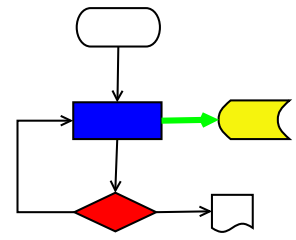


VICI



VISUAL CHART INTERPRETER Requirements

Publication History

Date	Who	What Changes
10 September 2012	Brenton Ross	Initial version.
2 November 2014	Brenton Ross	Addition of increment plan references.
18 July 2016	Brenton Ross	Additional ideas in Appendix D



Copyright © 2009 - 2014 Brenton Ross
This work is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License.
The software is released under the terms of the GNU General Public License version 3.

Table of Contents

1 Introduction.....	4
1.1 Scope.....	4
1.2 Overview.....	4
1.3 Audience.....	4
2 Functional Requirements.....	5
3 Environmental Requirements.....	11
4 Quality Requirements.....	12
4.1 Computational Requirements.....	12
4.2 Deployability Requirements.....	13
4.3 Software Text Requirements.....	13
4.4 Software Specification Requirements.....	14
4.5 Development Process Requirements.....	14
Appendix A - Commands.....	15
Appendix B – EBNF.....	22
Appendix C – Symbols.....	23
Appendix D – Ideas and Proposals.....	26

1 Introduction

This document is the list of requirements for the Visual Chart Interpreter (VICI) project.

1.1 Scope

The document lists the functional, environmental and quality requirements.

1.2 Overview

The requirements form the foundations upon which the entire development process is built.

The architecture process will select a set of tactics which will transform these requirements into a set of responsibilities. These responsibilities will then be assigned to modules and implemented to create the desired system.

The requirements each have a priority:

1	Mandatory requirement that must be provided.
2	Very important requirement.
3	Important requirement.
4	Nice to have or optional requirement.
5	Unimportant requirement, but which would not be detrimental.

Each requirement is assigned to a build increment which are defined in the Increment Plan.

1.3 Audience

This document is intended to be used by the developers of the system, and anyone else who needs to understand what the requirements were that drove the design process.

2 Functional Requirements

These are the requirements that define what the system must be able to do.

Identifier	Priority/ Increment	Requirement
R1	2 / #4	Provide a means of preparing a command for use by this system so that it can be more easily used by ordinary users..
R2	2 / #4	The commands listed in Appendix A will be prepared for use with this system.
R3	2 / #3	The system will allow an administrator to specify the EBNF of the syntax for the commands that are supported by this program.
R4	2 / #3	The EBNF used to define the syntax of a command's parameters and options will be that specified in Appendix B
R5	4 / #4	The system will allow an administrator to set up aliases for common commands and some of their parameters.
R6	1 / #6	Provide a means for users to create scripts using the graphical interface.
R7	1 / #8	The system will allow the user to select from the set of prepared commands.
R8	1 / #6	The system will allow the user to enter any command and its parameters, not just those that have been prepared for this system.
R9	3 / #8	Provide guidance for common commands, including guidance on the parameters and easy to understand documentation.
R10	1 / #6	The system shall allow the user to enter short options, long options, parameters and name=value pairs as required by the commands.
R11	2 / #8	Guide the user in selecting the parameters for the commands used in their scripts.
R12	2 / #8	The system will allow the user to select the parameters from a palette of parameters and options applicable to the selected command.
R13	3 / #8	The system will hide or disable options and parameters which are inapplicable.
R14	3 / #8	The system will display a syntax chart for the selected command.

Identifier	Priority/ Increment	Requirement
R15	1 / #6	Allow the user to define the processing order within a script by laying out the commands in a flow chart.
R16	2 / #6	Provide a palette of common flowchart symbols.
R17	2 / #6	The visual scripting language (VSL) will use the symbols described in Appendix C.
R18	3 / #6	Allow groups of symbols to be re-positioned while maintaining the links between symbols.
R19	3	Allow a group of symbols to be promoted into being a separate function.
R20	4	Provide an option that attempts automatic layout of the flowchart diagram.
R21	2 / #6	Allow the user to insert comments into the flowchart.
R22	1 / #10	The editor will create desktop files according to the freedesktop.org specification. desktop-entry-spec-1.0.html
R23	1 / #10	The editor will create menu entries for the desktop according to the freedesktop.org specification. menu-spec-1.0.html
R24	1 / #10	The editor will place desktop files and menu entries according to the freedesktop.org specification. basedir-spec-0.6.html
R25	1 / #10	The editor will allow the user to remove a script and its associated desktop files and menu entries.
R26	1 / #10	Allow the user to install their script from the editor in which it being created.
R27	1 / #5	Provide a means of allowing a user to find commands that satisfy their needs.
R28	2 / #5	The editor will allow the user to use the apropos command in order to search for commands to use.
R29	2 / #5	The editor will allow the user to search the descriptions of the prepared commands.
R30	2 / #8	The editor shall allow the user to view the man page for selected commands.
R31	2 / #8	The editor shall allow the user to view the info page for selected commands.
R32	3 / #5	Allow the user to associate tag identifiers with the commands.

Identifier	Priority/ Increment	Requirement
R33	3 / #5	Provide an initial set of tags for commands.
R34	3 / #5	Allow the user to select commands using tags.
R35	3 / #5	Allow the user to build set expressions for the tags, including union and intersection.
R36	3 / #5	The editor will allow the user to classify commands by associating one or more tags with each one.
R37	4 / #5	The search tags will be able to be classified into an hierarchy.
R38	2 / #9	Allow a user to test their script before installing it.
R39	2 / #9	Allow the user to step through the script one command at a time with a visual indication of what is happening.
R40	3 / #9	The VSL will allow the user to run a section of a script by defining start and break points.
R41	3 / #9	The VSL will allow the user to run all or part of a script in slow motion by inserting a configurable amount of sleep between each command.
R42	3 / #9	Allow a user to observe the internal working of a script while testing it.
R43	3 / #9	The VSL will allow the user to save and restore the values of variables and the state of the call stacks so that sections of a script can be re-run.
R44	3 / #9	The VSL will allow the user to edit the values of variables while the system is paused.
R45	3 / #9	The system will allow the user to observe the values of variables when run from within the editor.
R46	3 / #9	The system will clearly indicate which commands are about to be processed, or are being processed, and which variables are being referenced during testing.
R47	3 / #9	The system will show the stack variables for the thread that is currently being displayed.
R48	4 / #9	The system will allow the user to observe the content of selected files while being run from with the editor. An option to view the tail of the file shall be provided.
R49	4 / #9	The system will allow the user to monitor the size of files and other properties during testing.

Identifier	Priority/ Increment	Requirement
R50	4 / #9	The system will allow the user to select and display each thread separately.
R51	4 / #9	The system will allow the user to take a snapshot of a file and restore it as necessary for their testing.
R52	3 / #	The VSL will allow the user to define the colours, patterns and textures used for the background of commands and lines to best suit their vision and cultural requirements.
R53	4 / #	Provide the user with a configurable editor that allows sub-windows to be sized and positioned.
R54	4 / #	Automatically configure sub-windows according to the size and shape of the display screen.
R55	2 / #	The VSL will include the ability to group commands and split them out into a separate function which can be reused.
R56	2 / #6,7	The VSL will allow the user to attach a function to a menu item so that it can be executed independently.
R57	3 / #6,7	The VSL will allow a function to be run as a background task.
R58	1 / #6,7	The VSL will allow the user to determine which command to execute next based on the exit status of the previous command: either unconditionally, on success, on fail, or on a particular exit status value.
R59	1 / #6,7	The VSL will allow the user to connect commands using a pipeline. Connections may be from either stdout, stderr, or both.
R60	1 / #6,7	The VSL will include a built in command for changing the current working directory of the script.
R61	1 / #6,7	The VSL will include a built in command that passes one line of input at a time to its output pipe. This will allow the user to build a loop structure that process each line of input.
R62	1 / #6,7	The VSL will include a built in command that sets one or more parameters to the values on its input pipe. This will allow the user to build a loop structure that processes variables separated by white space.
R63	2 / #6,7	The VSL will allow commands to be run in the background. Such commands will be terminated when the script completes.

Identifier	Priority/ Increment	Requirement
R64	3 / #6,7	The VSL will allow the user to connect the output of a command to a named pipe.
R65	3 / #6,7	The VSL will allow the user to send a signal to a background command.
R66	1 / #6,7	The VSL will allow the user to connect the output of a command to a file, and may specify over-write or append modes.
R67	1 / #6,7	The VSL will include references to files.
R68	2 / #6,7	The VSL will include the equivalent of the in-line file. The in-line file may contain references to variables that will be expanded when the file is referenced.
R69	3 / #6	The VSL will allow the user to specify a file as being temporary, and such files will be automatically removed when the script terminates.
R70	1	The system must prevent the user from running random scripts. They should only be allowed to run scripts they have constructed themselves, or those provided by the system administrator.
R71	1 / #6,7	The VSL will allow the user to store the output of a command into a variable.
R72	1 / #6,7	The VSL will allow the user to define variables and constant values which can be referenced as (parts of) parameters to commands by preceding their name with a \$ sign.
R73	2 / #6,7	The VSL will allow the user to perform simple arithmetic on numeric variables. One of +, -, *, / and % can be specified as the operator.
R74	3 / #6,7	The VSL will include built in variables containing the process id of each command started in background mode.
R75	3 / #6,7	The VSL will include a built in variable containing the exit status of the last command to complete.
R76	3 / #6,7	The VSL will include built in variables that allow a function to have parameters. These will be numeric indexes into the parameters placed on the call stack.
R77	3 / #	The VSL will enable the user to define variables that will take the value dropped onto a text entry field in the runtime. The runtime will display a text entry field for each of these variables.

Identifier	Priority/ Increment	Requirement
R78	4 / #6,7	The VSL will enable the user to define mutex variables that can be used to synchronise the processing of two or more threads of control within the script.
R79	4 / #6,7	The VSL will enable the user to define a semaphore variable and a signalling command that can be used to increment the semaphore.
R80	1 / #10	Provide a means for users to run scripts from the graphical desktop.
R81	1 / #7	Allow the user to navigate the script to some folder.
R82	1 / #7	Allow scripts to be triggered from events such as program start, drag-n-drop, or menu action.
R83	2 / #7	Allow the user to terminate any running script from a menu option.
R84	2 / #7	The runtime shall include menu entries for any script function flagged as being attached to a menu.
R85	2 / #7	The runtime may provide text areas for accepting user input and/or displaying stdout and stderr if the script specifies them.
R86	3 / #	Allow the user to launch their script by dragging and dropping a file name onto the script's desktop window.
R87	3 / #	Allow the user to open a file browser using a menu in the scripts desktop window.
R88	3 / #7	The runtime shall include options to stop and to restart a script.
R89	4 / #11	Provide a means of automatically launching a script at a prescribed time.
R90	2 / #4	Allow the user to export one or more prepared commands.
R91	2 / #4	Allow the user to import one or more prepared commands.

3 Environmental Requirements

These are the requirements that are determined by the environment in which the system must operate.

Identifier	Priority	Requirement
RE1	1	The system must run on any Linux platform that supports the freedesktop.org specification.
RE2	1	The system must use open source, GPL components (or those which have no usage restrictions).

4 Quality Requirements

These are also known as the non-functional requirements. They include a lot of requirements that have the suffix “-ability”. These requirements are often the most important for driving the high level design.

4.1 Computational Requirements

These are the quality requirements that are to be satisfied by the executing programs.

Identifier	Priority	Requirement
RQC1	2	The system must record in the script the user name of anyone who edits it.
RQC2	2	The system must only open or run scripts which have been signed by the user or by one of an administrator controlled list.
RQC3	2	The system's UI components must present an attractive and non-intimidating interface.
RQC4	3	The system should not place any significant extra load on the system beyond what is necessary for the commands being executed.
RQC5	2	The system must provide clear feedback for all actions and error conditions. This must be provided in clear English and include some guidance on what action is required to resolve any problems.
RQC6	1	It must not be possible to use the system to gain access that the user would not normally have.
RQC7	3	It should be easy to demonstrate the system. The system should be packaged with a set of tutorials and guides that make it easy to learn how to use.
RQC8	1	The runtime component must be dependable so that the user can trust that their scripts will be executed correctly and when needed.

4.2 Deployability Requirements

These are requirements that are concerned with deploying the system and administering it.

Identifier	Priority	Requirement
RQD1	2	The system must be easy to administer. It should require little more than installing for it to be usable.
RQD2	3	The system must allow an administrator to promote a user's script for site wide use.
RQD3	2	The editor should automatically backup scripts and allow the user to easily revert to a previous version.
RQD4	2	Any configuration files must use a simple text format or an editor for the format must be included. (XML may be considered simple.)
RQD5	1	All data will be stored in open accessible formats such as XML.
RQD6	3	The system shall include an administrator's guide, a user's guide, tutorials and a video introduction that demonstrates a complete script being made, installed and run.
RQD7	4	The number of libraries that this system depends on should be minimised, but not at the expense of best practice.

4.3 Software Text Requirements

These are requirements concerning the quality of the source code for the system.

Identifier	Priority	Requirement
RQT1	3	It must be easy to extend the system to support additional requirements.
RQT2	2	It must be easy to understand the programs so that other developers can maintain and extend it.
RQT3	2	Variables and methods will have a consistent naming standard.
RQT4	2	It shall be easy to adapt the software for new locales.

Identifier	Priority	Requirement
RQT5	3	It shall be possible to adapt the software for any platform on which Linux runs a conforming desktop environment.
RQT6	4	It must be easy to replace the system with an alternative product.
RQT7	2	Support for testing the components must be included as part of the system.

4.4 Software Specification Requirements

These are requirements concerning the quality of the specification and design of the system.

Identifier	Priority	Requirement
RQS1	2	The design of the system must be easy to follow.
RQS2	3	It shall be straightforward to build the system.
RQS3	3	It shall be easy to adapt the system to accommodate new requirements.
RQS4	3	The design shall use a consistent naming standard.
RQS5	3	The design shall allow the system to be extended to cater for additional functionality.
RQS6	3	The system shall be constructed in a modular manner with the separate components capable of independent testing.
RQS7	4	Components of the system will be as independent of each other as is practical so that they may be changed without adversely affecting other components.

4.5 Development Process Requirements

These are requirements that relate to the development process that is responsible for constructing the system.

Identifier	Priority	Requirement
RQP1	3	The system shall be developed using a well defined development methodology.

Appendix A - Commands

The following commands have been selected for preparation for this system:

Command	Comments
alias	Not really applicable for a script language.
apropos	Built in to the editor's help system.
at	Not applicable.
atq	Not applicable
atrm	Not applicable
basename	
bc	
break	Not required.
bzip2	
cal	
case	Not required.
cat	
cd	Built in.
chgrp	
chmod	
chown	
cksum	
cmp	
colrm	
column	
comm	
continue	Built in.
cp	

Command	Comments
cpio	
crontab	Not required.
cut	
date	
df	
diff	
diff3	
dirname	
du	
echo	Built in.
env	
eval	Not required
ex	
exec	Not required
exit	Not required.
expand	
export	Not required.
expr	
false	
fc	Not required.
file	
find	
fmt	
fold	
for	Built in.

Command	Comments
free	
function	Built in.
grep	
gunzip	
gzip	
head	
history	Not required.
hostname	
id	
if	Built in.
ifconfig	
info	Built into the editor help system.
ispell	
jobs	Built in.
join	
kill	
killall	
last	
less	May need a terminal session to work.
let	Built in.
ln	
locate	
logname	
lpq	
lpr	

Command	Comments
lprm	
lpstat	
ls	
lsof	
lsusb	
man	Built into the editor help system.
mkdir	
mkfifo	
more	May need a terminal session.
mv	
netstat	
nice	
paste	
pidof	
ping	
popd	Not required.
pr	
printenv	
ps	
pushd	Not required.
pwd	
read	Built in
return	Built in
rm	
rmdir	

Command	Comments
rpm	
sed	
select	Not required
seq	
set	Built in
shift	Not required
sleep	
sort	
source	Not required
split	
strings	
tac	
tail	
tar	
tee	
test	
time	
times	
top	May need a terminal session
touch	
tr	
traceroute	
trap	
true	
typeset	Not required or built in ?

Command	Comments
ulimit	Not required.
umask	
unalias	Not required.
uname	
unexpand	
uniq	
units	
unset	Not required.
until	Built in.
uptime	
users	
usleep	
vmstat	
w	
wait	Built in.
wall	
wc	
wget	
whatis	
which	
while	Built in.
who	
whoami	
whois	
xargs	

Command	Comments
yes	Not required.
zcat.	

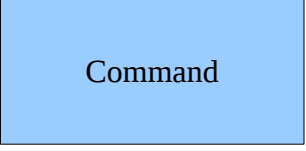


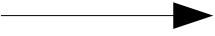


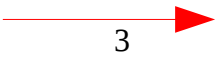



Appendix B – EBNF






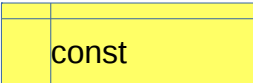




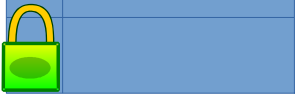
The following is the specification for the EBNF to be used, defined in its own terms:

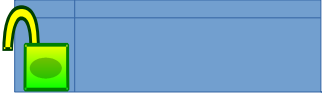
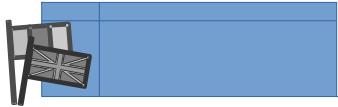
```
Grammar ::= { Production } ;
Production ::= Name " ::= " Symbol { Symbol } ";" ;
Symbol ::= Terminal-Symbol | Nonterminal-Symbol ;
Terminal-Symbol ::= Quotation [ "... " Quotation ] ;
Quotation :: Quote Characters Quote ;
Quote ::= "'" | '"' ;
Nonterminal-Symbol ::= Name | Choice | Option | Repetition ;
Choice ::= Symbol "|" Symbol ;
Option ::= "[" Symbol "]" ;
Repetition ::= "{" Symbol "}" ;
```

Appendix C – Symbols

This section defines the symbols used in the diagrams. The colours, patterns and textures are the default values.

	<p>A rectangle represents a command that is to be executed.</p>
	<p>A diamond can also be used for commands, but normally for those where the exit status is important to the logic of the script.</p>
	<p>A blue rectangle represents a command running as a background task.</p>
	<p>A black arrow represents an unconditional flow of control.</p>
	<p>A green arrow represents a flow of control where the previous command succeeded.</p>
	<p>A red arrow represents a flow of control where the previous command failed.</p>
	<p>An arrow with a number represents a flow of control for a specific exit status.</p>
	<p>A thick green arrow represents a pipe from stdout of the previous command to stdin of the next.</p>
	<p>A thick red arrow represents a pipe from stderr of the previous command to stdin of the next.</p>
	<p>A thick grey arrow represents a pipe between stdout and stderr of the previous command and stdin of the next.</p>

	<p>A blue arrow represents a signal between one process and another.</p>
	<p>A function is named by starting it from a rounded rectangle with a name.</p>
	<p>A green function start indicates that a menu item should be created for the function so that it can be run independently.</p>
	<p>A function can be referenced from another function using this symbol.</p>
	<p>A variable can be shown.</p>
	<p>A constant value is shown using a variable symbol in this yellow colour.</p>
	<p>A file is shown as this symbol.</p>
	<p>This represents an in-line file.</p>
	<p>This represents a temporary file which will be deleted when the script ends.</p>
	<p>This will represent a named pipe.</p>
	<p>This represents taking a lock on a mutex.</p>

	<p>This represents releasing a lock on a mutex.</p>
	<p>This represents a semaphore variable.</p>

Appendix D – Ideas and Proposals

The appendix contains ideas and proposals for enhancements to VICI that have not been formalised into specific requirements.

1. Integrate the Kaptain scripting system for creating dialogs for specifying commands: <http://kaptain.sourceforge.net/>

Kaptain is a universal graphical front-end for command line programs. It works on linux/UNIX platforms wherever Qt is available. Release 0.73 is using qmake and is compatible with Qt 4.

Someone writes a simple script (so called grammar) which describes the possible arguments for a command line program and Kaptain brings up a friendly dialog to the user to set up the command line.

2. Optionally place a grid on the flow chart, with user defined spacing, and optionally snap objects to the grid. This would remove the need to do fiddly aligning of the objects.
3. Use some graph theory and the object and line properties to find problems with the syntax of the flow charts.
4. Allow the user to undo changes. This will require the system to keep snapshots or mementos of the current flowchart so that it can be restored.
5. Performance boost by pooling Pipeline objects where they are reused in loops. Will require some way of determining what should be pooled, such as if it has been run before.
6. An interface to dbus that would allow the script to be either a client or dbus server.
7. Add the `info -apropos` command to search.
8. Allow local variables in functions. This would allow more interesting recursive functions.
9. Add “manifold” built-in programs so that flow of control can be split into separate threads and joined.
10. Add a data-manifold so that streams can be merged and the next process is blocked until data is available on all inputs.
11. Add support for message queues.
12. ~~Add parenthesis to the EBNF to define a sequence. This would reduce the need for subgraphs.~~
13. Add the ability to merge syntax charts.
14. ~~Add an export to SVG for the syntax charts.~~

15. Add ~~an export to SVG for the vici charts~~ and an XSL script to create a report of the script.
16. An option to for-each that runs its children in simultaneous threads.
17. Allow the user to add a description of the script. The help menu item on vici can display it, and it can be used as the description in the desktop file.
18. Add an internal command to translate URLs to filenames.