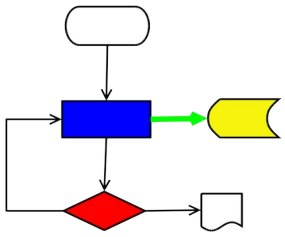
VICI



Visual Chart Interpreter

User and Administration Guide

Brenton Ross

Publication History

Date	Who	What Changes
12 March 2016	Brenton Ross	Initial version. Release 0.4
8 May 2016	Brenton Ross	Added Search. Release 0.5
13 June 2016	Brenton Ross	Added first cut of editor. Release 0.6
25 November 2016	Brenton Ross	Added run time and reference. Release 0.7
11 April 2018	Brenton Ross	Added command and search into editor. Release 0.9
5 May 2018	Brenton Ross	Added run and debugging. Release 0.10
7 May 2018	Brenton Ross	Updated the admin section. Release 0.10
13 May 2018	Brenton Ross	Simplified the admin section.
7 October 2018	Brenton Ross	Update for release 0.11
29 November 2018	Brenton Ross	Added instructions for installing a script.
1 February 2019	Brenton Ross	Added instructions for scheduling a script.
14 May 2019	Brenton Ross	Update for Release 0.12
29 June 2025	Brenton Ross	Update for release 0.12.920

Table of Contents

1 Introduction

This document contains the user and administration guides for the Visual Chart Interpreter (VICI).

1.1 Background

VICI gives a Linux desktop user the ability to create scripts that use the command line programs. These programs provide a rich set of functions for manipulating text files, however they are normally only available via a terminal session.

1.2 Scope

The guide covers the administration of the programs, including installation, configuration, and the operation of the vici-admin program to describe commands for the users.

The guide provides a tutorial and reference for the flowchart editor that forms the core of VICI. This guide will include instructions for creating a script, installing it into the desktop menus, and scheduling a script for later running.

Finally the guide provides a tutorial and reference for the run time program showing how it can be used to execute the VICI scripts.

1.3 Overview

Section 2 will be a quick introduction for the impatient.

Section 3 is a tutorial for the editor.

Section 4 is a tutorial for the run time.

Section 5 is a full reference for all the features of VICI.

Section 6 is a guide for the administration program.

Section 7 is an installation guide.

1.4 Audience

This document is intended for the users and administrators of VICI.

1.5 Glossary

The following terms are used in this document. They are either terms applicable specifically to the VICI project, or where they are used with a more specific meaning than they would be in normal everyday usage.

Term	Meaning
Administrator	A person responsible for ensuring that a computer system performs correctly. Usually an administrator will have additional privileges that allow them to perform tasks that an ordinary user is not allowed to.
Debugging	The process of determining the reason for a program not behaving as desired, and applying an appropriate fix.
Dialog	A window which is temporarily displayed by a GUI program, usually to allow the user to enter some information, or to advice of some change within the program.
Drag-n-drop	A technique that allows a user to drag a symbol (including text) from one program to another.
EBNF	Extended Backus–Naur Form. A language for specifying languages.
Editor	A program used to enter or modify the files used by other programs.
Flow Chart	A diagram that shows the sequence of events in some process, usually including special symbols for decision.
GUI	Graphical User Interface. A means of interacting with a computer system using windows, icons, menus and a pointing device (mouse).
IDE	Integrated Development Environment. A computer program designed to facilitate the creation of other computer programs. The VICI Editor might be considered to be a simple IDE.
info page	An help file that explains the purpose and usage of UNIX commands.
man page	An help file that explains the purpose and usage of UNIX commands
Parameter	Information passed to a computer program as it starts. This is normally used to modify the actions of the program.
Paths	The set of directories that a shell will examine when attempting to find a command.
Program	A set of instructions that control the actions of a computer.

Term	Meaning
script	The set of directions that control vici to perform some user defined operations. This may refer to the diagram in vici-editor, or the XML file in which it is saved.
stderr	The default error file for a program. This is typically connected to a terminal session so that the user can see any error messages generated by the program.
stdin	The default input file for a program. This is typically connected to the keyboard so that the user's input is fed to the program.
stdout	The default output file for a program. This is typically connected to a terminal session so that the user may see the results from the program.
Syntax Chart	A diagram describing the allowed syntax for a particular language.
Tag	An identifier that allows other objects to be classified in some way.
Thread	A sequence of commands that are executed at the same time as the commands in other threads.
VICI	The name of the project and the system.
vici	The name of the program that runs VICI scripts.
vici-admin	The name of a GUI for preparing commands to make them easier to use with vici-editor.
vici-cli	The name of a command line program for running the scripts.
vici-editor	The name of the program used to edit VICI scripts.
vici-search	The name of a GUI for searching for commands.
vici-syntax	The name of a GUI for creating syntax diagrams.
vici-cron	The name of the program that starts scheduled scripts.

2 Quick Introduction to Using VICI

This is a bit like trying to write a quick introduction to flying a 747. It isn't going to answer the questions you will inevitably have. Please read the rest of the user guide.

Start by pressing New Chart on the toolbar (to the right of the symbols). This will create an empty chart with a Function object called "Unnamed".

Right click the Unnamed Function object to bring up a context menu, and then select Properties. This will display a small dialog where you can enter the name for the function (and optionally a name for its menu entry in vici.) Typically the first function in the script is its main one, so call it "main".

In Release 0.12 the interpreter tries to start at a function called "main" so you will need to use that name.

We will now add a command to our script.

Select the Command symbol from the palette (top left – it will be outlined with a glow) then click below the Main Function object on the canvas. A Command box will be placed onto the diagram.

Right click the Command box to open the context menu and select Properties. Add the word "echo" for the Command Name and enter "hello world" on the first line of the Command Parameters, then click OK.

Next we define the thread of control.

Select the thin black line from the symbol palette. Click (and release) the Main Function object and then move the mouse to an open area between the the two objects and again click the mouse. This will place a corner object. Now move the mouse to the echo command box and click one more time to complete the line.

You have now completed your first script.

You should experiment with adding other objects and interconnecting lines. You will find that the canvas will disallow some lines as they would be incorrect for a script – such as trying to write to a constant.

Objects and lines can be removed by selecting the Remove option from the context menu. Objects can only be removed if there are no connecting lines.

When you are happy with the script save it using the Save or Save As menu option. You can save the file anywhere, but it is recommended that you set up a directory for your VICI scripts.

3 Editor Tutorial

3.1 *Vici Editor*

The vici-editor program is used to create flow-charts which are to be interpreted by the vici program.

This version (0.12) provides all the tools necessary to create and test a script. Some usability features such as the ability to set fonts and colours have not yet been implemented.

A script can be opened for editing by selecting it in your file manager and using the right mouse button menu to select “Open with Other Application” and selecting “VICI Editor”.

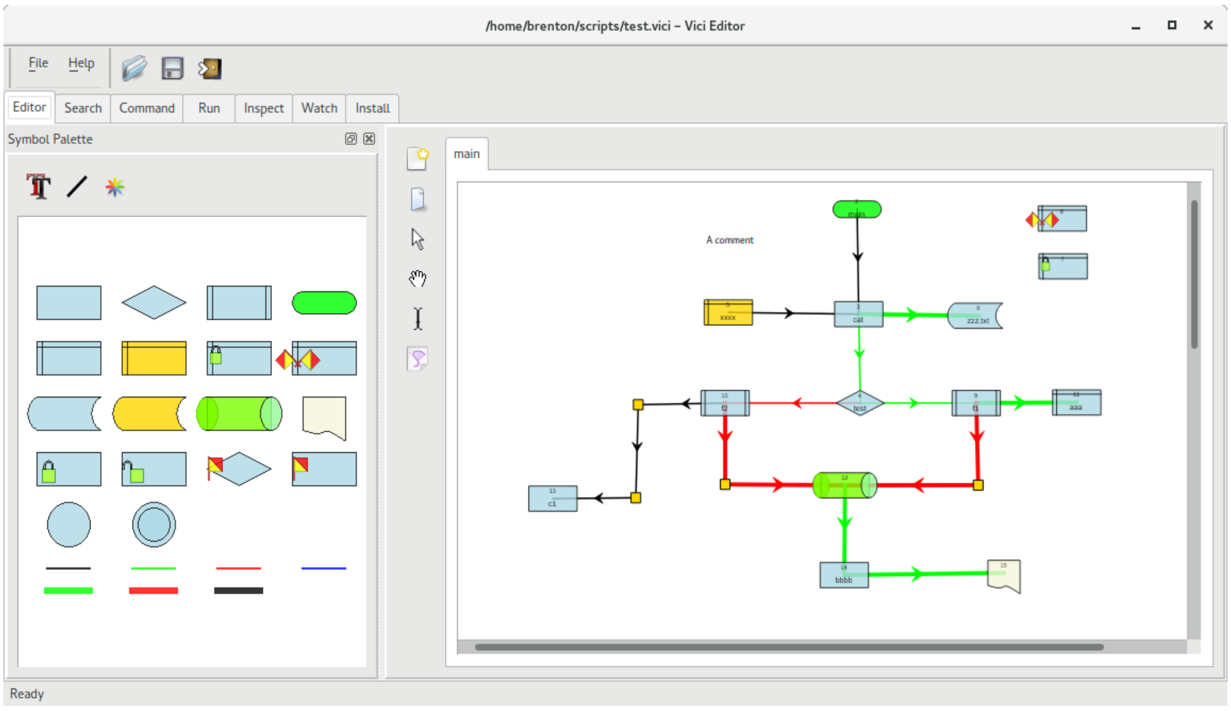
3.1.1 User Interface

The user interface for the editor is divided up into a set of tabs, each of which are described below. The main window provides options for loading and saving scripts.

Select the Open menu or toolbar button to open a file dialog for selecting the script to load.

Select the Save option to save any changes. This will also open a file dialog for selecting a new name for the file if the current chart has not yet been saved.

Select the SaveAs option to save the script with a new name. This then becomes the current name of the script for future Save actions.



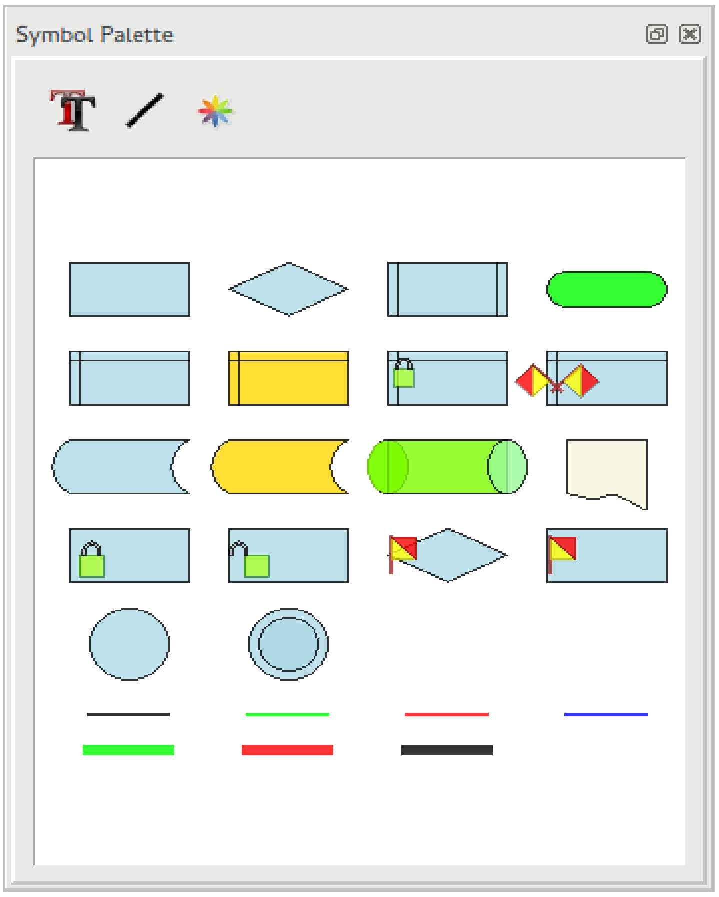
3.2 Symbol Palette

3.2.1 Purpose

The symbol palette on the Editor tab is used to choose the type of symbol that will be placed onto the canvas.

3.2.2 User Interface

The symbol palette has the following user interface. The palette is a docking window that can be dragged from the editor's main window and placed onto the desktop, or positioned at either side of the editor's canvas window. You can also zoom the window, using the mouse wheel, to provide a smaller or larger view if you want.

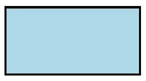
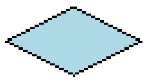
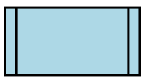
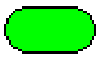
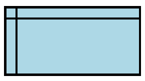
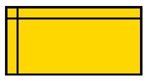
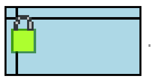
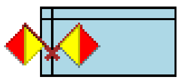
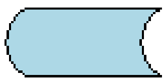


The window also includes a toolbar for selecting fonts, line styles, and colours. In this release (0.12) the dialogs are displayed, but no action is taken.

To place a symbol on the canvas, select the symbol from the palette by clicking on it (it will become surrounded by a “glow”) and then click on the desired location on the canvas.

3.2.3 The Symbols

Each symbol on the palette has a tool-tip that provides the type name. These are replaced by the command and arguments (or file name or variable name) when the symbol is placed on the canvas an the command defined.

	Command: This is used for normal commands.
	Choice: This is functionally identical to Command, but is used on the diagram where the main purpose of the command is to test something.
	Function Call: This is used to make a call to a function within the script.
	Function: This is the start symbol for each function.
	Variable: This is used to indicate a variable that will hold a value.
	Constant: This is for holding values which are not to be altered.
	Mutex: A mutex variable is used by the Lock and Unlock commands (see below) to control access to a sequence of commands.
	Semaphore: A semaphore variable is used by the Wait and Post commands (see below).
	File: This represents a file.

🟨	Inline File: This allows you to embed some text in the script that can be accessed as if it were a file.
🟩	Named Pipe: This allows you to merge streams.
▭	Display: The output from programs can be written to this and displayed by the runtime.
🔒	Lock: Prevents access by other threads of control until Unlock is called.
🔓	Unlock: Allows access by other threads of control.
◆	Wait: Causes the thread of control to wait until a Post is called in another thread of control or a time-out value is reached.
🚩	Post: Allow a thread waiting for Wait to proceed.
●	Threads: This starts one or more simultaneous threads of control and waits until they all complete.
◎	Manifold: This allows you to merge and split streams.
—	Flow: A thread of control connecting commands.
—	Success: A thread of control that is used when the command succeeded.
—	Fail: A thread of control that is used when a command fails. It can be associated with a particular exit code.
—	Signal: used to send an interrupt signal between commands.

🟩	Standard Out: The normal output from a command.
🟥	Standard Error: The error messages from a command.
⬛	Standard Input: The input stream for a command, usually from a file, variable or constant.

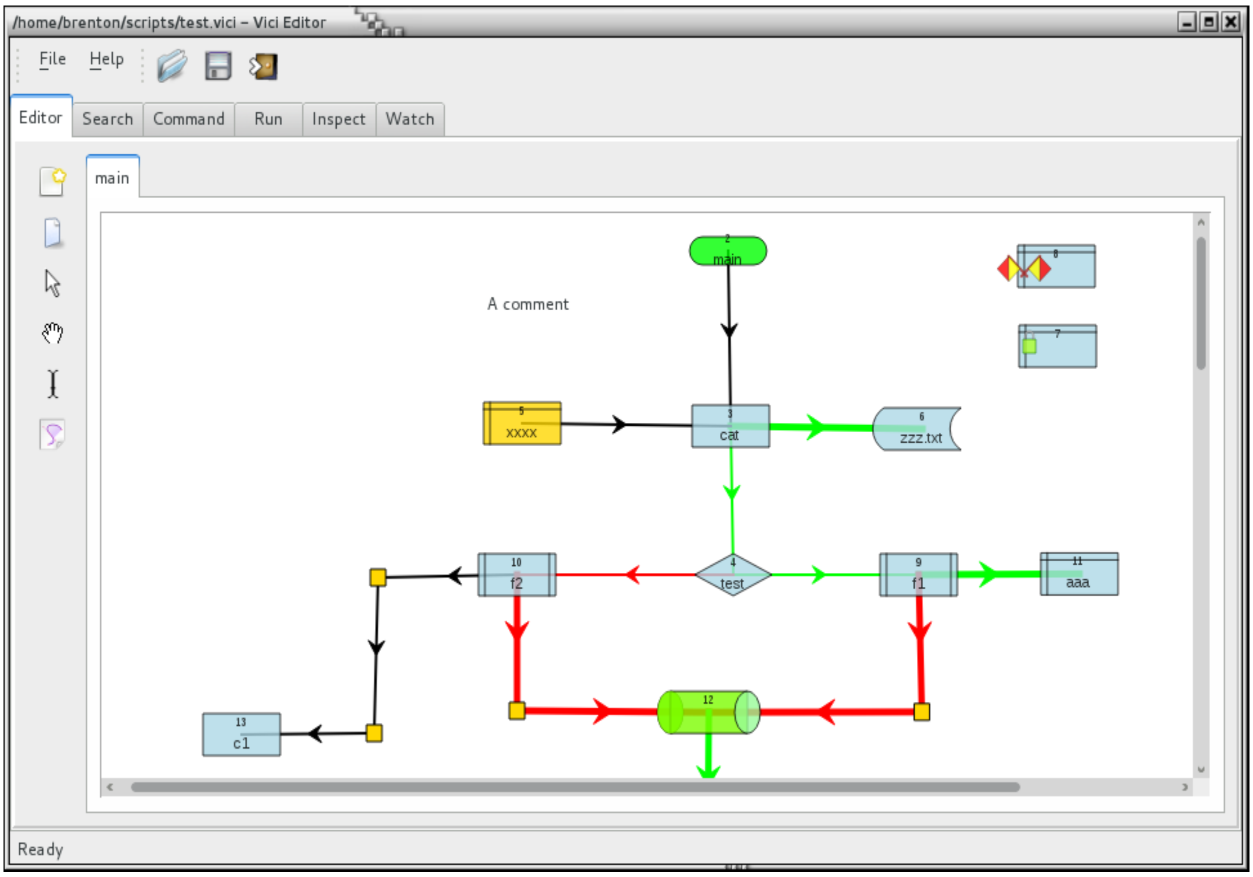
3.3 Canvas

3.3.1 Purpose

The canvas on the Editor tab is the area on which you draw the flow chart for your script.

3.3.2 User Interface

The canvas consists of a toolbar at the left and a set of tabs, one for each function that has been defined.



3.3.3 Toolbar

The canvas tool bar has the following buttons:

- New Chart – this will abandon the existing chart and allow you to begin a new one. A warning will be given if any changes to the existing chart have not been saved.
- New Function – this allows you to add a new function to the script. Each function should be drawn on a separate tab.

- Select – this allows you to select one or several objects by dragging a rubber band (dotted rectangle) around them.
- Drag – this allows you to re-position the diagram by dragging it. (You can also zoom the diagram using the mouse wheel.)
- Text – this allows you to add comments to your chart. Having lots of comments is encouraged.
- Export – this allows you to export the current function tab as an SVG image.

3.3.4 Creating a Chart

This section demonstrates how to create a simple chart.

Start by pressing New Chart on the toolbar. This will create an empty chart with a Function object called “Unnamed”.

Right click the Unnamed Function object to bring up a context menu, and then select Properties. This will display a small dialog where you can enter the name for the function (and optionally a name for its menu entry in vici.) Typically the first function in the script is its main one, so call it “main”.

In Release 0.12 the interpreter tries to start at a function called “main” so you will need to use that name.

We will now add a command to our script.

Select the Command symbol from the palette (top left – it will be outlined with a glow) then click below the Main Function object on the canvas. A Command box will be placed onto the diagram.

Right click the Command box to open the context menu and select Properties. Add the word “echo” for the Command Name and enter “hello world” on the first line of the Command Parameters, then click OK.

Next we define the thread of control.

Select the thin black line from the symbol palette. Click (and release) the Main Function object and then move the mouse to an open area between the the two objects and again click the mouse. This will place a corner object. Now move the mouse to the echo command box and click one more time to complete the line.

You should experiment with adding other objects and interconnecting lines. You will find that the canvas will disallow some lines as they would be incorrect for a script – such as trying to write to a constant.

Objects and lines can be removed by selecting the Remove option from the context menu. Objects can only be removed if there are no connecting lines.

When you are happy with the script save it using the Save or Save As menu option. You can save the file anywhere, but it is recommended that you set up a directory for your VICI scripts.

3.4 Searching for Commands

3.4.1 Purpose

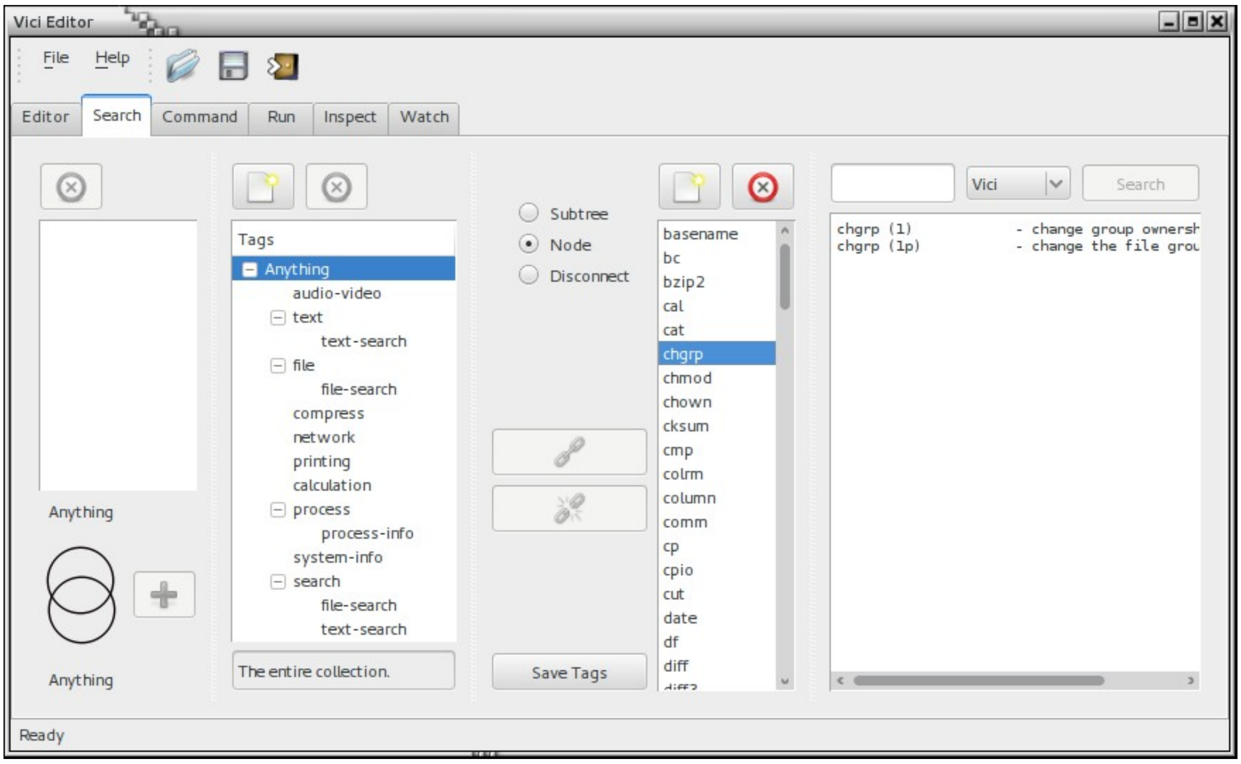
One of the problems confronting anyone that wants to write a script, or just run a command in a terminal, is finding out which command will provide the desired results. This problem is compounded for the users of VICI who are assumed to have little experience with running command line programs.

A second problem is that once found it is all too easy to forget what the command was. Linux commands often have names that are either terribly abbreviated or have no mnemonic character. There are few things more frustrating than remembering that you found what you needed previously but can no longer remember exactly what it was.

The search window (and the stand-alone vici-search program) attempts to address these problems by allowing the user to do key-word searches of the descriptions for commands and to tag and classify the commands so they can be quickly identified later.

3.4.2 The User Interface

The search tab has the following user interface. This interface will also be available as the vici-search program.



```
chgrp (1)             - change group ownersh
chgrp (1p)            - change the file grou
```

The Search window is divided into four panels that can be resized by dragging the vertical splitter bars.

Search Panel

The rightmost panel is where you can search for some key words from the descriptions of the commands. Enter the word or phrase to search for in the top left entry field.

The drop down menu in the top middle allows you select between:

- Vici: The search is over the descriptions for the commands that have been prepared for use in VICI. (See VICI Admin)
- Apropos: The search uses the apropos command to search the Linux man pages for the installed commands.
- Desktop: The search is over the descriptions provided by the desktop files that are associated with the installed GUI programs.
- Package: The search is over the descriptions for the packages in the RPM database. (This means that it won't work for systems that don't use rpm. Support for apt and zypper is planned.)

The search is initiated by clicking on the Search button.

The results for the search are displayed in the search results window.

Programs Panel

To the left of the Search Panel is the Programs Panel that shows the list of programs that have been tagged and classified (see later). Selecting one or more of these programs will cause the results of the whatis command to be displayed in the search results window.

The two buttons at the top of the programs list allow you to add and remove programs from the list.

The programs list shows the programs associated with the currently selected tag (in the next left panel). The “Anything” tag will give the full list of programs.

The radio buttons allow you to control which programs are shown in the list:

- Subtree: The programs associated with all the tags that are children of the selected tag are shown.
- Node: Only the programs associated with the selected tag are shown.
- Disconnect: The list of programs will remain unchanged as tags are selected.

The link and unlink buttons allow you to make or break associations between the programs and the tags. (See below for details on how to use these.)

Tags Panel

To the left of the Programs Panel is the Tags Panel which features a tree of tags that can be used to classify the programs.

The “Anything” tag is the top level of the tree and is associated with all of the programs. This tag cannot be removed.

The buttons above the Tags tree can be used to add or remove tags.

You can put a short description of your new tags in the entry field under the Tag tree. This will be associated with the currently selected tag.

Tags can be copied by dragging and dropping them to new locations within the tree.

The Save Tags button will save your current Tags tree as well as the associated list of programs.

(See below for a description of how to use this to classify programs.)

Search Tags Panel

At the left of the window is the Search Tags Panel. This can be used to create new tags that are the combination of existing tags.

As tags are selected the two labels above and below the intersecting circles are updated so that they show the two most recent selections.

You can then select combinations of the intersecting circles to create a new set of programs. If this is for a non-empty set that is different from either of the selected tags then the + button is enabled. If pressed it adds a new tag in the above list. You can then give a name to the new tag.

The new tags can be dragged to the Tags Panel and inserted into the tree.

3.4.3 Searching with Tags

To find a command the quickest approach is to use the Tags.

For example, say you were after a command for searching for some text.

First, make sure the Node or Subtree option has been selected.

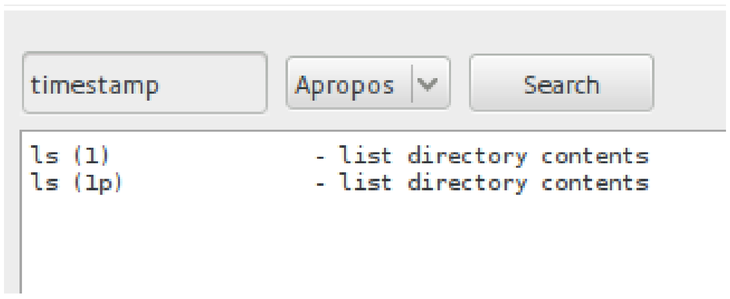
Selecting the “text” tag brings up the rather long list of programs that can be used for text processing. We could select each one in turn and read about what it does, but this might take a while.

Selecting the “search” tag also brings up a list of programs, but only some of these are for searching within files, while others are for searching directories for files.

If we now select the intersection of the two circles the list of programs is restricted to those that are both text and search, and we can quickly select “grep” as the command we are looking for.

3.4.4 Searching for Key Words

If the tags approach (above) doesn't return what you need then you will need to search for a program that has not been classified.



```
ls (1)              - list directory contents
ls (1p)             - list directory contents
```

Enter the word to search for, “timestamp” in the example, in the entry field.

You can then select the area to search using the drop-down.

- The “Vici” option searches the descriptions of the commands that have been prepared for Vici. This should be your first choice since these will be easier to use as a command.
- The “Apropos” option (as shown) searches the man pages. This will be over most of the installed command line programs.

You can also search over the desktop GUI programs or all of the installed packages if you wish, but these probably won't turn up anything that can be used in a VICI script. (*For Release 0.12 the packages option only works for RPM based systems.*)

Click on the “Search” button to run the search. (This may take a moment when searching the package database.)

Once you have found the command that suits your needs you should classify it so that you can find it more quickly next time.

3.4.5 Classifying Commands

This section describes how to use tags to classify commands so that they are easier to find the next time you need them.

Add a New Program

Start by selecting the “Anything” tag. All programs in the programs list are automatically associated with this tag, and selecting it enables the button to add new programs.

Press the “Add a program” button (located above the list of programs) and when the input dialog appears, type in the name of the program and press OK. Copy and paste from the Search panel is also possible.

Associate Program with a Tag

Select the “Disconnect” option. This allows you to select a tag without changing the contents of the program list. At this time your new program is not associated with any other tags so it would disappear from the list.

When a tag and a program are both selected the link button is enabled. Pressing this button will associate the program with the tag.

You can associate a program with more than one tag.

You can confirm the associations by selecting the Node option and then the associated tag. The new program should be in the list.

Note that when the program list is redisplayed it will have been sorted and your new program will be in its proper place rather than at the end of the list where you added it.

3.4.6 Classifying Tags

In addition to classifying the programs with tags you can also classify the tags with other tags to build up a hierarchy (or tree) of tags.

For example the default set of tags has a “text” tag for all programs that can be used to manipulate text files, and below that a “text-search” tag that consists of just those text programs used to search within the text files.

Just as a program can have multiple tags, a tag can also have several parent tags. The “text-search” tag is a child of both the “text” and the “search” tags.

You can drag a tag within the tag tree to make a copy of the tag with a different parent (or just to place it at a different position in the tree). At the moment moving a tag is not supported, but it easy enough to delete them if you want by selecting the tag to delete and pressing the tag delete button (which has “Remove a tag” as its tool-tip).

You can also use the intersecting circles to create new tags.

For example to create a new tag that has the system-info for processes you would first select the “system-info” tag, and then the “process” tag, and then select the intersection area of the two circles. The list of programs shows those that are associated with both tags and the “+” button is then enabled. Press that button and enter a name for the tag in the list above the circles.

You can then drag a copy of the tag from the list to the tag tree so that it becomes a permanent part of your catalogue.

3.5 Entering Commands

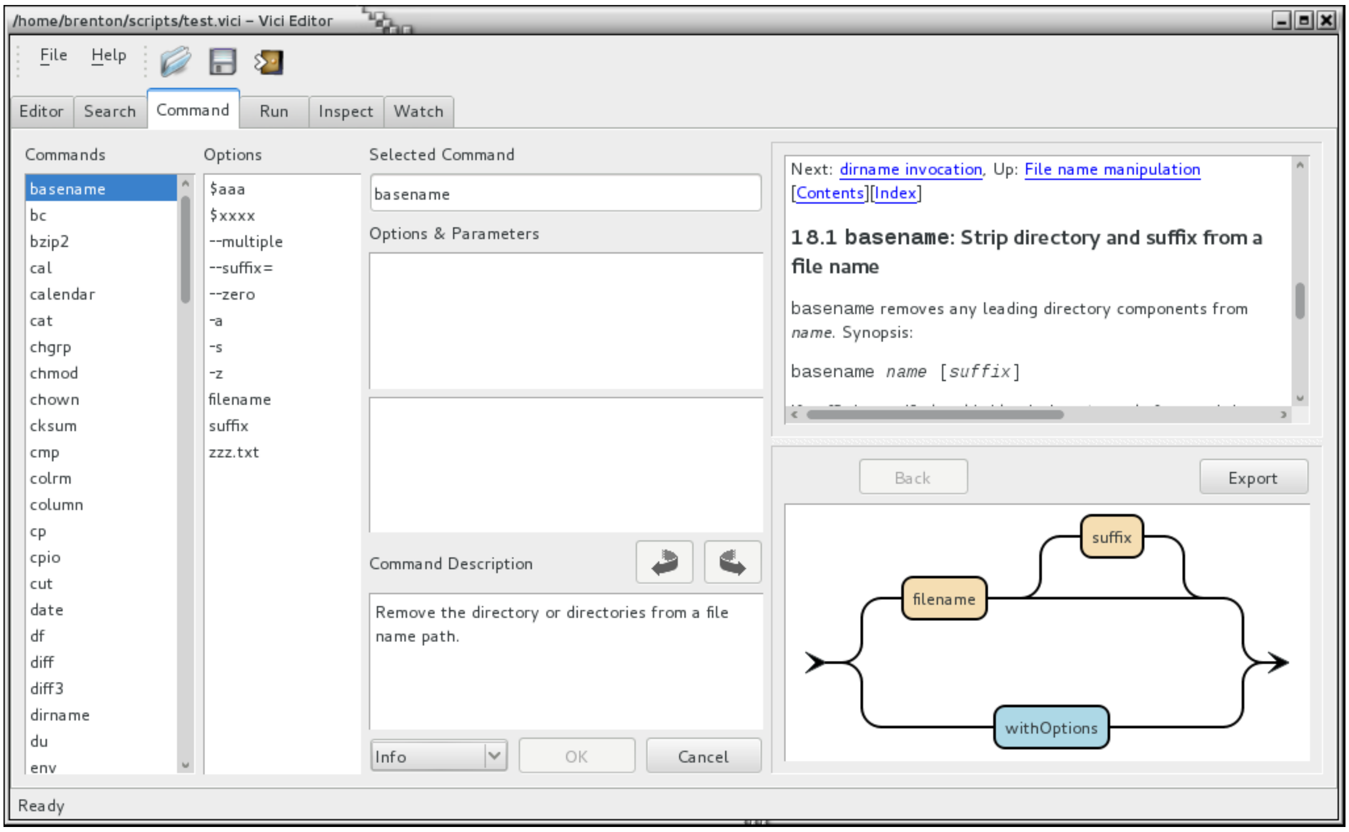
3.5.1 Purpose

Having found a suitable command the next problem is to set up its options and arguments. Unfortunately there is very little standardisation for command line program options either for the values or the format. The command tab attempts to assist the user in setting up the parameters for a command.

VICI will eventually come with a library of prepared commands that provide some additional guidance for the user. The vici-admin program is used to create this library. These commands are listed in the command tab, along with their options and a syntax chart to show the order of the parameters. However, any command can be entered (prepared or not) and if available some help text may also be displayed.

3.5.2 User Interface

The Command entry tab has the following user interface:



Commands: basename, bc, bzip2, cal, calendar, cat, chgrp, chmod, chown, cksum, cmp, colrm, column, cp, cpio, cut, date, df, diff, diff3, dirname, du, env

Options: $aaa, $xxxx, --multiple, --suffix=, --zero, -a, -s, -z, filename, suffix, zzz.txt

Selected Command: basename

Command Description: Remove the directory or directories from a file name path.

Next: dirname invocation, Up: File name manipulation
[Contents][Index]

18.1 basename: Strip directory and suffix from a file name

basename removes any leading directory components from *name*. Synopsis:

basename *name* [*suffix*]

Commands

At the left is a list of the prepared commands. Selecting one of these will display the available options and a syntax chart.

Options

This column shows the options for prepared commands. It also includes the names of variables and files that you have defined in your script. Double clicking an option will append it to the list of parameters.

Selected Command

This field is filled automatically when you select a prepared command, or if you return to the command tab from a node on the edit tab that already has a command associated with it.

Otherwise you should enter the desired command into this field.

Options & Parameters

Enter the options for the command in this list.

Only put one option on each row of the list. For example for some commands you specify the output file name using -o and the name of the file. You must place the -o on one row, and the file name on the next row.

Below the option entry list is a panel showing the command as it might appear when using a terminal interface. This can be handy if you want to cut and paste the option list.

Back and Forward Arrows

The back and forward arrows provide a means of accessing the options previously set for a command. This can be useful if your script uses the same command with similar options in several places.

Command Description

This provides a short description of the selected command (provided it was a prepared command). The aim is to provide something a bit simpler than the somewhat technical descriptions in the help text.

Help Selector

At the bottom of the centre panel is a combo-box for selecting the type of help data to use. The options are “Man” for showing the manual page (aka man page); “Info” for showing the information pages (these are only available for the core utilities); “Usage” which shows the help built into the program; and “Usage Option” which allows you to specify an alternative parameter for the “Usage” option. The default for “Usage Option” is “--help” but some programs use “-h” or “-?”.

OK and Cancel

The Cancel button simply clears the Command tab. The OK button may be disabled if there is no current node selected in the Edit tab, or if there is a danger of the existing data being accidentally clobbered. Press the OK button to transfer the command and options to the currently selected node on the Edit tab.

Help Text

This window shows the man page, info text, or usage for the selected command. For man pages and info text the display uses HTML, so you can follow links if you want.

Syntax Chart

This diagram attempts to describe the order in which the options should appear for the command. (Not many set up yet.)

3.6 Designing A VICI Flow Chart Script

This section provides some tips and ideas to help you design useful scripts.

3.6.1 A Warning

A script is a powerful tool. It can be used to do many useful things, however, much like a kitchen knife is useful for slicing vegetables but when used carelessly can cut fingers or if used maliciously can cause serious injury, a script has the ability to damage or erase your files or otherwise mess up your computer..

You should never run scripts from untrusted sources.

You should never run VICI scripts as the root user.

3.6.2 Global Variables

The variables and constants within a VICI script are global* and can be referenced from any function. This can be a problem if a function is running as a background job and accesses the same variable as another function. The interpreter automatically protects variables so that they are never updated simultaneously, but unless you use semaphores and mutexes to control things you might not get the results you expect.

* Note: A variable can be flagged to indicate it is a local variable. Such variables are local to thread of control (a function, or the body of a for-each command). See below for a more detailed description of local variables.

3.6.3 Interactive Input

The zenity program can be used to interact with the user while a script is running.

3.7 Testing Your Script

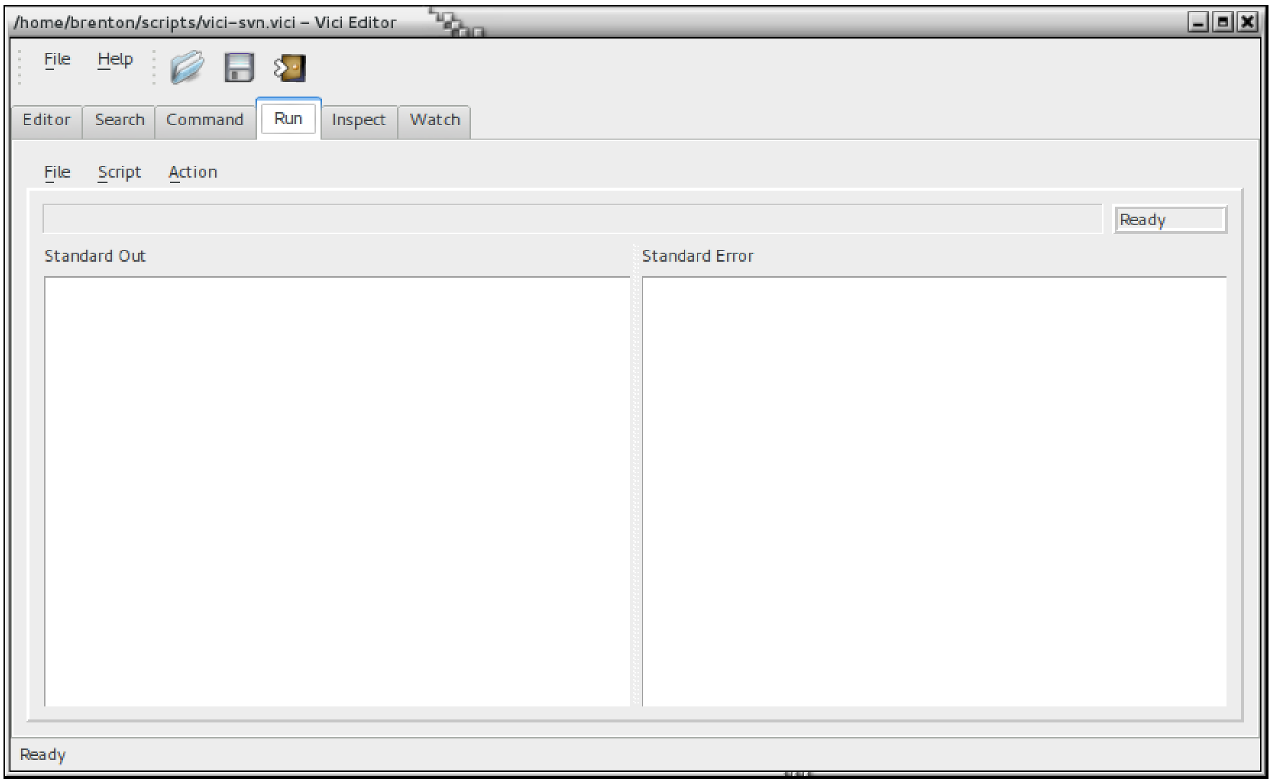
After saving your script it becomes available to run from within the editor. A slightly modified version of the run time program is shown on the "Run" tab. The "Inspect" tab allows you to control the debugging process (See below), and the "Watch" tab shows an animated view of the script with the currently executing command highlighted.

3.7.1 Running the Script

The user interface for the run time program has a menu for changing directory, running the script, clearing the output, and action entries as defined in the script (when you created and named a function).

Under the menu bar is a panel in which variables can be displayed, and a status indicator, showing "Ready" in the image below.

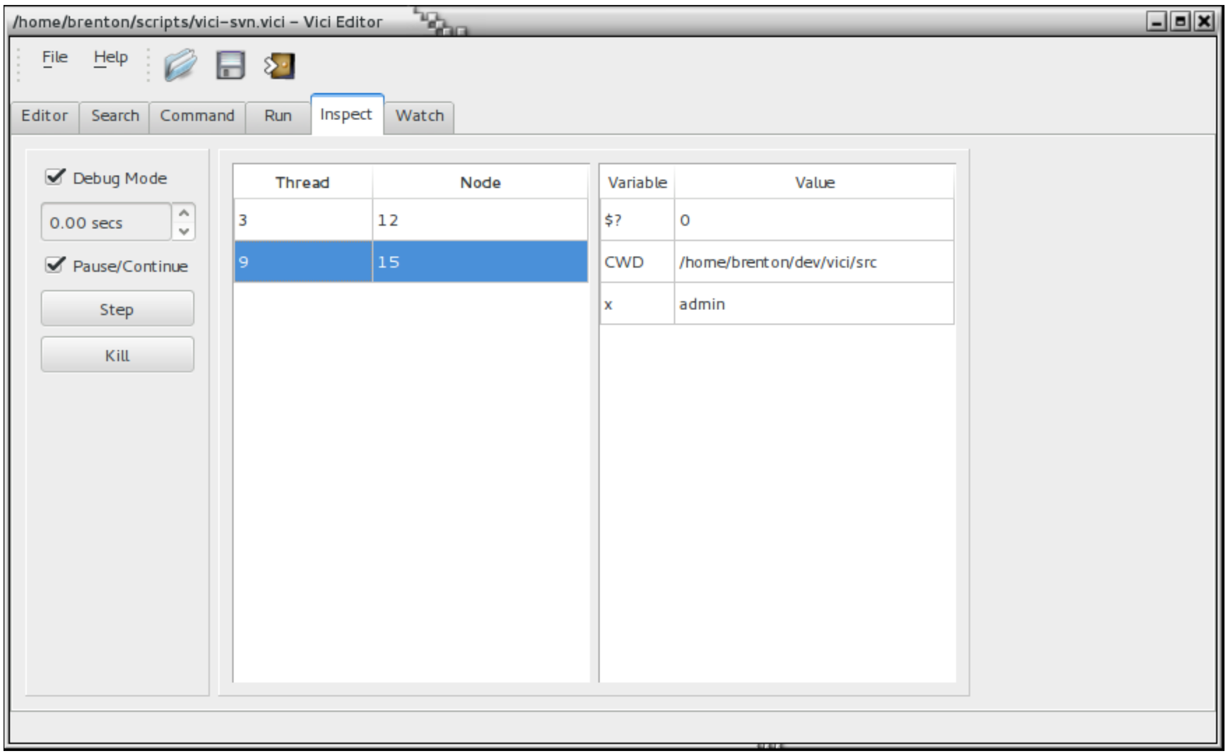
The main area of the window has areas for the output and error streams, plus any displays defined in the script.



When the status indicator shows "Ready" you can then select "Script | Run" or an option under "Action" (depending on your script) to start the script. The status indicator will show "Running" and then either "Success" or "Failed".

3.7.2 Controlling and Inspecting

The “Inspect” tab allows you to control the speed at which the script runs and to inspect the variables.



✓ Debug Mode

0.00 secs

✓ Pause/Continue

Thread	Node
3	12
9	15

Variable	Value
$?	0
CWD	/home/brenton/dev/vici/src
x	admin

In order to enable debugging select “Debug Mode”. Without this selection the script will run as normal.

The delay between running each program in the script can be set in the spin box.

The script can be paused or resumed using the “Pause/Continue” button.

When the script is paused you can select a thread. This enables the “Step” button. Pressing this button allows the selected thread to progress by a single command.

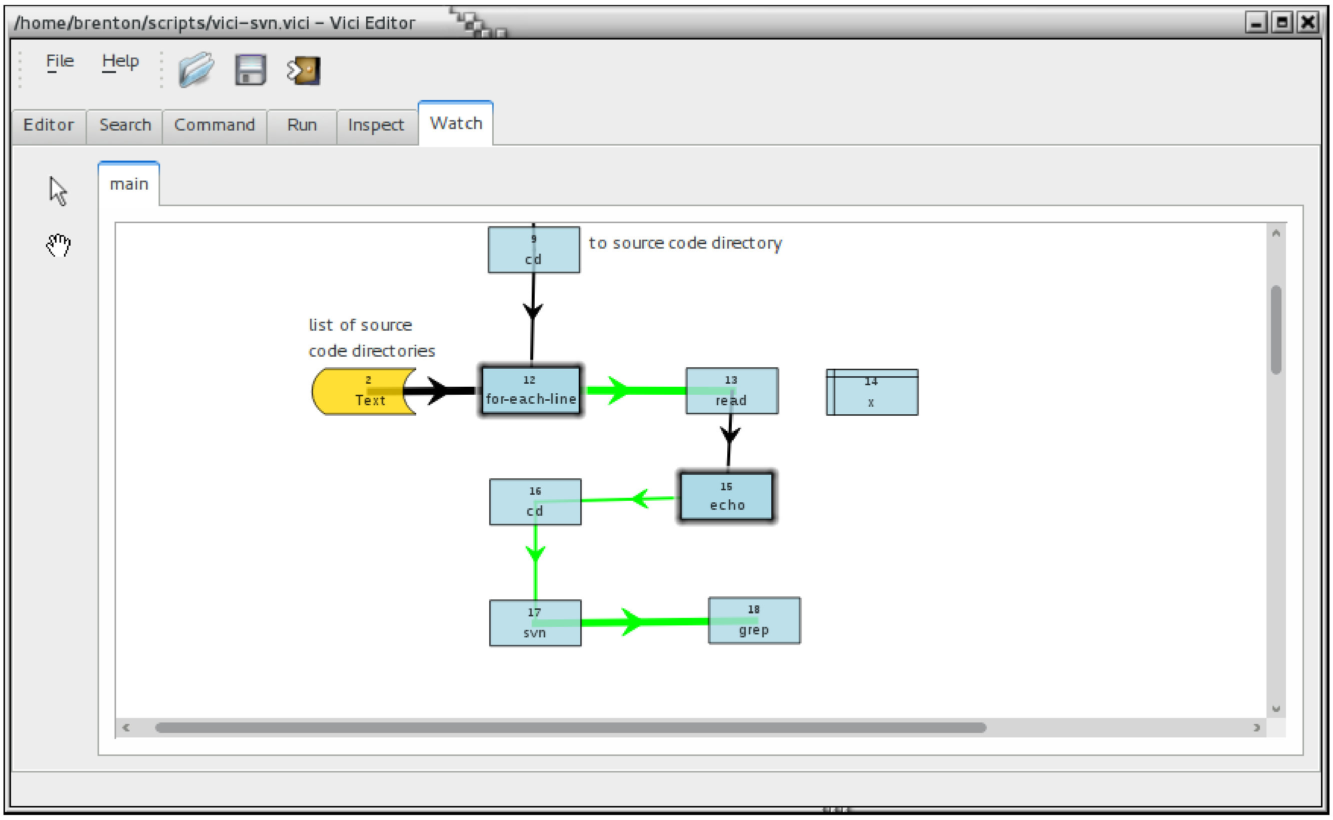
The “Kill” button abandons execution of the script.

The “Thread | Node” list shows the execution point of each thread. In the above display thread #3 is waiting at node #12 while thread #9 is about to execute node #15. The node number is displayed at the top of the symbol.

The “Variable | Value” list shows the exit code of the last command to run ($?), the working directory of the script (CWD) and the current value of the variable named “x”.

3.7.3 Watching the Script

You can monitor the progress of the script from the “Watch” tab.

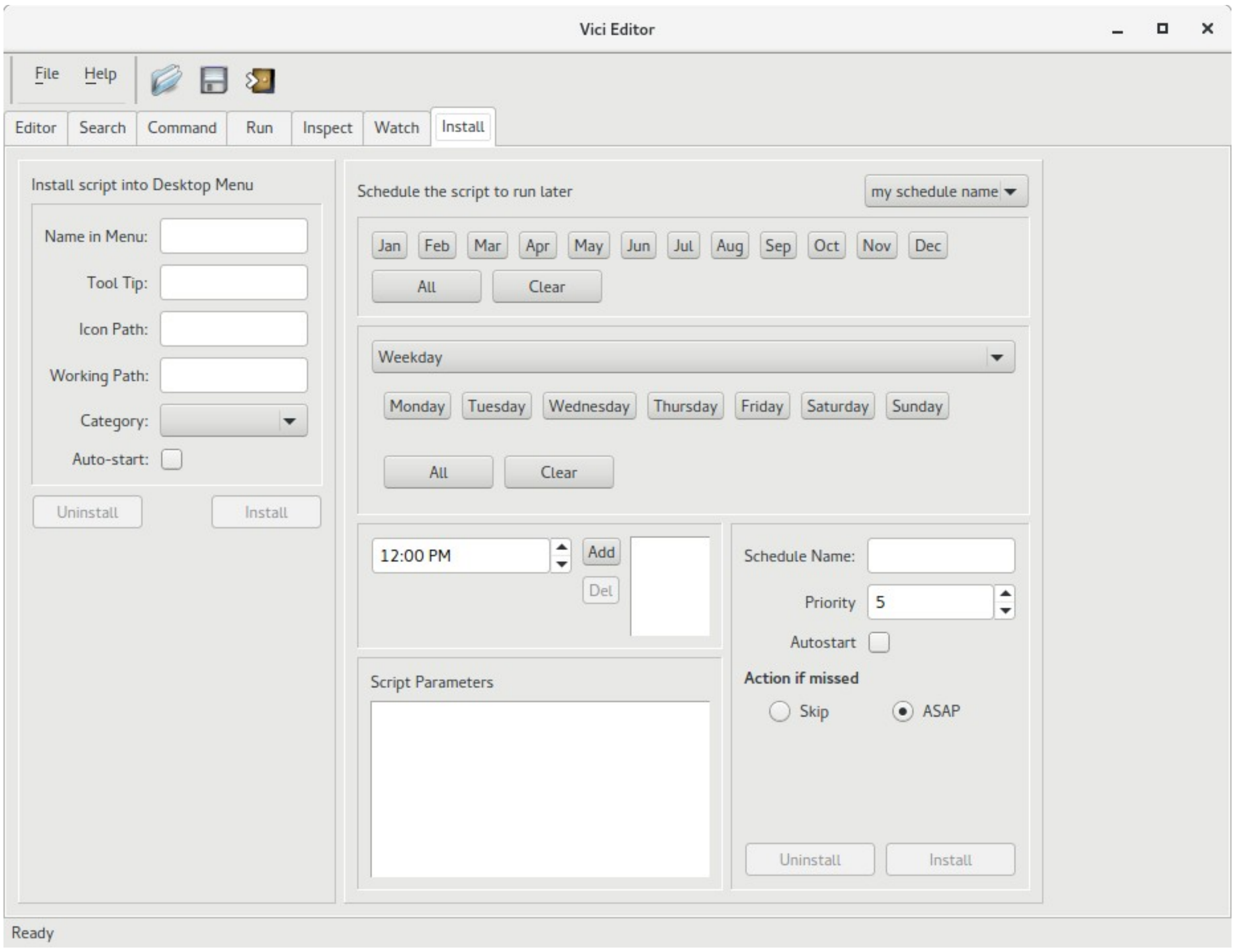


In the above image you can see the scripts threads are waiting at nodes 12 and 15 (as described in the previous section).

(Release 0.12 has a problem when two threads arrive at the same node. When one thread moves on the node is no longer highlighted even though the other thread is still waiting there. This will be fixed in a future release.)

3.8 Installing Your Script

You can install your script into the desktop menu.



my schedule name

Weekday

12:00 PM

Priority 5

Action if missed: Skip [✓ ASAP]

Select the Install tab once your script has been loaded. Installation is done using the left panel, titled "Install script into Desktop Menu".

Enter the name that you want for your script in the menu. This should be different from other items in the menu.

Enter a tool tip for the menu entry. (This might not be displayed in some systems.)

Enter a path to the icon you want to use. PNG and SVG files work best. The default is for the VICI icon.

Enter the path that you want the script to start in. This defaults to your home directory. (You can also change the working path using a menu in the VICI program.)

Select a category for your script. This will determine where you script is placed in the menu, but the details are dependent on your desktop environment.

If your script begins from “main” then you can have it run as soon as the script is opened by selecting the Auto-start check box.

Press the Install button. This will install the current script into the menu system by creating a desktop file called vici-<script-name>.desktop.

3.9 Scheduling Your Script

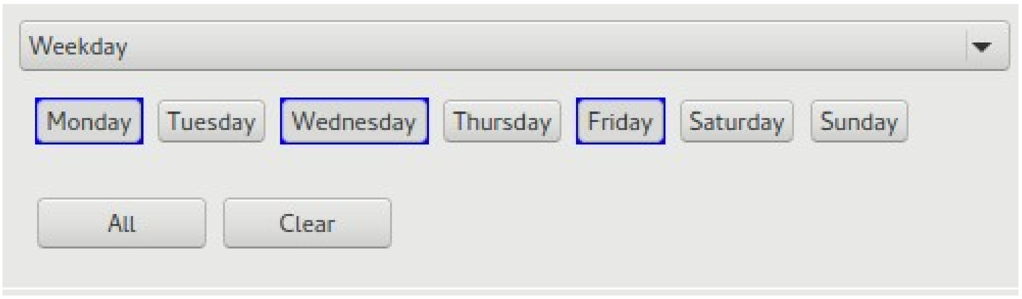
You can have your script automatically open, and optionally run, at set times using the right hand panel of the Install tab (see the previous sub-section).

At the top right is a drop down menu listing the schedules that have been set up for this script. You can have as many different schedules for a script as you want. Before you install any schedule it just reads “my schedule name”.

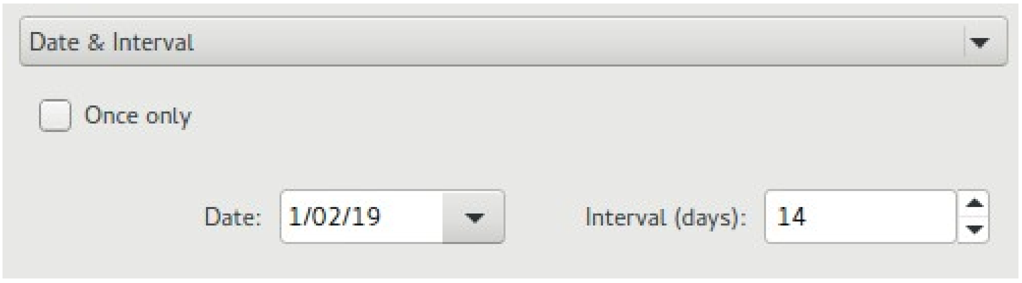
At the top are a set of month buttons. These determine which months the schedule applies to (except for Date & Interval – see below).

A drop down menu is used to select the way days are selected for each month:

Weekday: This option provides a set of buttons so you can select which days of the week the script should be run. The example shown is for running the script every Monday, Wednesday and Friday.



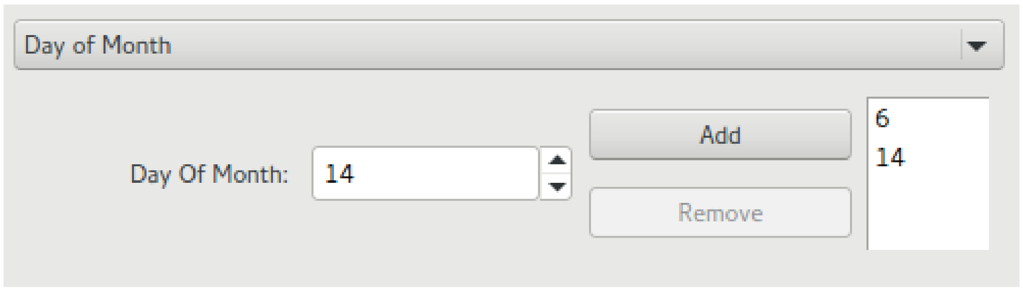
Date & Interval: This option allows you to select a start date and, optionally, an interval (in days) that the script should be run. This would be useful for running a script fortnightly. If you check the “Once only” check box the script will be run on the day selected and then the schedule will be removed.



Date: 1/02/19

Interval (days): 14

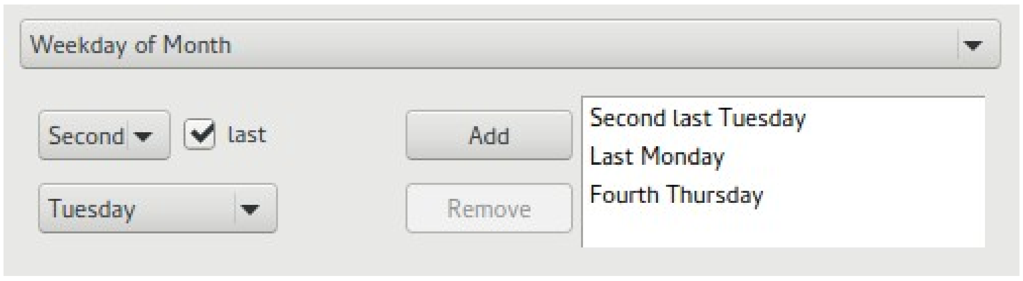
Day of Month: This option allows you to run a script on one or more dates for the selected months. The example shows a schedule for running a script on the 6th and 14th of each month.



Day Of Month: 14

6
14

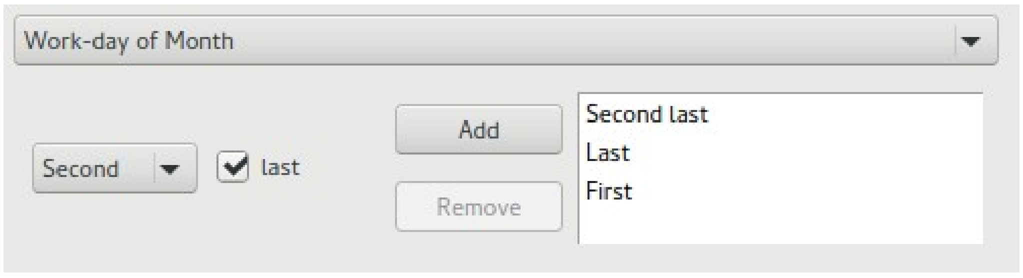
Weekday of Month: This option allows you run a script on particular weekdays each month. The example shows a script set to run on the last Monday, the second last Tuesday, and the fourth Thursday of each select month.



Second ✓ last
Tuesday

Second last Tuesday
Last Monday
Fourth Thursday

Work-day of Month: This option is useful for running scripts near the ends of each month. The example shows a script set to run on the first, last, and second last working day of the selected months. A working day is Monday to Friday.



Second ✓ last

Second last
Last
First

Times: For each selected day you need to enter one or more times that the script will be run.

Schedule Name: Each schedule has to be given a name. These only need to be unique for each script.

Priority: This determines the order that scripts will be started if they should have the same time set. (In this version of VICI all scripts are started at their defined time, but in the future there may be a delay defined according to the priority.)

Autostart: If your script starts from “main” then you can check this box to have the script started as soon as its been loaded.

Action if Missed: The options are to either skip it or run it as soon as possible. This applies when the user has not been logged in (or the system is suspended) and scripts were due to be run during that time.

The schedule can be installed or uninstalled using the corresponding buttons. After being installed it will appear in the drop down menu at the top right of the panel.

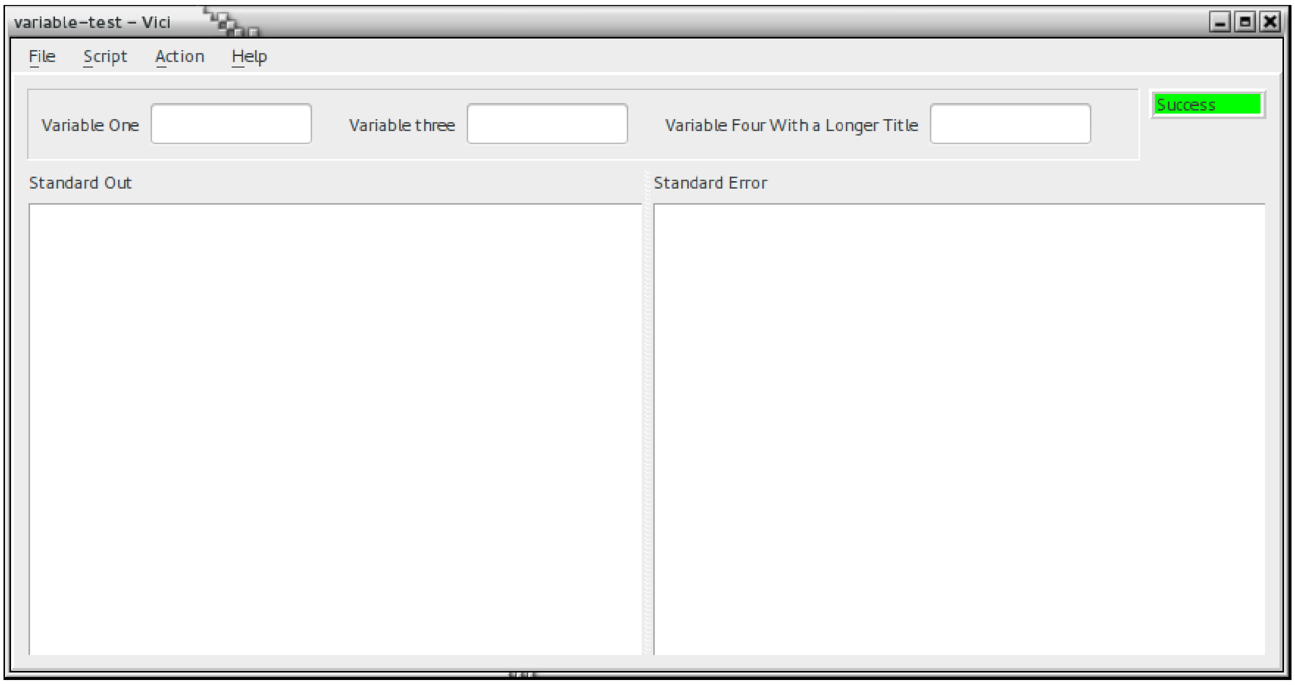
4 Run Time Tutorial

4.1 *Vici Program*

The vici program is used to run scripts that have been created using the vici-editor program.

4.1.1 User Interface

The program provides a set of menus for loading, running and controlling scripts, a panel in which variables can be displayed, an indicator showing the status, and (after a script has been loaded) at least two panels that show the output of the script. There may be additional panels if the script has Display objects. The panels can be resized or hidden using the separating splitter bar(s).



At the top is a panel for variables. You can set values into these before running a script, and they are retained once the script terminates. This can be used to pass parameters into the script, such as file names.

(You can drag a file name into a variable from a file manager, but be aware that you might get a URL such as "file://home/fred/myfile.txt" rather than just "myfile.txt".)

The status indicator starts out with "No Script" and changes to "Ready" when a script has been loaded. It shows "Running" once the script has been started and until the script completes. After it will either turn green and show "Success" or turn red and show "Fail" with the exit code from the script.

The left panel shows the normal output from the script (often called standard out, or stdout). The next panel to the right shows error messages from the script (often called standard error, or stderr).

Additional panels will be shown for each Display object in the script that has been loaded.

(There is no support for standard input in vici. You can use zenity .)

4.1.2 Menus

The menu options are:

File | Open	Opens a file selection dialog so a script to load can be selected. The scripts are files with a suffix of .vici that were created using vici-editor.
File | Change Dir	Change the directory in which the script will be run.
File | Exit	End the program.
Script | Run	Begin executing the script.
Script | Pause	Long running scripts can be paused.
Script | Resume	Resume a paused script.
Script | Stop	Force a script to end.
Script | Clear	Clear the displays.
Action | ?	This menu has an entry for each function in the script that was given a Menu name. It starts the script running from the corresponding function.
Help | Qt About	Qt copyrights
Help | About	VICI copyrights and a link to the home page.

4.2 Running a Script

Start the vici program, and use File | Open to load the script.

Alternatively, select the script in your file manager, and use its context menu to open the script in the vici program.

The scripts can also be opened from a terminal command line by simply entering the name like any other command.

Select Script | Run to begin the script.

Alternatively the script can be started using an Action menu option.

Observe the output in the panels and the exit status once it completes.

You can use File | Open to load a second script. This will replace the currently loaded script and revert the status to “Ready”.

4.3 Run on Command Line

VICI scripts can be run from the command line of a terminal session using the vici-cli program. This allows them to be run as part of a bash script or scheduled using cron or even as a command in another VICI script.

Scripts that are intended for use on the command line should avoid using the facility for attaching a menu entry to a function as there is no menu available on a command line program and should not set the Display Name for variables. They should also avoid using any additional Display objects as the output would be lost.

A script that is run from the command line can read from standard input and should have a single command that is responsible for this read operation.

The script can also have parameters which are referenced within the script by $1, $2, etc.

For example

```
vici-cli script.vici something “something else”
```

would run a script called script.vici with $1 = “something” and $2 = “something else”

5 Reference

This section is the reference for the objects that make up a VICI script.

5.1 Functions

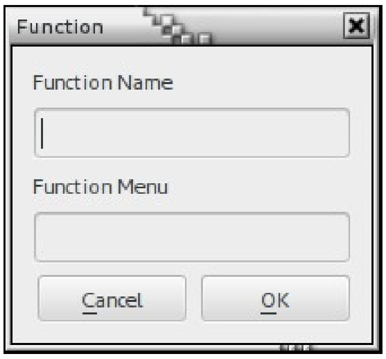
A VICI script is a collection of functions. Normally you would use a separate page (tab) in vici-editor for each function and the function name will be displayed in the tab title. (It is possible to place several functions on one page but this is not recommended.) Click the “New Function” button on the left side of the canvas to create a new function on a new page.

Each function begins its thread of control at the function symbol which should have a single “flow” line leading to the function's first command.

Each function can be given a menu name. These are listed in the Action menu of the vici program.

The functions can be called much as you would for a command (see below) or can run simultaneously if started from the Action menu.

Properties for a Function:



Function Name:(Required) Enter the name of the function. This will be copied to the tab title in vici-editor. It is the name you will use when calling the function from within the script.

Function Menu: (Optional) This is the name of the function displayed in the Action menu of the vici program.

See also: Function Call and Return, Threads of Control.

5.2 Commands and Choices

The Command is where you tell the script what program to run.

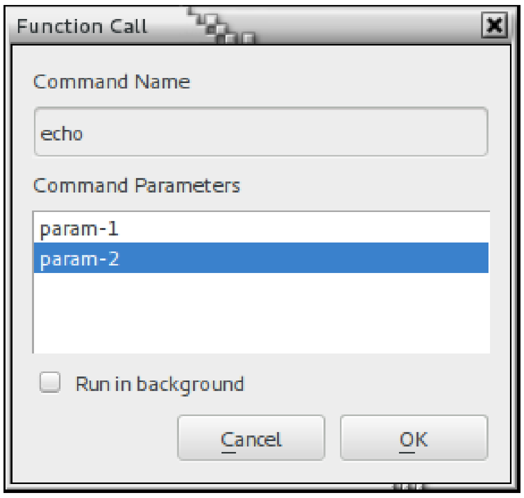
The Choice is functionally identical to the Command but is normally used to indicate that the script is making a decision about what to do next. Typically the program run by a Choice is the “test” program (or the built-in command “switch” - see below).

Each command can be connected to other commands via streams that pass text between the programs. A command has one input stream, one output stream and one error stream (usually for reporting problems).

A command will have an exit code once it completes. Normally this is zero for successful processing or some number to indicate an error.

A command can be run as a background job. In this case the script does not wait until the program completes but immediately moves on to the next command. Hence you should use the “flow” to connect to the next command rather than “success” or “fail”.

Properties for Command:



Command Name: echo

Command Parameters:
param-1
param-2

☐ Run in background

Command Name: (Required) This is name of the program to run. You can use just the program's name if it is on the path, or provide the full path (/usr/bin/echo). It can also be one of the built-in commands described below.

Command Parameters: Place each parameter or option for the program on a separate line. If you try to place two parameters on the same line they will be passed to the program as one parameter which will probably not be understood by the program.

Run in background: Check this option to run the program as a background job.

See Also: Streams, Parameters and Options, Threads of Control, Built-in Commands.

5.3 Parameters and Options

This section describes the various alternatives for the parameters and options that are placed in the Command Parameters list items.

Remember to only put one entry in each list item.

5.3.1 Simple

The simplest parameters are just text strings. These might be options to a program, such as -i or --name, or the name of a file. Some file names have spaces and VICI allows you to just use such a name without having to use quotes.

5.3.2 Quotes

You can place quotes (“,') around a parameter. This is useful if you want to pass an empty parameter to a program or prevent glob expansion (see below).

In this release (0.12) it has some issues when only part of a parameter is quoted. It is recommended that quotes be placed around the entire parameter if they are required.

5.3.3 Escapes

You can use the backslash (\) character to prevent any special effects that a character might have. This useful if you need to include a dollar sign ($) in a parameter.

Note that in this release (0.12) special escape characters (such as tab, new line) are not working as expected.

5.3.4 Variables

The value of a variable or constant can be placed into a parameter by prefixing its name with a dollar sign ($). For example, if you have a variable called “var_1” which has a value “fred”, then by using $var_1 in the parameter the program will get passed “fred”.

The value passed to the program is the value that the variable has at the moment the program begins execution.

5.3.5 Environment Variables

The value of an environment variable can be placed into a parameter by prefixing its name with a dollar sign ($). For example, if you have a variable called “var_1” which has a value “fred”, then by using $var_1 in the parameter the program will get passed “fred”.

The value passed to the program is the value that the variable has at the moment the program begins execution.

5.3.6 Parameters

You can pass parameters to functions, just as you can for a command. To access these from within the function use a numeric parameter such as $1 which would access the first parameter, $2 for the second, and so on.

Similarly you can pass parameters to the vici-cli program and also access them using numeric parameters. (This is not normally available for the GUI since there is no easy way for the user to specify them, however they are available for scheduled scripts.)

5.3.7 Specials

There are a few special symbols that can sometimes be useful in your scripts:

$?	Replaced by the exit code of the previous program.
$$	Replaced by the process number of the vici program.
$!	Replaced by the process number of the previous program.
$#	Replaced by the number of parameters
$*	Replaced by a space separated list of all the parameters.
$0	Replaced by the name of the function.

5.3.8 Indirect

It is possible to use one variable to specify another. For example if the first parameter of a function was the name of a variable then you could access the value of that variable with $($1).

5.3.9 Glob

This is the (rather unfortunately named) term for expanding one parameter into a list of file names that match the expression. Please refer to the man page for glob:

```
man 7 glob
```

In this release (0.12) glob expansion is always performed on the entire parameter unless it enclosed in quotes.

5.4 Function Call and Return

The function call is where you tell the script to run one of the other functions defined in your script.

Each function call runs the called function in a new thread of control (see below).

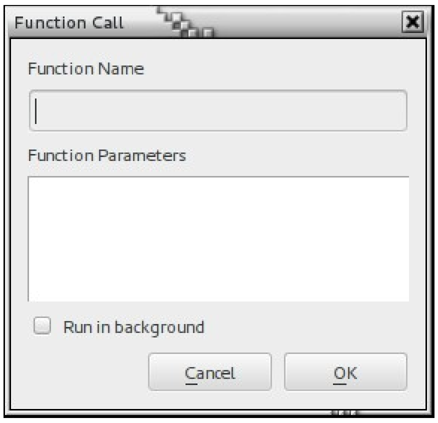
Each function call can be connected to other commands and function calls via streams that pass text between them. A function has one input stream, one output stream and one error stream (usually for reporting problems).

A function will have an exit code once it completes. Normally this is zero for successful processing or some number to indicate an error.

A function can be run as a background job. In this case the script does not wait until the function completes but immediately moves on the the next command. Hence you should use the "flow" to connect to the next command rather than "success" or "fail".

The built-in command "return" can be used in a function to end the processing. This is useful if the function detects a problem and needs to end.

Properties for Function Call:



Function Name: (Required) This is name of the function to run.

Function Parameters: Place each parameter or option for the function on a separate line. If you try to place two parameters on the same line they will be passed to the function as one parameter which will probably not be understood.

The function's parameters may include "--save" and local variable name pairs. This can be used to return results from a function.

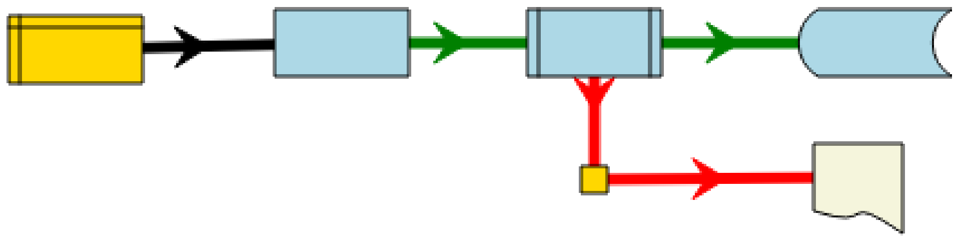
Run in background: Check this option to run the function as a background job.

See Also: Streams, Parameters and Options, Threads of Control, Built-in Commands.

5.5 Streams and Pipelines

In Linux each command line program has three connections (or streams) that it can use to transfer text data. One (standard input) is used for reading input, another (standard output) is used for writing output, and the third (standard error) is for writing error messages. It is therefore possible for the output of one program to be used as the input for another.

A set of programs connected this way is called a pipeline.



You can create these connections in vici-editor by selecting one of the thick lines at the bottom of the symbol palette and using it to connect commands.

VICI also includes the ability to read from functions, variables, constants, files, pipes and manifolds, and to write to functions, variables, files, pipes, manifolds and displays.

The diagram shows a command reading from a constant and sending its output to a function. The output of the function is written to a file while any error messages from the function are presented to the user in a display panel.

In vici-editor the streams are colour coded. Green is used for standard output and red for standard error. Black is used where you want both the output and error streams to go to the next program, but is more commonly used as the input from variables, files, etc.

5.6 Threads of Control

A thread of control defines the order in which commands get run, and based on the exit code of the programs, which commands are run. It is the fundamental control mechanism in a VICI script.

The thread of control begins at the function called “main”, or if you have defined menu names, at the function corresponding to the names in the Action menu of vici.

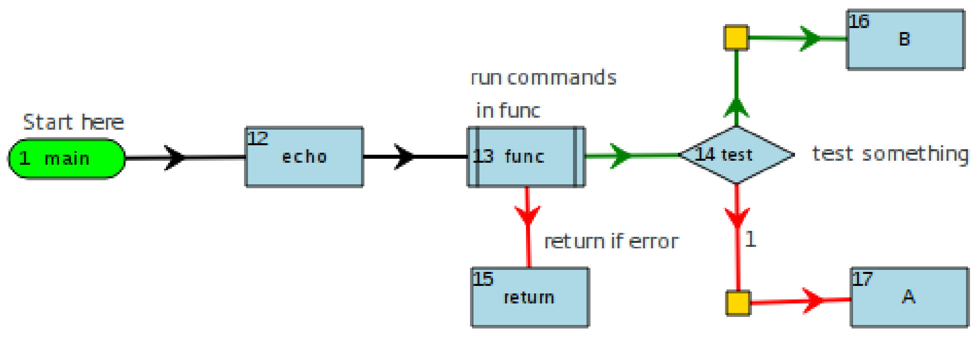
You define the thread of control in vici-editor by connecting the function, commands and function calls using the “flow”, “success” and “fail” lines. These are the thin lines near the bottom of the symbol palette. They are colour coded with black for “flow” (meaning any exit code), green for “success” (meaning an exit code of zero) and red for “fail” (meaning an exit code greater than 0).

A thread of control should be connected to the first command (or function call) of a pipeline.

Since a program may have several different exit codes for different modes of failure you can set an exit code against the “fail” lines. Once the line has been connected, select it (which is shown with a “glow”) with the right mouse button to open the context menu. Select “Exit Code” and set the desired value in the resulting dialog box.

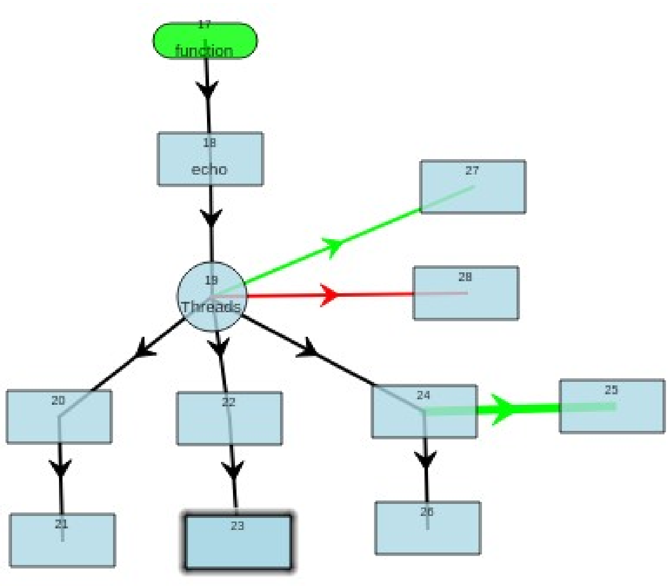
A thread of control may start other threads of control. It can either wait until the thread completes or it can run the threads concurrently. The function call and the for-each built-in programs each start another thread and wait for it to complete. If you flag a command or function call to run in the background it will also start another thread, but in this case it will not wait for it to complete.

A thread of control completes when there is no next command defined that matches the last program's exit code.



5.7 Threads

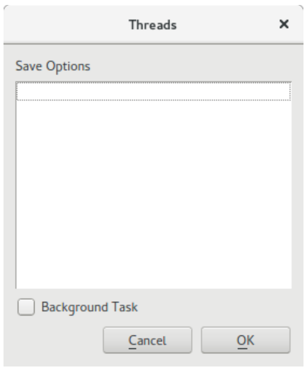
A thread of control can be split into multiple threads of control with the Threads object.



In this example the first thread of control runs the echo command, and then the Threads object. This starts three new threads of control (the black lines) and waits until they have competed. If there are no errors control follows the green line, otherwise the red line defines which command is next.

Normally it is up to you whether you use black or green flow lines, but for the Threads object the black lines define the child threads.

Properties for Threads



The Threads object can have parameters in the form of pairs: --save var-name (one per line, as usual).

The variable being saved should be a local variable. As each thread completes any local variables that were updated by the thread are stored back in the context of the parent thread.

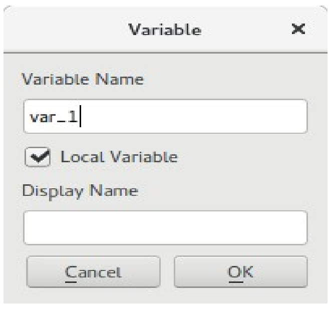
5.8 Variables and Constants

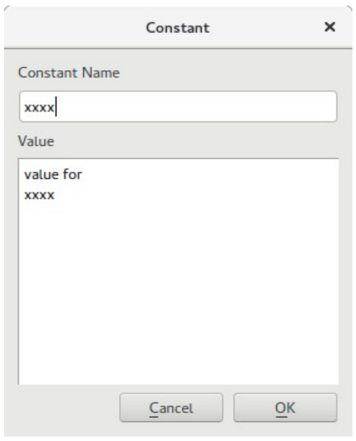
Variables are used in a VICI script to hold data during the running of the program. The script is responsible for setting the values stored in variables.

Constants are similar to variables but the value is set using vici-editor when the script is created.

Both variables and constants have a name and a value. The value is stored as a text string.

Properties for Variables and Constants:





The value of a variable can be set using the read built-in command and the let built-in command for integer numeric values. You can also connect a stream to a variable to set its value.

If the Display Name is set the variable will be displayed on the top panel of the vici program and you can set a value before running your script.

You can access the value when specifying a parameter by prefixing the name with a dollar sign ($). You can also connect a stream to read the value, as shown in the example above for Streams and Pipelines.

The symbol for a variable can appear more than once in a script however it must appear at least once. It is good practice to place symbols for each constant and variable along an edge of the flowchart diagram.

Variables and constants are global in a VICI script. This means that they are accessible from any function in the script. If you are running a function as a background job some care may be necessary to avoid confusion. However, variables are protected so that only one thread of control can access a variable at a time (this avoids corruption of the data value but not the possibility of confusion – please see Mutexes).

Variables are not cleared when a function completes. This allows you to preserve some information for use with different Action menus.

5.9 Local Variables

Variables can be marked as being a “Local Variable”. These are stored with the context of the thread of control. When a new thread of control is created (e.g. by a function call) a copy is made of these local variables and stored in the new context for the new thread of control. Please see the discussion about using threads in the for-each command, the function call and the threads object.

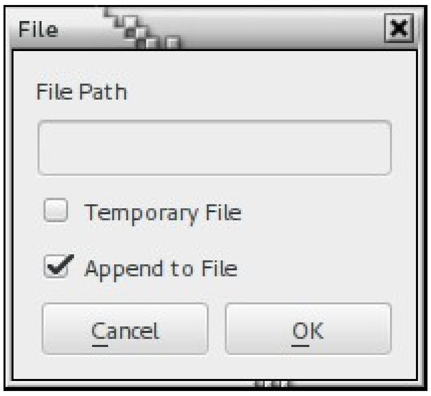
5.10 Files

The files in VICI scripts are expected to be text files. Other types of files must be accessed using commands and passing the file name as a parameter.

Files are used for storing data. Normally you would use a stream to read from or write to a file, but some commands require the file name to be a parameter.

The symbol for a file can appear in more than one place in a script. If the name is the same (after any variable expansion) then it will refer to the same file. Alternatively a single file symbol may refer to many files if its name contains a variable which changes value.

Properties for Files:



File Path: The name of the file. This can either be just the name (e.g. myfile.txt) if it refers to a file in the current working directory, a relative path (e.g. ./mydir/myfile.txt) or an absolute path (e.g. /tmp/myfile.txt)

The File Path may contain variables ($var_1). The actual name is determined at the time the file is opened.

Temporary File: If this is checked the file will be deleted when the script terminates provided that the file was created by this script.

Append to File: If this is checked data will be written to the end of the file after any existing data. If you don't have this checked and you write to the file in more than one place in your script then only the last write will be retained. You can use the truncate command to clear a file.

5.11 Named Pipes

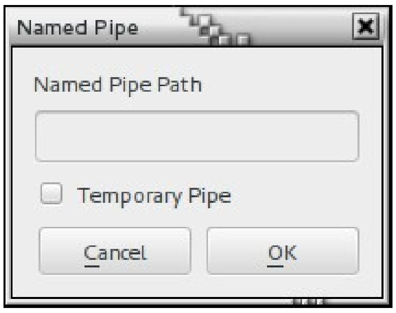
A named pipe is somewhat like a file (it has a file name in a directory) but it does not store anything on the disk, just in a memory buffer.

It can be used to merge several output streams into a single input stream, which is useful since a command only has one input stream.

The difficulty with named pipes is that you have to read and write at the same time. This means that you have to use a background job or a different thread for either the read or write (or both).

The symbol for a pipe can appear in more than one place in a script. If the name is the same (after any variable expansion) then it will refer to the same pipe. Alternatively a single pipe symbol may refer to many pipes if its name contains a variable which changes value.

Properties for Named Pipes:



Named Pipe Path: The name of the pipe. This can either be just the name (e.g. myfile.txt) if it refers to a pipe in the current working directory, a relative path (e.g. ./mydir/myfile.txt) or an absolute path (e.g. /tmp/myfile.txt)

The Named Pipe Path may contain variables ($var_1). The actual name is determined at the time the pipe is opened.

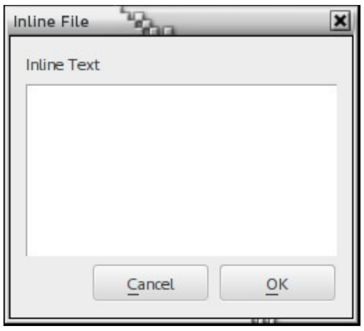
Temporary Pipe: If this is checked the pipe will be deleted when the script terminates provided that the pipe was created by this script.

5.12 Inline Files

Sometimes you may want to have a significant amount of text embedded in the script, for example some boiler plate text that is to be included in other files.

The inline file provides this capability.

Properties for Inline Files:

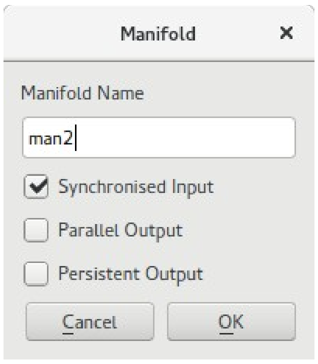


Inline Text: The text to make available to the script. Variables etc. are expanded with the values they have when the inline file is opened.

5.13 Manifolds

A manifold can be used to merge lines of text from different threads, or to distribute them to different threads, or both at once.

Properties for Manifolds:



Manifold Name

man2

[✓] Synchronised Input

[ ] Parallel Output

[ ] Persistent Output

Manifold Name: A name for the manifold. You can place more than one manifold symbol in the script with the same name so that the symbols refer to the same manifold. This is useful for use in functions.

Synchronised Input: If selected the manifold will wait until it has a line from each input before sending the data to the output.

Parallel Output: Normally each input line is passed to the next available output stream in a round-robin fashion. If this option is selected then each line of the input is passed to all the output streams.

Persistent Output: Normally once all the inputs have been closed (e.g. the programs supplying the data have finished) the output streams are then closed. This signals the commands reading the data that they can also terminate.

However, when collecting data from a thread in the body of a for-each command the inputs are closed after each word/line. Selecting this option will prevent the outputs from being closed which allows you to collect all the data from a for-each command.

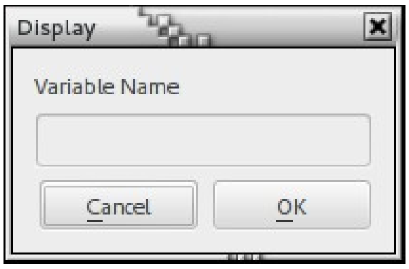
A consequence of using Persistent Output is that your script will not end normally. You will need to either use the “Stop” menu, or include an exit command in the script.

5.14 Displays

A display allows you to provide additional panels in the vici program for displaying results.

The symbol for a display can appear in more than one place in a script.

Properties for Displays:

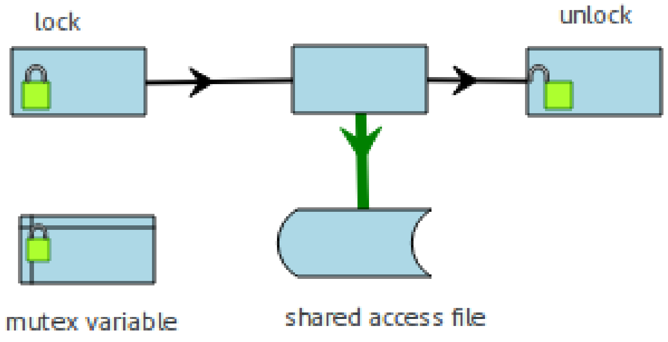


Variable Name: A name for the display. The vici program will have a display panel for each unique name in the script.

5.15 Mutexes

If a function is running as a background job and needs to access a file, variable, pipe or display that is also accessed by the foreground processing (or a different background job) there is the possibility of things getting a bit confused. For example results from the two (or more) threads of control may end up interleaved in undesirable ways.

The mutex variable and associated lock and unlock built-in commands can be used to ensure that only one thread of control can access the items at a time.



Once a thread of control has run the lock command any other thread that tries to run the lock command that references the same mutex variable is forced to wait until the unlock command is run for that mutex variable.

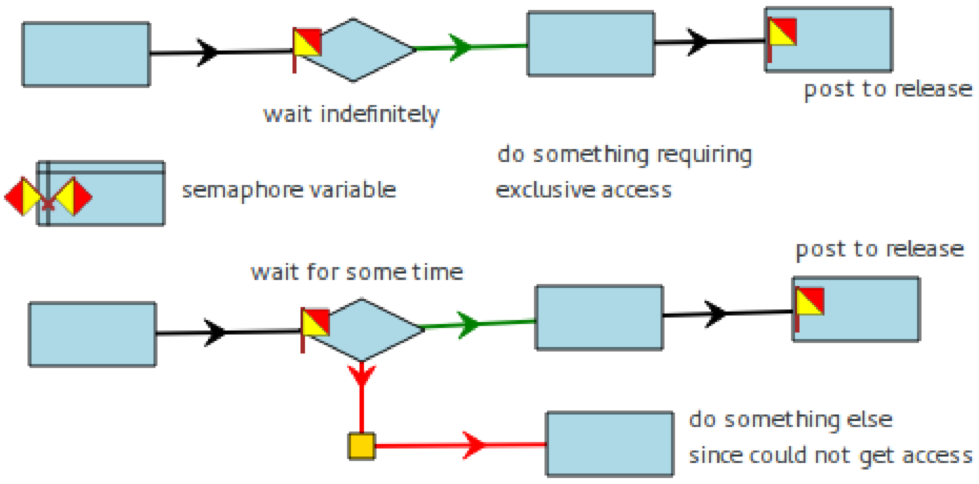
Note that some care is needed when using mutexes. You must ensure that once the lock command is run that there is no path along the thread of control that avoids the unlock command. You might need several unlock commands on separate paths to be sure.

If you have more than one mutex variable that you need to lock then you should ensure that the locks are always taken in the same order. Otherwise you will end up with each thread of control having one lock and waiting for the other – forever.

5.16 Semaphores

If more than one thread of control needs exclusive access to something (file, display, etc) for a period it can use a semaphore to indicate that it has control and that other threads need to either wait or find something else to do.

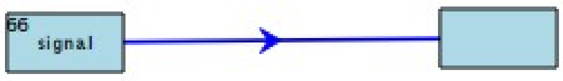
Unlike a mutex the built-in command “wait” can either wait indefinitely, wait for some number of seconds, or give up immediately if the semaphore is in use. If the wait command returns success then the thread has the semaphore. When exclusive access is no longer required it should use the “post” built-in command to allow other threads access.



5.17 Signals

Linux provides the ability to send signals to processes. For example you may have used Ctrl-C to interrupt a program.

Within a VICI script you can send a signal to a program that is running as a background job. However it is limited to commands that are in the same function as the source of the signal. (This is partially due to the difficulty of drawing a line between commands in different functions, and partially due to the difficulty in determining exactly which running instance of the command is to get the signal if the function is being run multiple times simultaneously.)



The built-in command “signal” is used to send the signal.

The parameter can either be a number (refer to documentation on signals and the kill command) or one of the following:

SIGHUP	1	Hangup detected on controlling terminal
SIGINT	2	Interrupt from keyboard
SIGQUIT	3	Quit from keyboard
SIGKILL	9	Kill signal
SIGTERM	15	Termination signal
SIGUSR1	10	User-defined signal 1
SIGUSR2	12	User-defined signal 2
SIGCONT	18	Continue if stopped
SIGSTOP	19	Stop process

5.18 Built-in Commands

Name:	cd
Description:	Change the working directory for subsequent commands in the same thread of control (or one of its children).
Parameter	The path to the directory. This may be an absolute path or a relative path.

Name:	export
Description:	Add or change an environment variable. Without a second parameter the variable is removed. This applies to the commands in the same thread of control (or one of its children)
Parameters	
1	The environment variable name
2 (optional)	The new value for the environment variable.

Name:	let
Description:	Perform a calculation using integer numbers.
Parameter	The expression to evaluate. The expression can be split across multiple parameters, or placed all in one entry since they are gathered together before evaluation. The expression must have the name of a variable to place the result into, followed by an equals sign (=), followed by the remainder of the expression. The expression must use integer numbers, variables (with a $ prefix) and the operators for addition (+), subtraction (-), multiplication (*), division (/), and modulo (%). Parenthesis ( ) can be used to group parts of the expression. The normal arithmetic precedence rules apply.

Name:	for-each
Description:	Read from the input stream and for each word (separated by space characters) start a new thread of control and pass the word to the input stream of its first command.
Parameters	(optional – see discussion below)
1	--maxThreads
2	A number specifying the maximum number of threads to use.

Name:	for-each-line
Description:	Read from the input stream and for each line start a new thread of control and pass the line to the input stream of its first command.
Parameters	(optional – see discussion below)
1	--maxThreads
2	A number specifying the maximum number of threads to use.

Name:	echo
Description:	Collect all of the parameters and write them to the output stream.
Parameters	Any number of parameters may be provided. They will be written out in order.

Name:	dup
Description:	Read from the input stream and for each line write it to both the output stream and the error stream.
Parameters	None.

Name:	read
Description:	Read one line from the input stream and for each word (separated by space characters) save in the variable named in the corresponding parameter. The last variable gets the remainder of the line (if any). If there are insufficient words in the line, the remaining variables are cleared. Commonly used with for-each-line.
Parameters	Any number of variable names. For example with three:
1	The name of the variable to get the first word.
2	The name of the variable to get the second word.
3	The name of the variable to get the remainder of the line.

Name:	switch
Description:	Compare the first parameter with the remaining parameters. The exit code is the number of the first match, or zero if no matches. Note: This command reverses the usual sense of exit codes since zero implies it has failed to find a match.
Parameters	Any number, but usually three or more.
1	The text to compare the other parameters to.
2	The first text to compare. Exit code is 1 if it matches.
3	The second text to compare. Exit code is 2 if it matches.
4	The third text to compare. Exit code is 3 if it matches.

Name:	return
Description:	Ends processing of a function.
Parameter	The exit code for the function. If not specified it will be zero indicating success.

Name:	exit
Description:	Ends processing of the script. (See persistent manifolds.)
Parameter	The exit code for the script. If not specified it will be zero indicating success.

Name:	lock
Description:	Waits until the named mutex is free.
Parameter	The name of a mutex variable.

Name:	unlock
Description:	Free the named mutex so that other threads of control might claim it.
Parameter	The name of a mutex variable.

Name:	wait
Description:	Waits until the named semaphore is free or some time has elapsed.
Parameter	
1	The name of the semaphore variable
2	The time to wait. If negative wait indefinitely to get access, otherwise only wait for the specified number of seconds.

Name	post
Description	Release control of a semaphore
Parameter	The name of the semaphore variable

Name	signal
Description	Send a signal to a background command in the same function.
Parameter	Either a signal number or a signal name (see the reference).

5.18.1 Threads with for-each Commands

The –maxThreads option on for-each and for-each-line can be used to process each word/line simultaneously in separate threads. This is only useful if each word/line can be processed independently and the processing order is not important.

Normally the for-each command creates a new thread of control for each word/line and passes it to the standard input of the first command. It is also normal for this command to be the “read” command that stores the words into variables.

This doesn’t work very well for global variables since the threads would over-write each other. The answer is to use local variables as each thread then stores its own copy.

6 Administration Guide

This section describes how to use the vici-admin program to prepare commands so that they are easier for the user to understand.

The aim of VICI is to make it easy for the average desktop user to use the wealth of command line programs available on a Linux system. Unfortunately the documentation for these commands is mostly in the form of man or info pages, neither of which could be considered particularly friendly forms of documentation.

To attempt to address this the vici-admin program is provided so that a more user friendly description for each command can be constructed, along with a diagrammatic method of showing the parameters and options. The information that is entered into vici-admin is made available to the users of vici-editor when they access the command entry window.

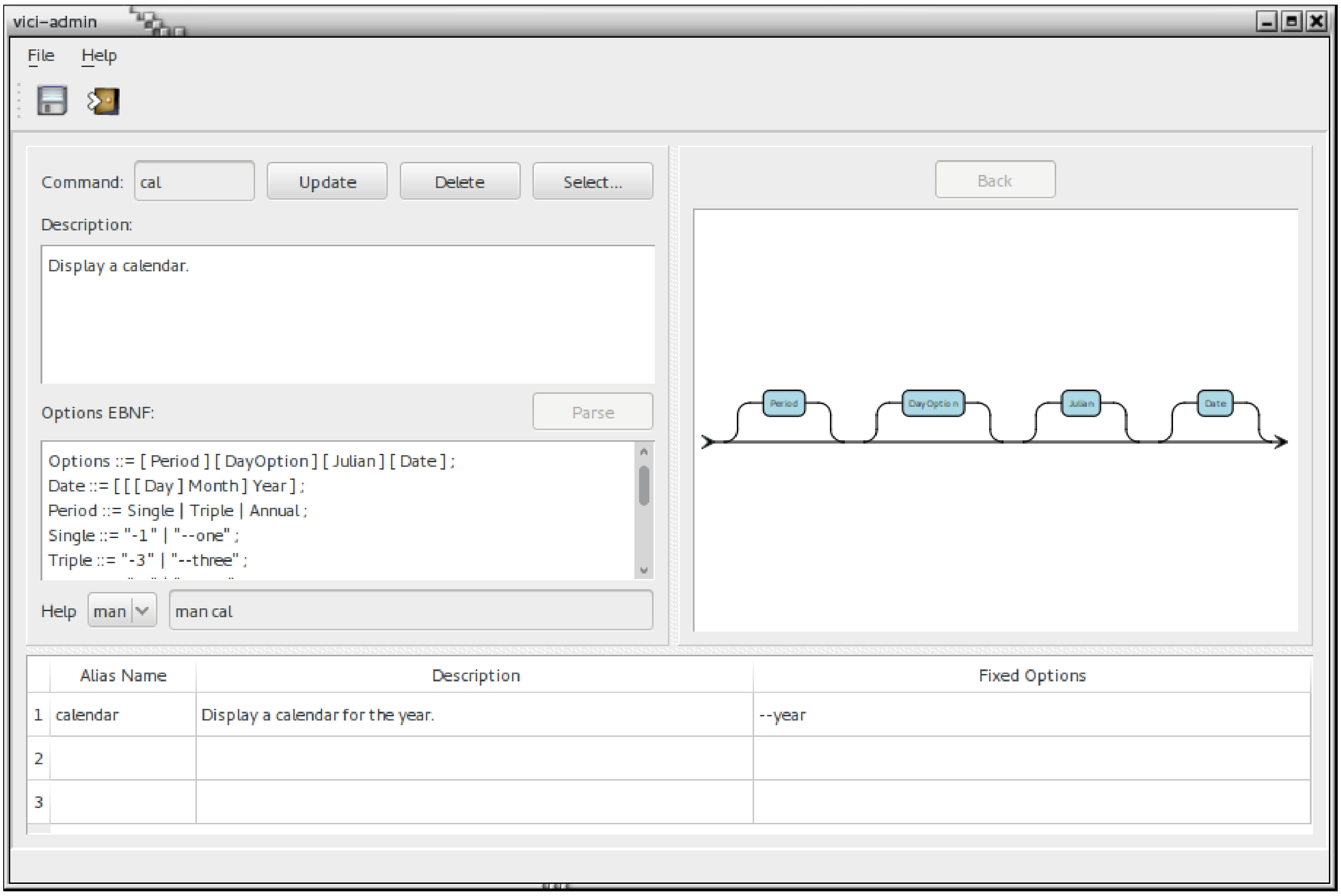
6.1 Command Database

The data entered into vici-admin is stored in one XML data file. The file is specified by the entry in the configuration file with the XPATH of

```
VICI-CONFIG/config/Command/path
```

Please refer to design documentation if you should need the specification for this file.

6.2 The vici-admin Program



Command: cal

Description:

Display a calendar.

Options EBNF:

```
Options ::= [ Period ] [ DayOption ] [ Julian ] [ Date ] ;
Date ::= [ [ [ Day ] Month ] Year ] ;
Period ::= Single | Triple | Annual ;
Single ::= "-1" | "--one" ;
Triple ::= "-3" | "--three" ;
```

Help man man cal

	Alias Name	Description	Fixed Options
1	calendar	Display a calendar for the year.	--year
2			
3			

6.2.1 Internal Structure

The program has three levels of data. There is the data that is presented in the window (as seen above). When the Add/Update button is pressed this data is saved into an internal buffer. When the Save menu or toolbar button is pressed the data is saved to the XML database file.

You can access the internal buffer using the Select button which will allow you to select from the set of prepared commands.

You can use the Delete button to remove an entry from window and the internal buffer. It will be deleted from the XML database when you press Save.

6.2.2 Command

The Command entry field is used to enter the name of the command. If the name entered is already in the set of prepared commands its details will be displayed and the Add button will change to Update.

6.2.3 Description

This is a free form text entry field where you can enter a nice friendly description for the command.

You should try to avoid overly technical terms, and keep in mind the audience for which you are writing.

The description will be searched for words (or phrases) when the user does a search in vici-editor.

6.2.4 Options EBNF

Use this text entry field to enter the syntax of the command's options and parameters.

The first rule must start with

```
Options ::=
```

The left hand side of the remaining rules must be words used in other rules.

All rules must end in a semicolon, ;

The | symbol separates two alternative options.

The [ ] symbols enclose something which is optional.

The { } symbols enclose something which is repeated.

The ( ) symbols allows an entire sequence to be an option.

Explicit values are enclosed in double quotes. A range of values can be shown by including an ellipsis between two explicit values.

The Parse button is used to check the validity of the rules. If it is OK it will be displayed as a syntax chart in the right hand side panel.

Try chmod for an example.

6.2.5 Syntax Chart

The right hand panel shows the current EBNF as a syntax chart. Words that are defined in other rules are shown in blue. These can be pressed to show the chart for that rule.

The back button allows you to return to the previous chart.

6.2.6 Help

The combo box at the left shows the supported types of help documentation. For each one that is available you should enter the complete command to display the help text, such as “man cal” or “info basename”.

6.2.7 Aliases

For each command you can create one or more aliases. These are short cuts that have some of the parameters and options set.

Enter the name for the alias. This should not conflict with other commands, or aliases for other commands. (Support for detecting these collisions is not currently included.)

Enter a description for the alias. This does not have to repeat the description for the command but should make it clear what the alias is for.

Enter the options and parameters that are fixed for the alias.

6.2.8 Save

To save the changes that have been made either press the Save toolbar button, or select Save from the File menu, or the keys Ctrl+s.

6.2.9 Export and Import

In order to facilitate the adoption of these prepared commands you can export one or more to a file that can be sent by email, or other means, to someone who wants them. The recipient can then import the commands.

Commands will be added to the export file without removing those that are already there. You can remove the file if you need to start afresh.

The import will not replace or update commands that are already in your database. You can delete the entries first if want the imported replacements.

7 Installation Guide

7.1 *Installing from tar file*

These instructions are provided as a guide, however, you should check the README file provided with the tar file for the latest information.

The description here is for the straightforward case. Please refer to the README for variations involving Qt libraries, Lua, etc.

7.1.1 Prerequisites

You will need the C/C++ development tools, plus the development versions of the following:

```
libxml2
Qt – Qt6 is currently the supported version.
X libraries  (X11, Xext, Xi and Xt)

You will also need the following program: texi2any
```

7.1.2 Building

As a normal user:

Unpack the tar file.

Change directory to the unpacked directory.

Create a build directory. Note that building from the build directory is recommended but not necessary.

Defining the VICI variable is necessary.

```
$ mkdir build
$ export VICI=$(pwd)
$ cd build
```

Then run configure from the build directory so that the temp files don't clutter up the source. This may take a little longer than normal since each of the sub-projects has its own configure which are run consecutively.

```
$ ../configure --prefix=$VICI
$ make
```

7.1.3 The Qt Libraries

If you have Qt installed in a non-standard location or have more than one version installed you can specify it by giving the path to its package configuration file.

For example if a package file was `/opt/Qt/5.5/gcc_64/lib/pkgconfig/Qt5Core.pc`

you would use the --with-qt-pkg option:

```
--with-qt-pkg=/opt/Qt/5.5/gcc_64/lib/pkgconfig
```

7.1.4 Testing

Confirm that the build has been successful:

Create a directory for the log files:

```
$ mkdir -p $VICI/logs
$ make check
```

(Note that during the check GUI programs should automatically run their tests and exit, and there will be a delay of about ten seconds during one test which confirms a timeout.)

Please see the README file for what to do if the tests should fail.

7.1.5 Installation

VICI will be installed in the directory specified with the prefix option of configure (see above).

During this development phase it is recommended that VICI is not installed in the usual /usr or /usr/local directories, but rather it should be installed in $VICI.

```
$ make install
```

Optionally you can install .desktop files and mime type files. These will install entries in the main system menu and allow scripts to be started from a file manager. It will also install an autostart .desktop file for vici-cron which is needed if you want the scheduling.

```
$ ./desktop.install
```

7.2 Configuration

This section describes how the system should be configured once it has been installed.

7.2.1 Environment

The following environment variables are used by the VICI programs:

VICI

The VICI programs rely on this environment variable to be set to the path that leads to the VICI programs, libraries, and data files. (See the directory structure below.)

VICI_CONF

This (optional) variable can be used to specify the path to the main configuration file.

7.2.2 Directories

The $VICI environment variable should point to the following directory structure. This is normally set up by the “make install” command during installation.

```
.
|-- bin
|-- docs
|-- include
|   `-- vici
|       `-- test
|-- lib
|   `-- vici
|-- libexec
|   `-- vici
|-- logs
|-- share
|   `-- vici
`-- var
    `-- vici
```

The GUI programs save their settings, such as window size and position, in the user's home directory at

```
$HOME/.config/VICI/*.conf
```

7.2.3 Configuration File

The VICI programs get many of their parameters and options from an XML configuration file. The programs search for this file in the following order:

$VICI_CONF

This environment variable, if defined, should specify the path of the configuration file.

./vici.xml

The file vici.xml in the current working directory. (This alternative is mostly used for testing.)

$HOME/.local/share/vici/vici.xml

This alternative allows a user to provide an overriding version for their personal use.

$VICI/src/top/vici.xml

This alternative is used during development.

$VICI/share/vici/vici.xml

This is the standard location for the configuration file.

(The initial version will be configured for testing, but later will be replaced by a version suitable for normal use.)

The following section describes the parameters and options that are set in the configuration file. XPATH notation is used to specify the location in the XML.

VICI-CONFIG/config/ipc/keyfile

This contains the path to a file that is used to create a unique identifier for inter-process communications.

VICI-CONFIG/config/Command/path

This contains the path to the file that contains the prepared commands.

VICI-CONFIG/config/Tags/path

This contains the path to the file that contains the XML for the tags. One will have an attribute user='true' to indicate that it is where the file is saved to. The other are for the original delivered version.

VICI-CONFIG/config/Help/script

The path to a script that is used to convert man pages to HTML

VICI-CONFIG/config/Help/info

The path to the HTML file of core utilities documentation.

VICI-CONFIG/config/Interpreter

The path to the vici program. This is used when creating desktop files.

VICI-CONFIG/config/Schedule/path

VICI-CONFIG/config/Schedule/socket

VICI-CONFIG/config/Schedule/lastRun

Files used by the scheduling component.

VICI-CONFIG/PlugIns/PlugIn

The prog attribute is the name of the program that the plugin is intended for.

VICI-CONFIG/PlugIns/PlugIn/name

The name of the plugin. It is possible to have more than one plugin in a library, so this provides an identifier for it.

VICI-CONFIG/PlugIns/PlugIn/mode

This must be one of “autorun”, “demand”, or “resident”.

A resident mode is the normal operation where the library is loaded and remains resident for the duration of the program.

A demand mode is used for plugins that are loaded when required, and may be unloaded when no longer needed.

An autorun mode is for use with testing where the plugin is loaded and immediately run, then unloaded.

VICI-CONFIG/PlugIns/PlugIn/path

There can be multiple entries for this path to the plugin library. The system will search for the plugin in the order listed.

VICI-CONFIG/LOG/

There is a section here for each log name.

VICI-CONFIG/LOG/*log-name*/type

This specifies the type of logging:

cout, cerr, or clog: Log entries are sent to these standard streams;

file: Log entries are written to a file. The entries are formatted;

unformatted: Log entries are written to a file without formatting;

trace: Log entries are written to a file formatted for use by the trace processing;

syslog: Log entries are sent to the system logging;

udp: Log entries are sent to a logging server, such as vici-logger.

VICI-CONFIG/LOG/*log-name*/file

This specifies the name of the log file.

VICI-CONFIG/LOG/*log-name*/ident

This is the identifier for a syslog entry.

VICI-CONFIG/LOG/*log-name*/host

This is the host name for a logging server.

VICI-CONFIG/LOG/*log-name*/port

This is the port number for a logging server.

VICI-CONFIG/Test/

There is an entry here for each test.

VICI-CONFIG/Test/*test-name*/Report/name

The name of the log that the test's report will be written to. The log should be of type “unformatted”.

VICI-CONFIG/Test/test-name/script

The path of the XML test script as used by the GUI Test Harness