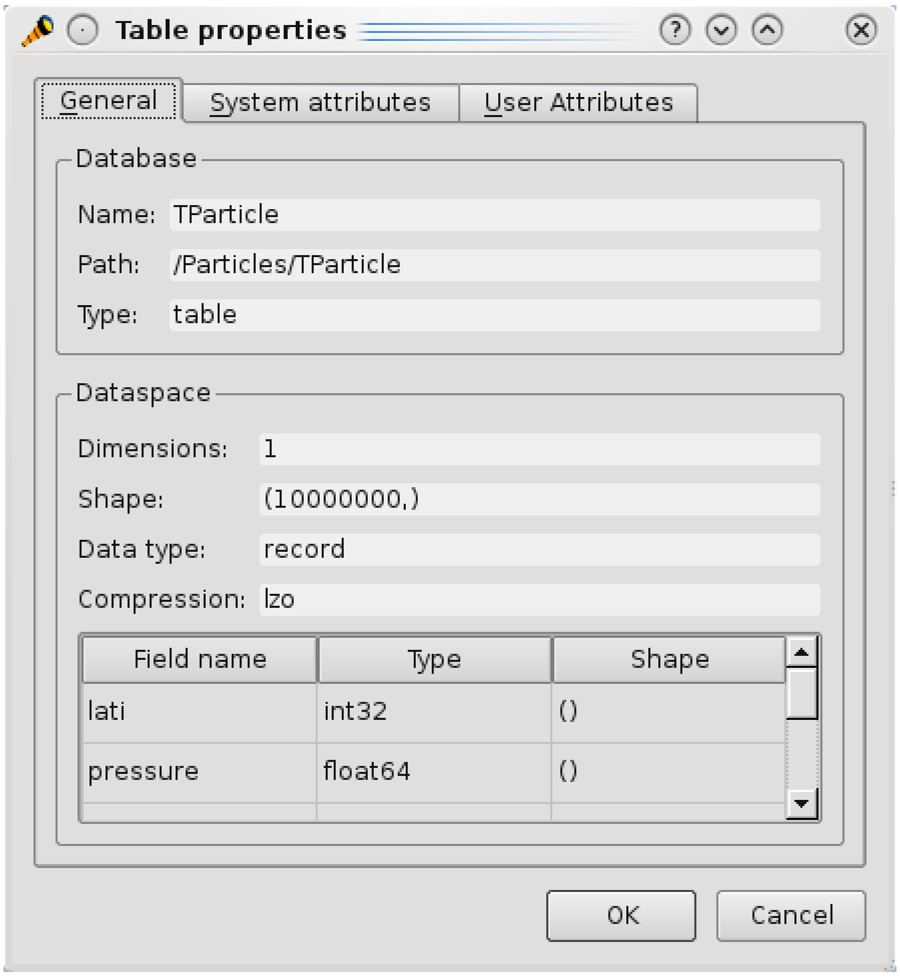
/home/vmas/ViTables/hg_branches/plotter_plugin/examples/arrays/vlarray_samples.h5->/vlarray11

Figure 3.2: Zooming a cell

3.2 Getting Info

For a given node two kinds of information are available: metadata and data. From their metadata you can retrieve information about the objects on disk, such as table and column names, titles, number of rows, data types in columns or attributes, among others.

The available metadata about a given node (group or leaf) can be accessed by right-clicking the mouse on the node and launching the *Properties* command from the context menu that appears. This can also be achieved from the *Node* menu. Then the *Properties dialog*, that contains the requested metadata, is displayed. The dialog is made of three tabs labelled as General, System attributes and User attributes. The General tab contains generic information about the selected node, i.e. name, path, etc. The System and User tabs contain tables that describe the attributes of the node.



Name:	TParticle
Path:	/Particles/TParticle
Type:	table
Dimensions:	1
Shape:	(10000000,)
Data type:	record
Compression:	lzo

Field name	Type	Shape
lati	int32	()
pressure	float64	()

Figure 3.3: The Properties dialog

Aside from the Properties dialog, you can get information in several other ways.

The full path of the node currently selected in the tree view is displayed in the status bar. This can be useful when the object tree is large and guessing a full path is not easy.

The top left icon of views shows the kind of displayed data (array or table).

Finally, some generic information can be obtained by launching the command *Help* → *WhatIsThis* (or clicking the appropriate button on the corresponding toolbar).

3.3 Querying Tables

An interesting feature of *ViTables* is its capability to make table selections. This means that we can select a set of table rows that fulfill a given condition. You can filter any table (even if it is closed) by issuing the command

Dataset → *Query...* A dialog (see *this Figure*) will be displayed where you can create a query and select the range of rows to which the query will apply. Notice that, *you can make complex queries, i.e. queries that involve* *more than one table field. However the queried fields cannot be multidimensional or contain data with Complex data type.*

ViTables always do its best for not being frozen due to out of memory problems when you do complex queries or the queried table is huge (or both) but it is not guarateed that it can achieve this goal.

The selected rows are stored in a new table (but not removed from its original location) that we will call filtered table from here on. Filtered tables are stored in a temporary database [1] labeled as *Query results* in the databases tree viewer. The *Query results* node is always placed at the bottom of the databases tree.

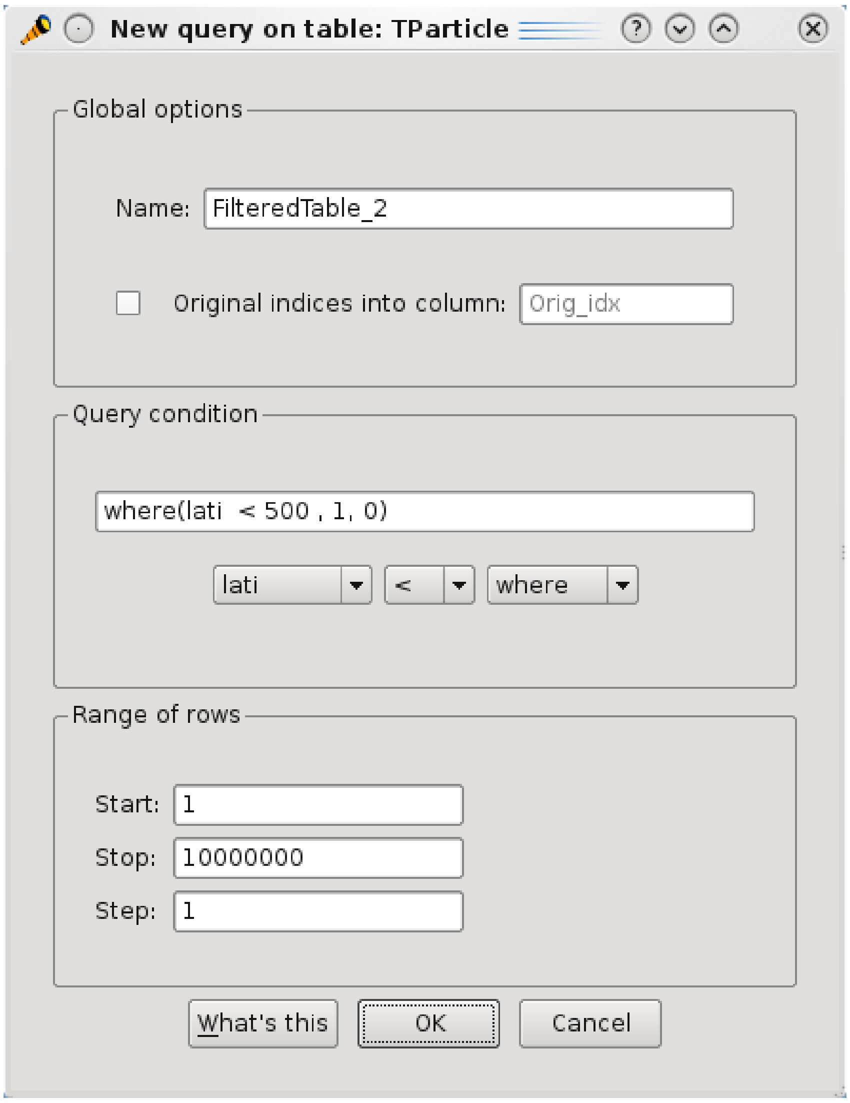
Filtered tables can be edited as any other leaf opened in read-write mode.

By default an automatic title FilteredTable_X is given to the X-th filtered table created. In addition, those tables have three user attributes that are, in principle, only defined for filtered tables. These attributes are:

query the applied query

query_path the full path of the queried file

query_table the full path of the queried table in the object tree hierarchy


New query on table: TParticle

Global options

Name: FilteredTable_2

☐ Original indices into column: Orig_idx

Query condition

where(lati < 500 , 1, 0)

lati ▾ < ▾ where ▾

Range of rows

Start: 1

Stop: 10000000

Step: 1

Figure 3.4: The New Query dialog

[1] Every time a *ViTables* session starts, a new temporary database is created from scratch. This database has a flat structure and is stored with a unique name in a temporary directory so the operating system will remove it every time the directory is cleaned.

CHAPTER

FOUR

EDITING FILES

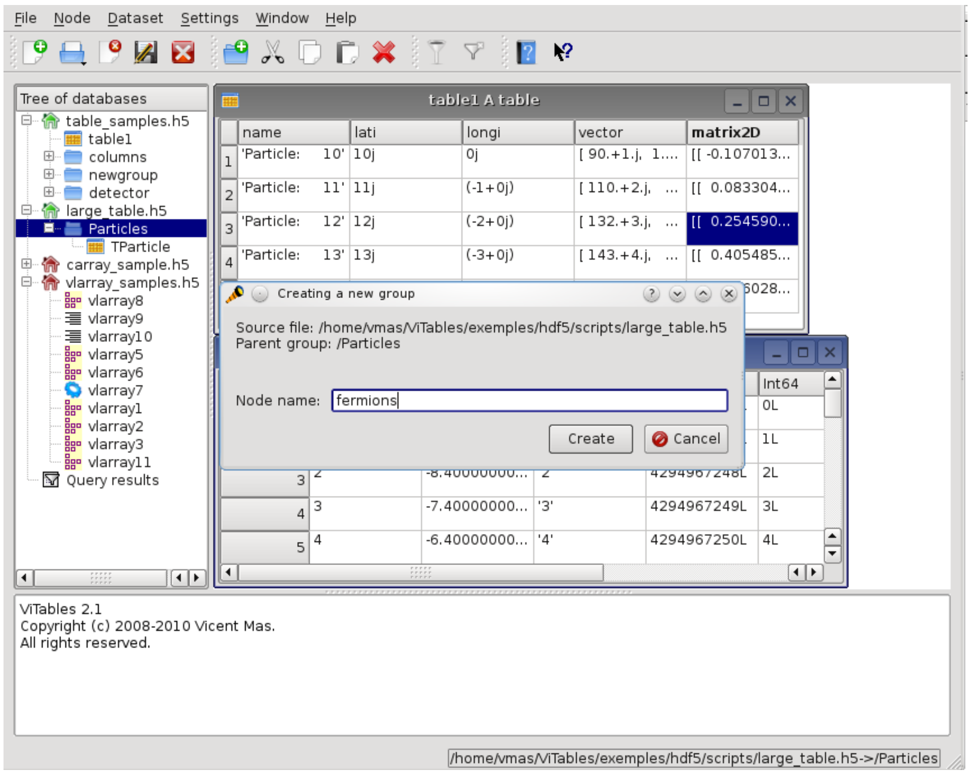
In this chapter we are going to describe briefly the editing capabilities of *ViTables*.

4.1 Creating Complex Hierarchies

ViTables supports a complete set of editing operations on files and nodes. The result of these operations is made immediately visible on the databases tree viewer.

To create a new file in write mode, just issue the command *File* → *New...* (`Ctrl-N`). By default, the file will be created with a `.h5` extension but you can provide your desired extension.

You can add new empty groups to a writable file as easily as you create a new file. Simply select a group on the the tree pane and launch the command *Node* → *New group...*. A new, empty group will be added to the previously selected group. By combining this operation with file creation, you can easily create complex hierarchies. Later on, you can populate the hierarchies with real data using your PyTables programs.



	name	lati	longi	vector	matrix2D
1	'Particle: 10'	10j	0j	[90.+1.j, 1....	[[-0.107013...
2	'Particle: 11'	11j	(-1+0j)	[110.+2.j, ...	[[0.083304...
3	'Particle: 12'	12j	(-2+0j)	[132.+3.j, ...	[[0.254590...
4	'Particle: 13'	13j	(-3+0j)	[143.+4.j, ...	[[0.405485...
					5028...

Source file: /home/vmas/ViTables/exemples/hdf5/scripts/large_table.h5
Parent group: /Particles

Node name: fermions

					Int64
					0L
					1L
3	2	-8.40000000...	2	4294967248L	2L
4	3	-7.40000000...	'3'	4294967249L	3L
5	4	-6.40000000...	'4'	4294967250L	4L

ViTables 2.1
Copyright (c) 2008-2010 Vicent Mas.
All rights reserved.

/home/vmas/ViTables/exemples/hdf5/scripts/large_table.h5->/Particles

Figure 4.1: Creating a new group

4.2 Editing Object Trees

Files opened in write mode can be modified by moving their nodes (groups and leaves) around. From the *Node* menu you can copy, paste, cut, rename or delete any selected node (except root groups). Typical keyboard shortcuts are available for copy and paste operations. Of course, you can drag and drop nodes from one location to a different one using the mouse.

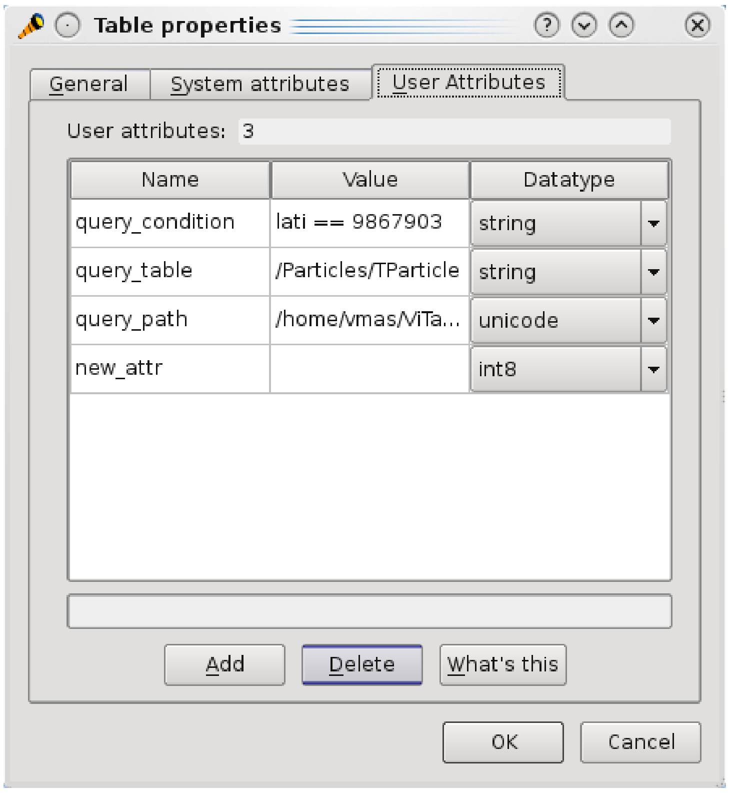
Nodes can be moved to a different location in the object tree, but can also be reallocated in a different file. This way you can *merge* open files in a very flexible and comfortable way.

As usual, while an operation is being performed on a given node, the shape of the mouse cursor will change into a clock, reminding you that a PyTables operation is being executed.

Given a node opened in read-write mode you can edit its user attributes from the User attributes page (see *this Figure*) in the node Properties dialog. This page contains the user attributes table. You can add and remove attributes with the respective buttons or you can edit any existing attribute by clicking the table cell that you want to modify and introducing the new value. This way you can change name, value and type of any existing attribute.

Note: multidimensional attribute values are not supported by *ViTables*. Also be aware that scalar attributes will be saved as scalar Numpy objects instead of serialized using cPickle (which used to be the default PyTables behavior). This way you will be able to read them using generic HDF5 tools, not just PyTables.

Finally, the value of the TITLE system attribute can also be edited. Just click its cell in the System Attributes tab and enter the desired value.



User attributes: 3

Name	Value	Datatype
query_condition	lati == 9867903	string
query_table	/Particles/TParticle	string
query_path	/home/vmas/ViTa...	unicode
new_attr		int8

Figure 4.2: Editing user attributes

4.3 Editing Leaves

At the moment of writing, editing the real data stored in leaves has not yet been implemented.

CHAPTER

FIVE

CONFIGURING VITABLES

As mentioned earlier, many aspects of the *ViTables* behaviour can be customized by you through the *Settings → Preferences* command.

It shows a dialog offering you several customization possibilities. The dialog is made of three stacked pages, *General*, *Look & Feel* and *Plugins*.

The *General* page allows to change the *ViTables* behavior at startup. You can set the initial working directory to be that one from which *ViTables* is starting or to be the last used working directory. And you can recover your last working session.

The *Look & Feel* page allows to change visual aspects of the application such as fonts, colors or even the general style, so you can adapt the global aspect of *ViTables* to what would be expected on your platform.

From the *Plugins* page the plugins can be managed. You can add or remove searching paths for plugins and enable or disable the available plugins

The *OK* button will apply the new settings and make them permanent by saving them in the Windows registry, in a configuration file (on Unix platforms) or in a plist file (on Mac OS X platforms). Even if settings are stored in a plain text file editing it by hand is not recommended. Some settings, like fonts or geometry settings [1], are stored in a way not really intended to be modified manually.

The configuration file is divided into sections, labeled as [section_name]. Every section is made of subsections written as key/value pairs and representing the item that is being customized. Currently the following sections/subsections are available:

[Geometry] HSplitter the size of the horizontal splitter

[Geometry] Layout the position and size of toolbars and dock widgets

[Geometry] Position the position and size of the application window

[Geometry] VSplitter the size of the vertical splitter

[HelpBrowser] Bookmarks the list of current bookmarks of the help browser

[HelpBrowser] History the navigation history of the help browser

[Logger] Font the logger font

[Logger] Paper the logger background color

[Logger] Text the logger text color

[Look] currentStyle the style that defines the application look & feel. Available styles fit the most common platforms, i.e., Windows, Unix (Motif and SGI flavors), and Macintosh

[Plugins] Enabled the list of full paths of enabled plugins

[Plugins] Paths the list of paths where plugins will be searched

[Recent] Files the list of files recently opened

[Session] Files the list of files and views that were open the last time *ViTables* was closed

[1] Entries in the Geometry section allow for keeping the aspect, size and position of the application window between sessions.

[Startup] lastWorkingDir the last directory accessed from within *ViTables* via Open File dialog

[Startup] restoreLastSession the last working session is restored (if possible) which means that both files and leaves that were open in the last session will be reopen at application startup.

[Startup] startupWorkingDir possible values are *current*, and *last*. These values indicate how the application will setup the startup working directory.

[Workspace] Background the workspace background brush

APPENDIX

A

ABOUT PLUGINS

Since version 2.1 *ViTables* has a simple but powerful plugins framework.

If you are interested in writing plugins the next paragraphs can be of utility. If not you can skip to the list of available plugins.

Plugins can live anywhere in your hard disk. A plugin can be a pure Python module or can have a package structure (i.e. a directory with a `__init__.py` file). Packages can have as many directories as you want but plugins must be located at top level of the package.

The use of contracts is not enforced for writing plugins so you have nearly complete freedom for writing. The only requirement that you must fulfil is to declare in your plugin a variable named `plugin_class` and set it to the name of the class invoqued when your plugin is executed by *ViTables*.

In some cases it can be useful to use convenience variables or methods. For instance, if you are going to load the *Menu* plugin then you may be interested in declare the `__version__` variable in your plugin. You can also consider to define methods `configure()` and/or `helpAbout()`.

Of course some knowledge (not necessarily a deep one) of the *ViTables* code is required in order to bind your plugin to the application core. This task is commonly achieved via the menu bar of the main window or via the signals/slots mechanism (convenience signals can be defined in the application if needed).

If you need more help just send an email to developers or ask to the *ViTables* Users' Group.

Three plugins are currently distributed along with the application:

Menu adds a *Plugins* menu to the menu bar. For every loaded plugin this menu has an entry from which a short description about the plugin is shown to users.

Time series formats time series in a human friendly way. It supports PyTables time datatypes and PyTables time series created via scikits.timeseries module. The format used for displaying times can be configured by user via the *Menu* plugin or editing by hand the `time_format.ini` configuration file.

CSV provides import/export capabilities from/to CSV files.

APPENDIX

B

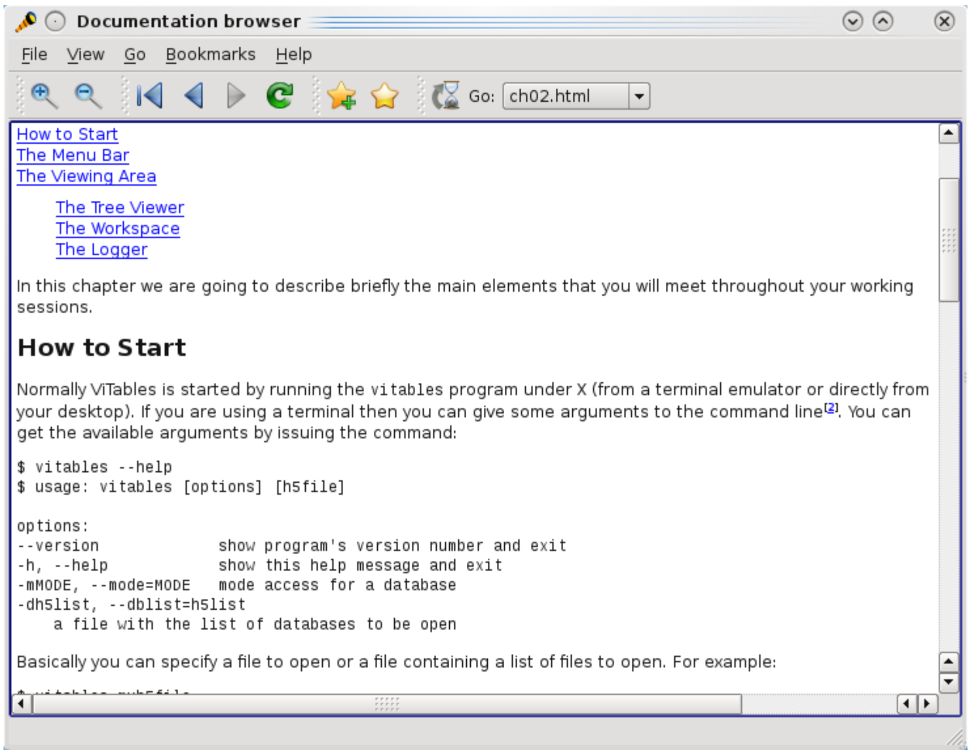
THE HELP BROWSER

ViTables comes with its own fully-integrated documentation browser. It allows the *ViTables* User's Guide to be browsed without leaving the current working session and without opening external applications. You can start the browser issuing the *Help* → *User's Guide* command or from the toolbar.

The help browser is a small HTML browser for *local* documents. Despite its small size it exhibits some nice features

- bookmarks
- session history
- easy document navigation through navigation buttons

A nice feature of bookmarks is that they can be navigated while they are being edited with the Bookmarks Editing dialog. Simply double click on a bookmark and it will be displayed in the browser.



How to Start
The Menu Bar
The Viewing Area

The Tree Viewer
The Workspace
The Logger

In this chapter we are going to describe briefly the main elements that you will meet throughout your working sessions.

How to Start

Normally ViTables is started by running the `vitables` program under X (from a terminal emulator or directly from your desktop). If you are using a terminal then you can give some arguments to the command line[2]. You can get the available arguments by issuing the command:

```
$ vitables --help
$ usage: vitables [options] [h5file]

options:
--version             show program's version number and exit
-h, --help            show this help message and exit
-mMODE, --mode=MODE   mode access for a database
-dh5list, --dblist=h5list
    a file with the list of databases to be open
```

Basically you can specify a file to open or a file containing a list of files to open. For example:

Figure B.1: The Users' Guide browser

INDEX