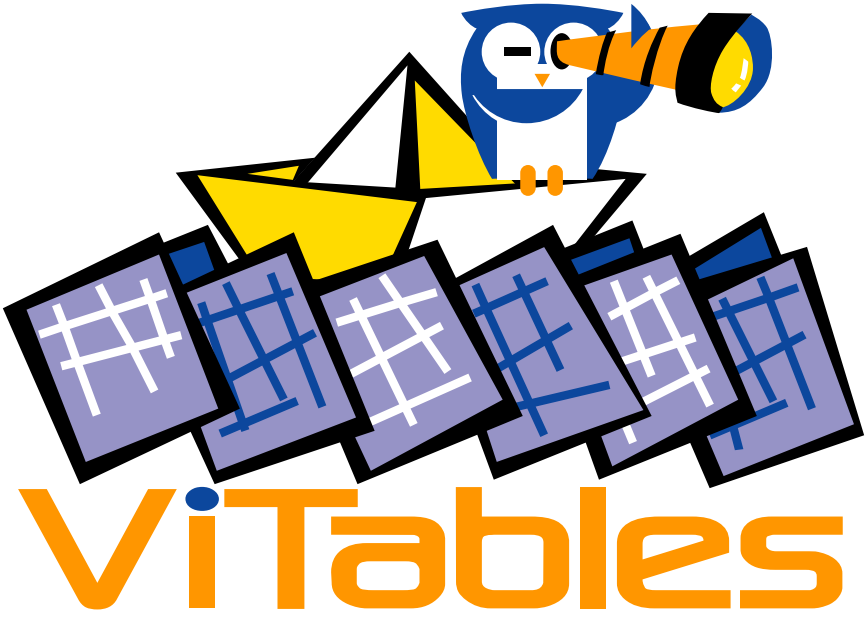


ViTables 2.0 User's Guide

Vicent Mas

ViTables 2.0 User's Guide

Vicent Mas

Publication date December, 12, 2008

Table of Contents

List of Figures

Chapter 1. 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, MacOSX, 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, editing and querying capabilities. Most of them are listed below. Details are discussed in the related chapters.

1.2.1. Browsing 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.

1.2.2. Editing Capabilities

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

- File creation and renaming.
- 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:

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

- *Ability to view really large datasets*.
- Support for complex table queries.
- Flexible and fast dataset navigation.
- A logger area, where the result of user requested operations is printed.
- Several levels of help are available: context help , tooltips, status bar...
- Configurable look and feel.

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 ≥ 2.4, PyTables ≥ 2.0 (so you have to fulfill its own requirements), the Qt4 ≥ 4.4 library and PyQt4 ≥ 4.4.

At the moment, ViTables has been fully tested on Linux and Windows XP platforms. Other Unices should run just fine when using the Unix version of ViTables 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 [http://www.python.org] 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

Currently there is no Windows binary installer for ViTables. So it has to be installed from sources following the instructions given in the previous sections.

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
```

1.4.4. Further Reading

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

Questions and feedback can be mailed to the developers.

Chapter 2. 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

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[1]. 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 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 the toolbar.

[1] Currently command line arguments are available only in Linux platforms.

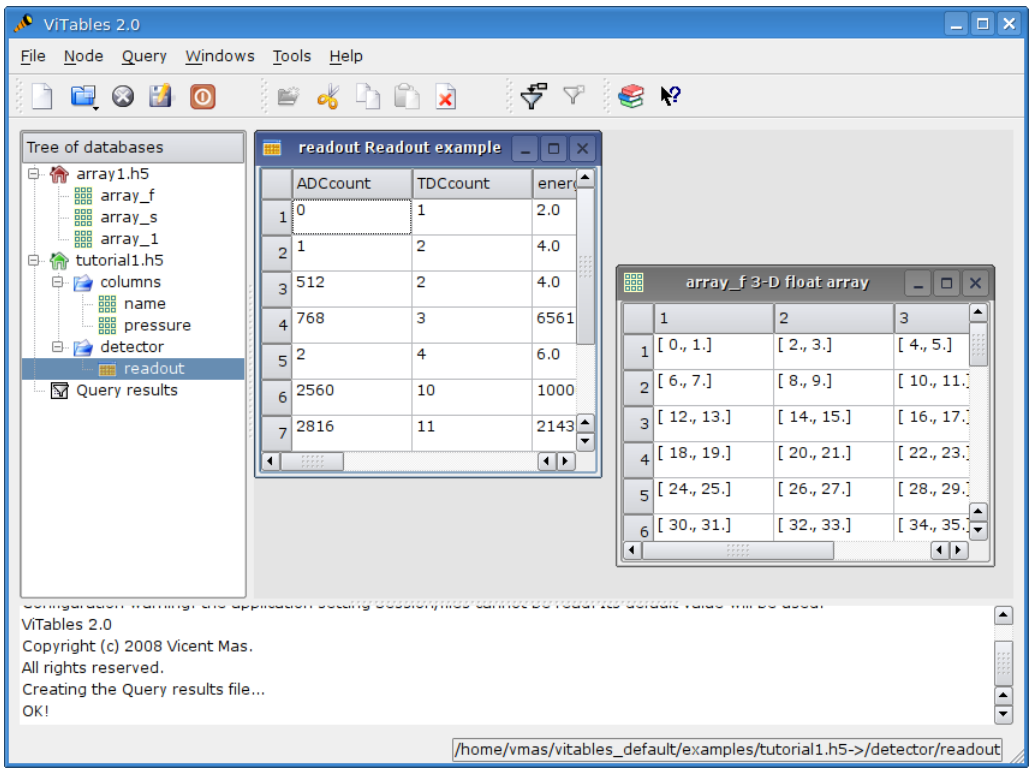


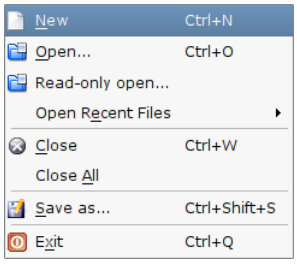
Figure 2.1. The main window

2.2. The Menu Bar

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

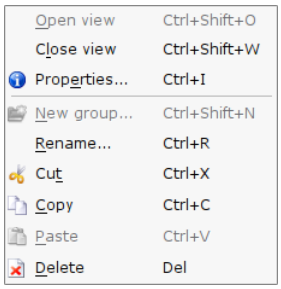
File menu

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.



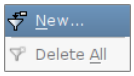
Node menu

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



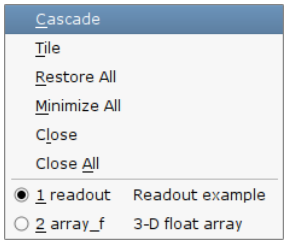
Query menu

With this menu you can make selects in any table. The result of your selects will be available under the Query Results hierarchy in the tree pane.



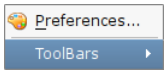
Windows menu

The Windows 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.



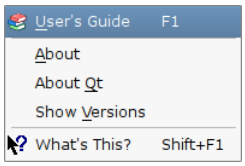
Tools menu

This is the menu from which the application can be customized. Customization includes behavior and look and feel. See Appendix A for more information on this subject. From this menu you can also show, hide and line up the application toolbars. At the moment four toolbars are available: File, Node, Query and Help.



Help menu

The Help menu displays this User's Guide in HTML4 format and a couple of *About* boxes. The Show Versions entry shows the version numbers of the libraries being used by ViTables (including PyTables related libraries, like Zlib or LZO). Finally, from this menu you can enter the *What's This* mode for context help.



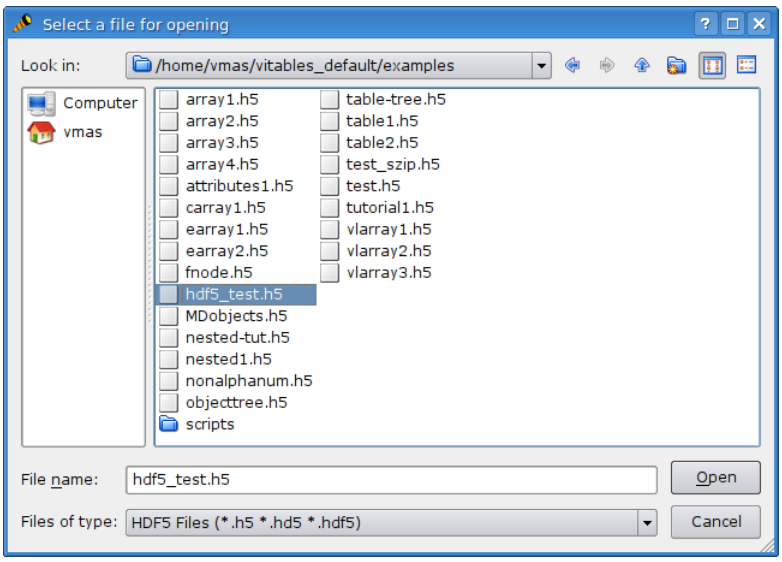
2.3. The Viewing Area

As mentioned before, the viewing area is divided into three regions: the object 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 Tree Viewer

After starting your session, you are likely to open some files. Just drag the file(s) you want to open into the tree 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**) [2] and choose a file.

[2]Remember that files can be opened in two different modes: read-write and read-only. In read-only mode all editing functions are disabled, so files cannot be modified. The opening mode of a file can be inferred from the icon of its root node, see Figure 2.3



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

The root node It is the node from which all other nodes hang.

Groups Groups are nodes that contain other nodes.

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

Once the object tree has been created, its root node is added to the tree viewer (see Figure 2.2), at the top left side of the viewing area. By double-clicking on it, the root node is opened, and the tree structure below it is displayed. This way you can easily browse and manage the different nodes stored in the PyTables file.

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.

Working with object trees is really easy. 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 leave will display its content in 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. Finally, 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.

A node can be renamed in-place (without raising the Rename dialog) by keeping pressed the Shift key while the node is double clicked.

The object tree can be navigated with the keyboard too. Pressing Enter will expand the selected node if it is a group or will open it if it is a leaf. The + and - keys expand an collapse groups.

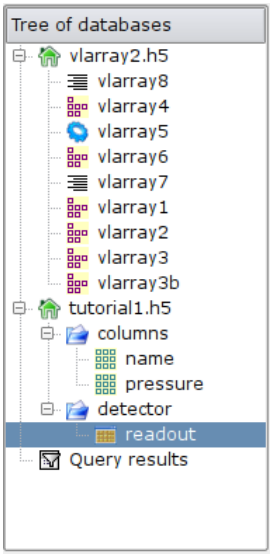


Figure 2.2. The tree viewer

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

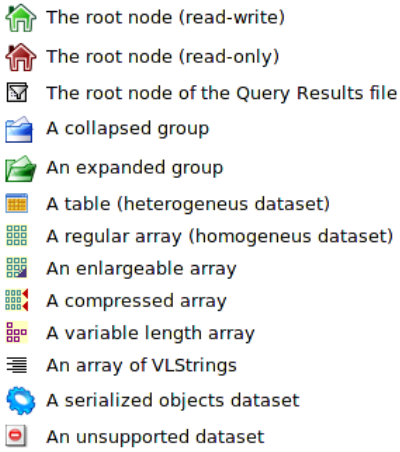


Figure 2.3. The node symbols

2.3.2. The Workspace

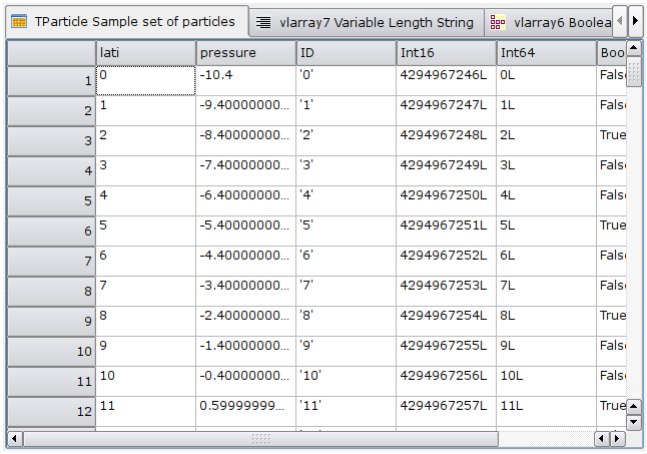
At this point you should have one or more files opened, and their object trees displayed in the tree viewer. Your next step will be to select a leaf and display its data. Remember that the object tree imitates the structure on disk, which makes it very easy to browse the hierarchy of the file and locate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

Note that the 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 in the workspace. The opposite is also true, click on a view in the workspace and its node will be automatically selected in the tree viewer.

The Windows popup menu provides some additional commands that will help you to manage and arrange the opened views. From this menu you can, for instance, rearrange the opened views, see the list of open views (which is particularly useful when the workspace is cluttered with so many open views that it's difficult to find the one you want) or close all open 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 Windows popup menu.



	lati	pressure	ID	Int16	Int64	Boo
1	0	-10.4	'0'	4294967246L	0L	Fals
2	1	-9.40000000...	'1'	4294967247L	1L	Fals
3	2	-8.40000000...	'2'	4294967248L	2L	True
4	3	-7.40000000...	'3'	4294967249L	3L	Fals
5	4	-6.40000000...	'4'	4294967250L	4L	Fals
6	5	-5.40000000...	'5'	4294967251L	5L	True
7	6	-4.40000000...	'6'	4294967252L	6L	Fals
8	7	-3.40000000...	'7'	4294967253L	7L	Fals
9	8	-2.40000000...	'8'	4294967254L	8L	True
10	9	-1.40000000...	'9'	4294967255L	9L	Fals
11	10	-0.40000000...	'10'	4294967256L	10L	Fals
12	11	0.59999999...	'11'	4294967257L	11L	True

Figure 2.4. A tabbified workspace with 3 leaves

2.3.3. The Logger

The logger is the long region placed at the bottom of the viewing area, see Figure 2.5. 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.

Figure 2.5. The logger

Chapter 3. Browsing and Querying Datasets

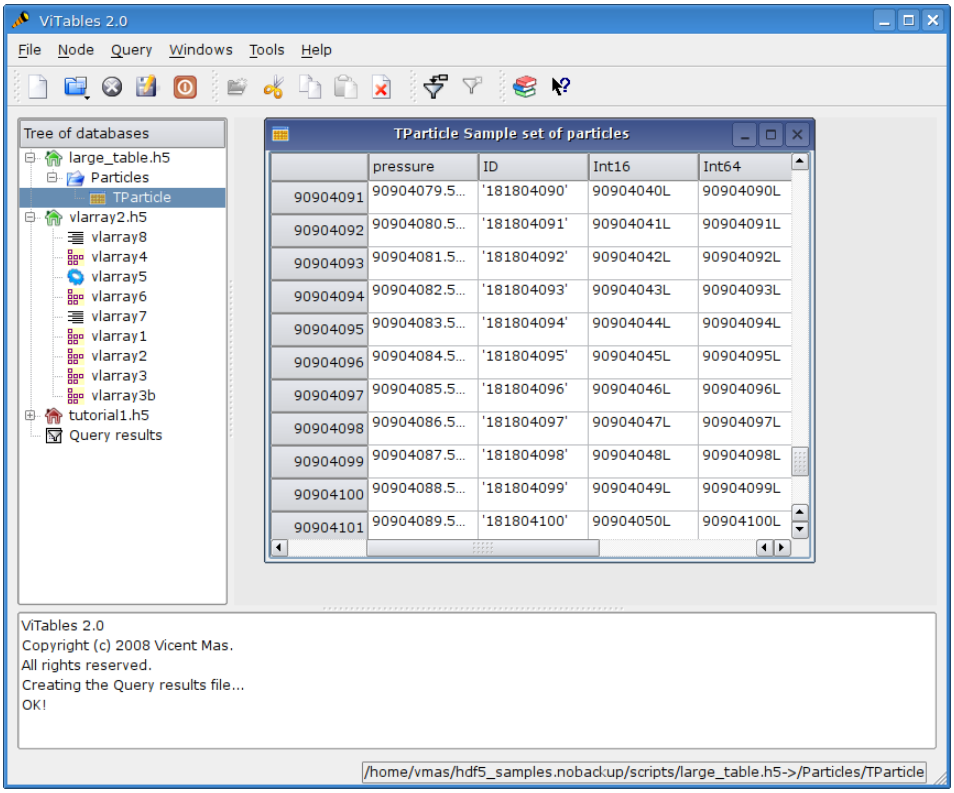
In this chapter we are going to describe how the information contained in a dataset can be navigated and selected.

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.

Datasets are displayed in views, that are spreadsheet-like windows. Then can be navigated as you expected: via scrollbar, keyboard or wheel of mouse. The interesting thing 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.



Browsing a large dataset

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. file the node belongs to, name, path, etc. The System and User tabs contain tables that describe the attributes of the node.

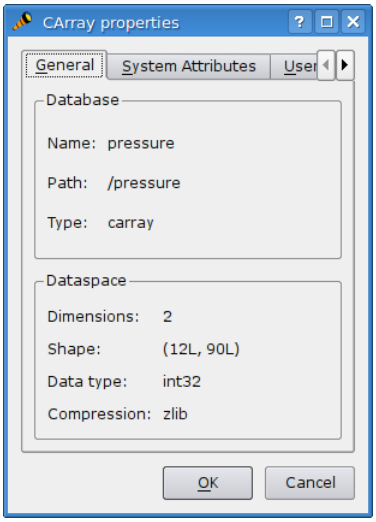


Figure 3.1. The Properties dialog

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).

The next figure shows some of these mechanisms in action.

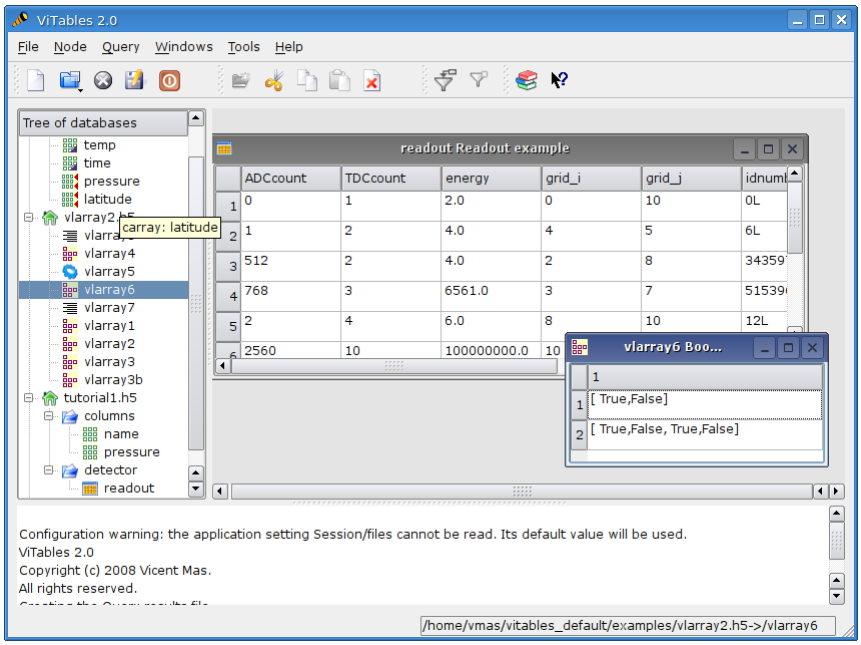


Figure 3.2. A common working session with ViTables

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 Query → New... A dialog (see Figure 3.3 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.*

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] with a flat structure. This database is always displayed in the tree viewer under the label Query Results. Filtered tables can be edited as any other leaf opened in read-write mode.

The views related to filtered tables are easily distinguished from the other views because of their title: it is made with the query used to get the filtered table. In addition, these 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

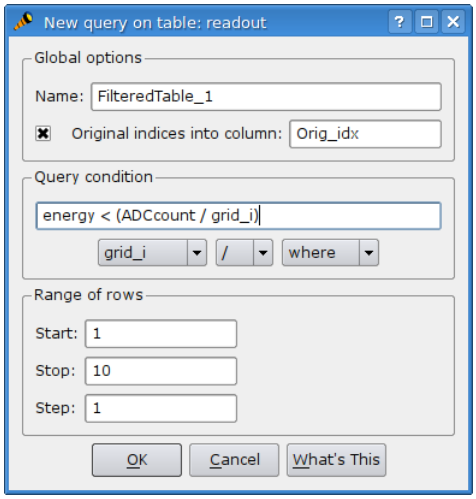


Figure 3.3. The New Query dialog

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

Chapter 4. Editing Files

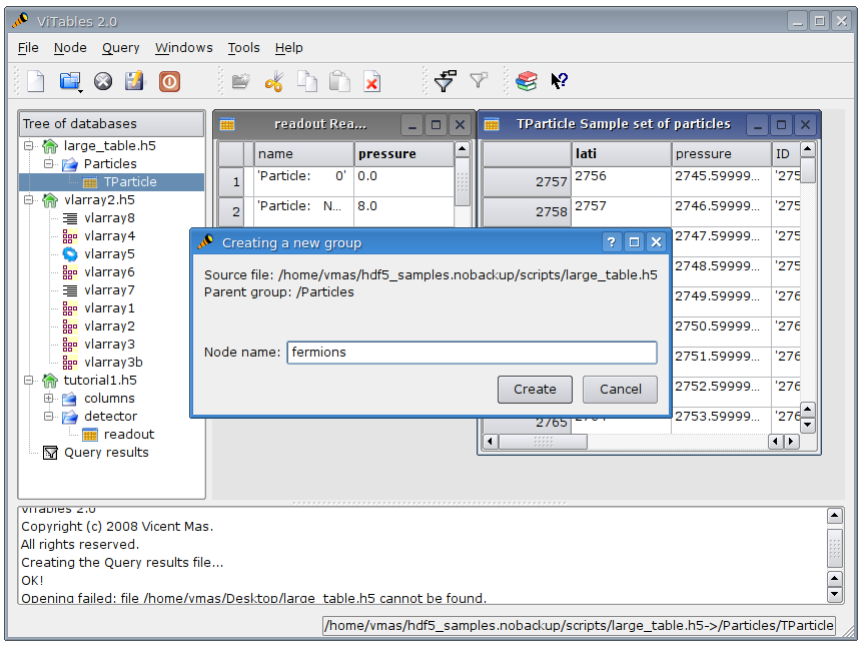
In this chapter we are going to describe briefly the editing capabilities of ViTables.

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 in the tree view.

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 in the the tree view and launch the command Node → New. 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.



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, 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.

Once you have created a node you can edit its user attributes from the User attributes page (see Figure 4.1) 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.

Please note that 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.

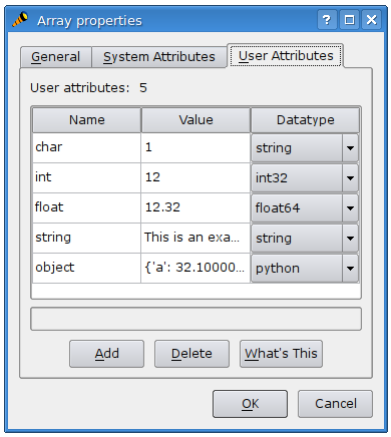


Figure 4.1. Editing user attributes

4.3. Editing Leaves

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

Appendix A. The Configuration File

Many aspects of the ViTables behaviour can be customized by you through the Tools → Preferences command. It shows a tabbed dialog, offering you several customization possibilities. The dialog is made of two pages, General and Look & Feel.

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.

The OK button will apply the new settings and make them permanent by saving them in the Windows registry or in a configuration file (on Unix platforms). The full path of this file will be `$HOME/.vitables/vitables.org/ViTables.conf`. Although it is a plain text file and ViTables could be configured by editing it by hand, this is not recommended. Some settings, like fonts or geometry [1] settings, 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
[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
[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.

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

[Workspace] Background the workspace background brush

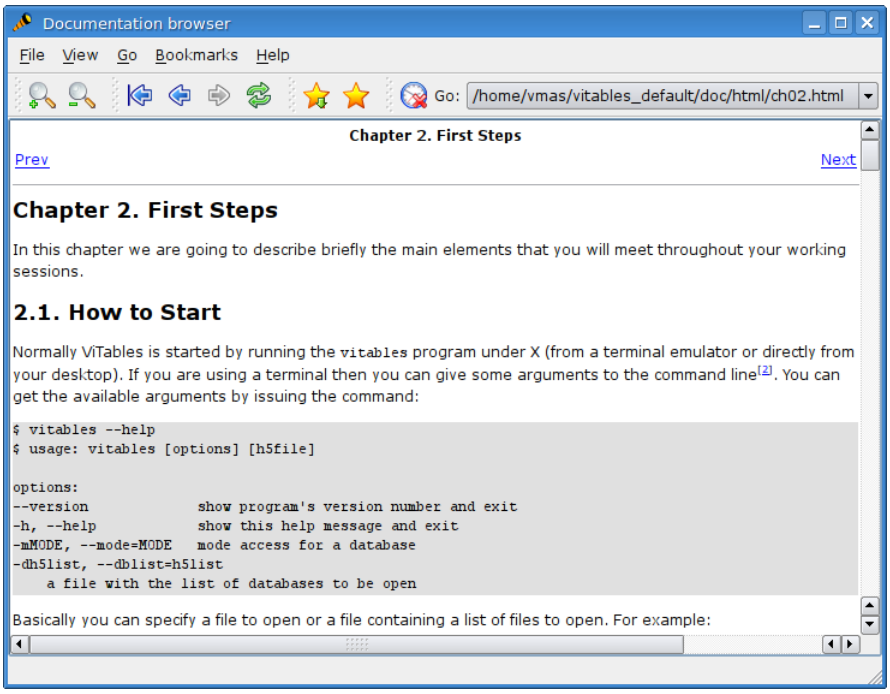
Appendix B. The Help Browser

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

- capable of multiple views
- bookmarks
- session history
- list of most recently opened files
- 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.



The documentation browser