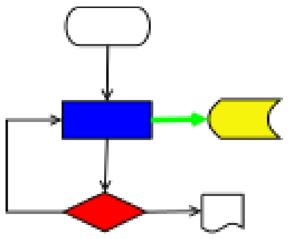
VICI



VISUAL CHART INTERPRETER

Design of libcommand

Publication History

Date	Who	What Changes
4 October 2012	Brenton Ross	Initial version.
4 January 2014	Brenton Ross	Detailed design
19 June 2014	Brenton Ross	Updated for Vici

Table of Contents

1 Introduction

This is part of the system design document for the VICI project.

1.1 Scope

This document covers the detailed design of the command library. This component is responsible for allowing the user to enter a command and its parameters that will be executed as part of a VICI script.

The document will cover the Application Design and the User Interface Design.

This design is for increment #3.

1.2 Overview

The detailed design includes:

- Interface Stubs: A framework of facade classes for the modules.
- Use Case Descriptions: A description of how a user is expected to interact with the application.
- Application Design: The classes and their relationships.
- User Interface Design: The design and layout of the graphical components of the system.
- Persistent Storage Design: The specifications for the XML files used to store configuration and scripts.

1.3 Audience

This document is intended to be used by the designers and developers, and later the maintainers, of the VICI project.

2 Overview

2.1 Responsibilities

The libcommand library is responsible or providing a user interface for entering the commands that are executed by the VICI run-time. It has the following responsibilities:

T4.1: Display a list of prepared commands.

T4.2: Attach the selected command to the selected symbol.

T4.3: Verify the selected command is appropriate for the selected symbol.

T4.5: Collect a command and its options and attach it to the selected symbol.

T5.1: Provide a window for displaying help text.

T5.2: Display the short description of prepared commands.

T5.3: Display the results of running the help command for prepared commands.

T5.4: Display the man page for non-prepared commands.

T5.5: Display the info page for the selected command.

T6.1: Display the syntax chart for the command.

T6.2: Display the set of options and parameters available for the command.

T6.4: Indicate which options and parameters are legal at the current point in the command.

T6.5: Build a command line for the command from the user's selection of options and parameters.

2.2 Interfaces

The architecture document describes the following interfaces to th command library. The library must implement these interfaces.

2.2.1 IF05 Command UI

This is the user interface that allows a user to select a command and set its options and parameters.

Transport Medium:	Displayed in a window.
Protocol:	Event driven with Windows, Icons, Menus and a Pointer.

Content:

1. Prepared Command List (O)
2. User command selection (I)
3. User suggested command and options. (I)
4. Short description of a prepared command (O)
5. Use selection of type of help text. (I)
6. Help text. (O)
7. Option and parameter list (O)
8. User option or parameter selection. (I)
9. Command line (I/O)

2.2.2 IF24 Vici-ed using Command

This is the interface vici-ed uses to display the Command component that is responsible for constructing the command and its options and parameters.

Transport Medium: Memory

Protocol: C++ function calls.

Content:

1. Sub-window (I)
2. The command (I)
3. The parameters and options (O)

2.2.3 IF25 Command using libsyntax

This interface allows the Command component to display the syntax diagram for the current command.

Transport Medium: Memory

Protocol: C++ function calls.

Content:

1. The EBNF specification for the command (I)
2. The sub-window to display the syntax chart into. (I)

2.2.4 IF26 Command using libxmlwrap

This interface allows the Command component to read the database of commands.

Transport Medium: Memory

Protocol: C++ function calls.

Content:

1. Create an XML document structure.
2. Add nodes to the document.
3. Add properties to nodes.
4. Save the document.

5. Read an XML file
6. Get nodes from the document
7. Remove nodes from the document.
8. Get properties from the document node.

2.2.5 IF27 Command using libebnf

This interface allows the Command component to suggest valid alternatives for parameters and options.

Transport Medium: Memory

Protocol: C++ function calls.

Content:

1. Request the parsing of an EBNF string returning a pointer to a parse tree data structure.

2.2.6 IF37 Command using libQtGui

This interface allow the command component to provide the graphical elements that allow user interaction.

Transport Medium: Memory

Protocol: C++ function calls.

Content:

1. Command lists.
2. Text display fields.
3. Option and parameter lists.
4. User selections
5. User text input.

2.3 Design Approach

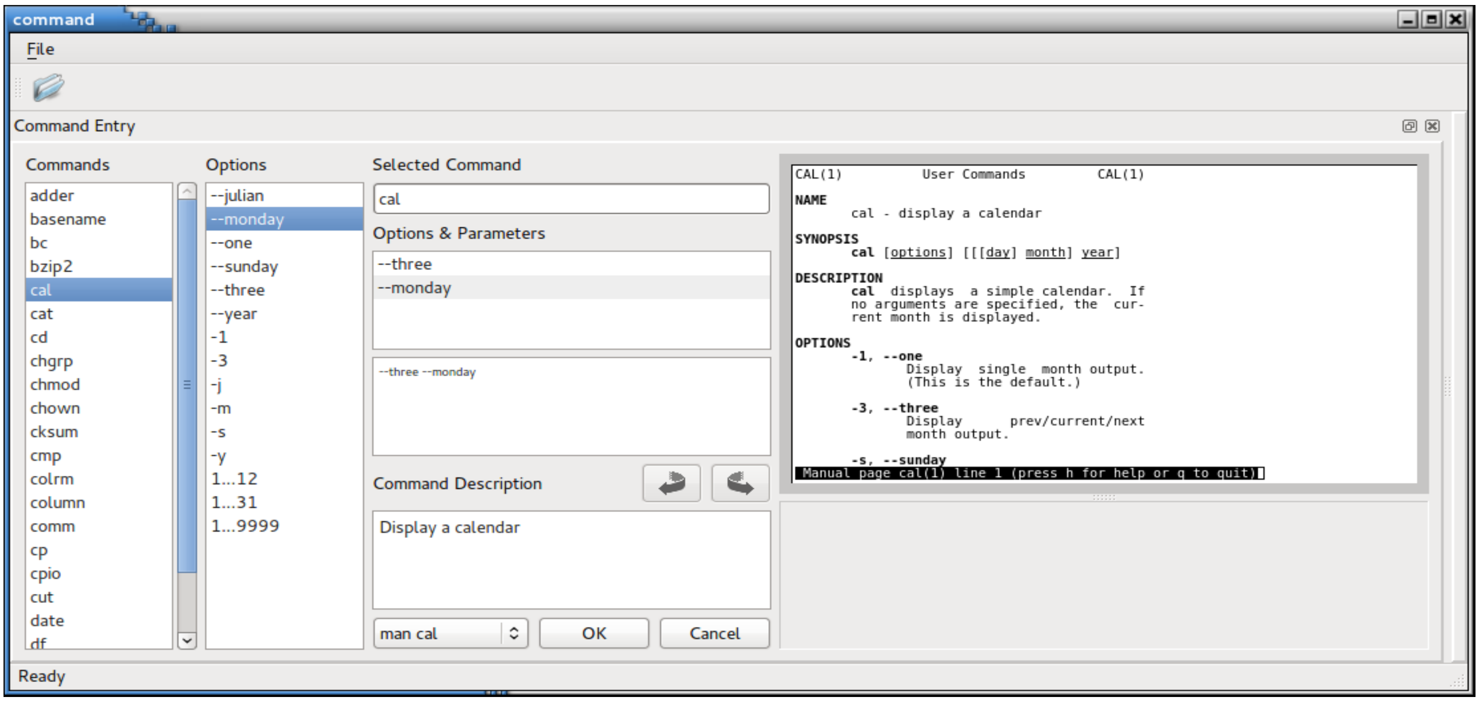
The Qt library provides a suitable set of widgets for building the interface. A feature of the Qt library is an extension to the class interface that allows one to define functions as “signals”or “slots” which can then be connected together making the usual widget call-back pattern unnecessary.

The most complex part is the running of the *man* or *info* in a terminal session. Several alternatives were prototyped, but the most successful was to run an X-Terminal as an embedded application. The user must then use the user interface provided by the man or info programs in text mode, which is a bit of a mismatch.

Otherwise the library is a standard user interface.

3 User Interface

The following diagram illustrates the user interface provided by the library.



Commands: adder, basename, bc, bzip2, cal, cat, cd, chgrp, chmod, chown, cksum, cmp, colrm, column, comm, cp, cpio, cut, date, df

Options: --julian, --monday, --one, --sunday, --three, --year, -1, -3, -j, -m, -s, -y, 1...12, 1...31, 1...9999

Selected Command: cal

Options & Parameters: --three, --monday

--three --monday

Command Description: Display a calendar

man cal

```
CAL(1)              User Commands             CAL(1)

NAME
       cal - display a calendar

SYNOPSIS
       cal [options] [[[day] month] year]

DESCRIPTION
       cal  displays  a simple calendar.  If
       no arguments are specified, the  cur-
       rent month is displayed.

OPTIONS
       -1, --one
              Display  single  month output.
              (This is the default.)

       -3, --three
              Display      prev/current/next
              month output.

       -s, --sunday
 Manual page cal(1) line 1 (press h for help or q to quit)
```

The “Commands” column lists all the commands that have been prepared for Vici. Selecting one of these causes the options to be presented, and the command to be entered into the “Selected Command” edit field. It also causes the description to be displayed, and the man or info page to be displayed.

Double clicking on an entry in the “Options” column causes it to be appended to the “Options & Parameters” list. Entries in the list can be edited or dragged within the list. (Later enhancement should allow dragging of file names from Nautilus or other file managers.) The current full option list is displayed below the list.

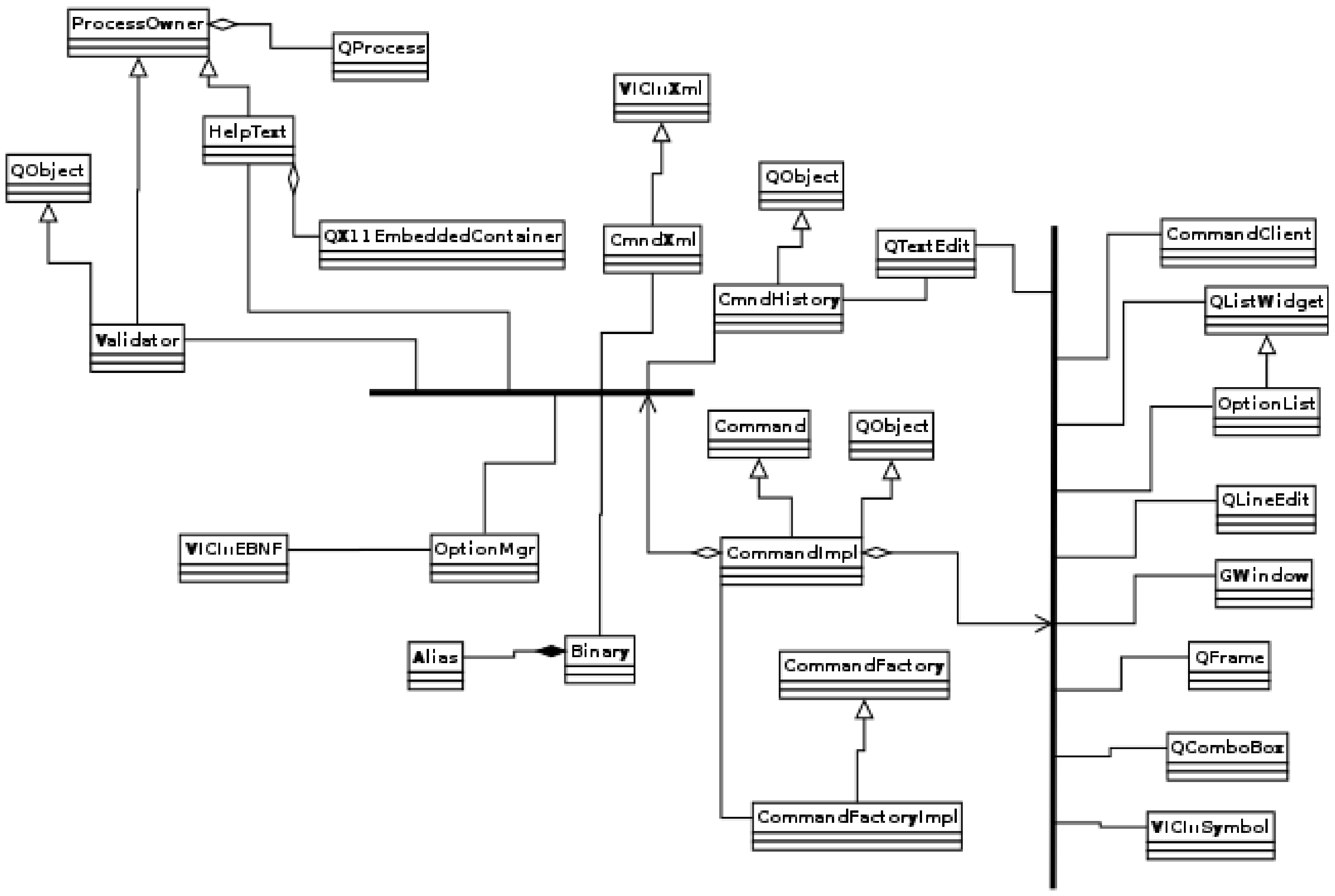
Previous options and parameters for the selected command can be accessed from the history arrows. This speeds up the case where a command is being reused with slightly different options.

The user can also select either man pages or info pages to be displayed in the terminal session at the right.

The syntax chart (not shown) is displayed below the terminal, allowing the user to drill down to the allowed options for the selected command.

4 Application Design

The following diagram describes the relationships between the classes used in this library.

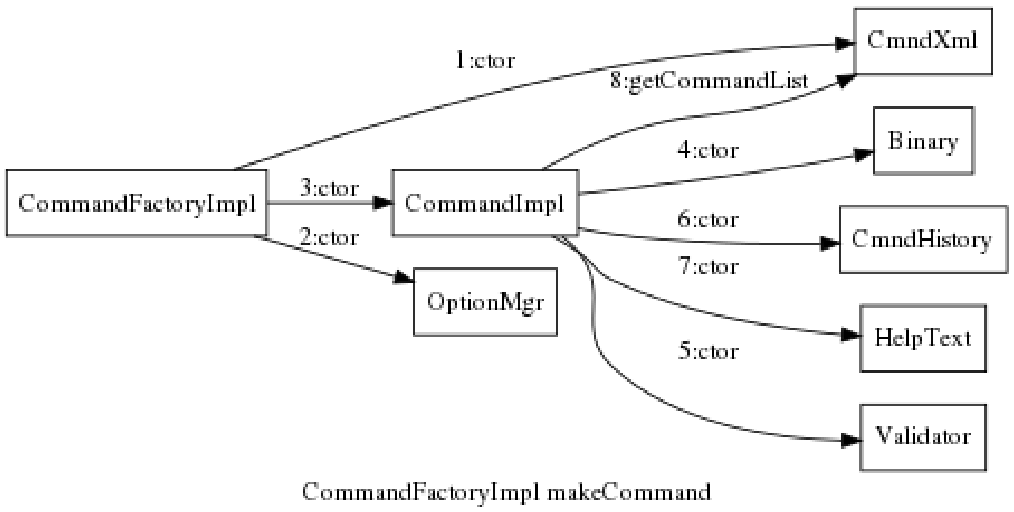


5 Collaboration Diagrams

The following diagrams illustrate the interactions that take place when use cases are performed.

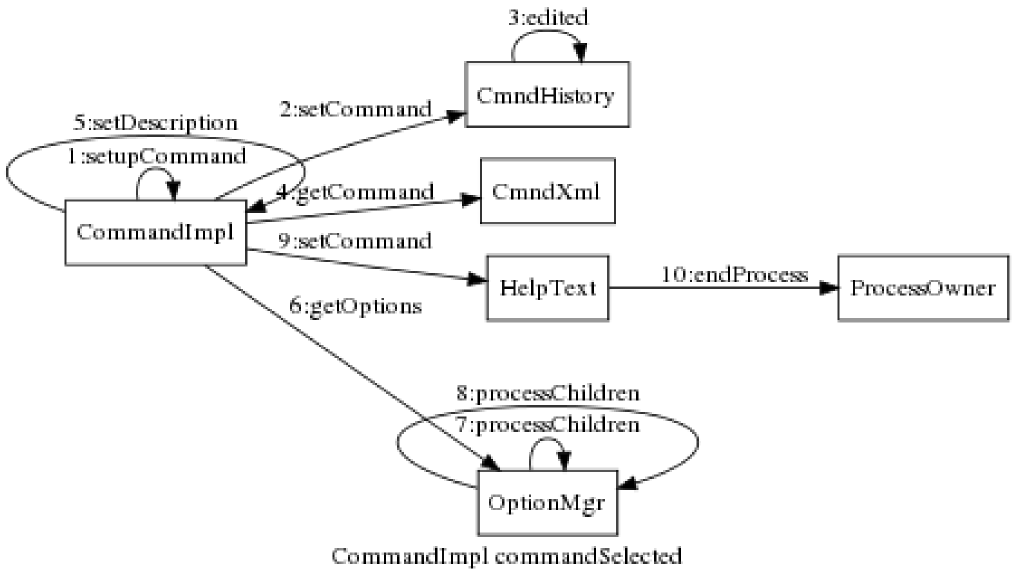
5.1 Starting the Application

The following diagram shows the sequence of events as the library initialises itself.



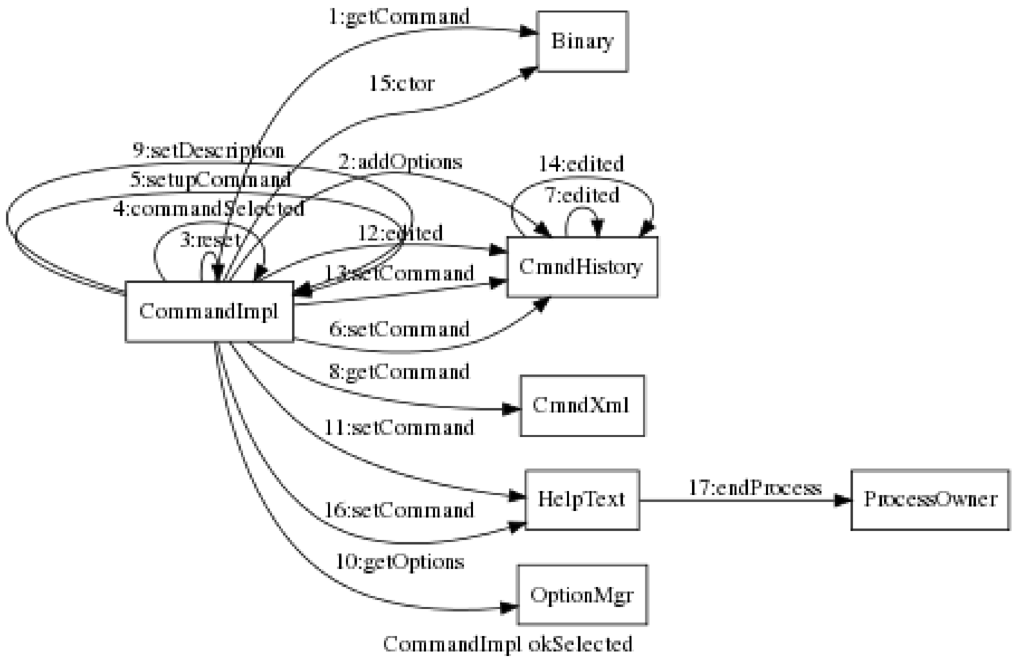
5.2 Selecting a Command

The following diagram shows the sequence of calls when the user selects a command.



5.3 Accepting a Command

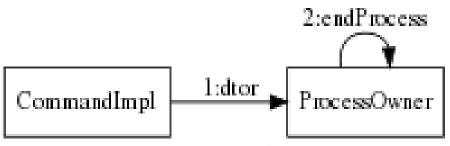
The following diagram shows the sequence of calls when the user presses OK.



CommandImpl okSelected

5.4 Stopping the Application

The following diagram shows the sequence of events as the library shuts down.



CommandImpl dtor

6 Class Designs

This section describes each class, including its responsibilities, and its public and protected members.

6.1 ProcessOwner Class

This class is responsible for managing a QProcess object, ensuring that it is removed once it has completed. The class is used as a parent for the help text display and also the command validator.

```
class ProcessOwner
{
protected:
        QProcess *process;
       void endProcess();
public:
        ProcessOwner();
        virtual ~ProcessOwner();
};
```

6.2 HelpText Class

This class is responsible for displaying the help text for a command. This is done by running an xterm instance and passing it the man or info command to run.

```
class HelpText : private ProcessOwner
{
public:
        HelpText( QWidget *frame );
       void setCommand( csr );
        QWidget *widget();
};
```

6.3 Validator Class

This class is responsible for ensuring that a command can be executed. It does this by running the “which” command which checks the user's PATH variable to find an instance of the command.

```
class Validator : public QObject, private ProcessOwner
{
public:
        Validator( QWidget *parent );
       void checkCommand( csr );
};
```

6.4 OptionMgr Class

This class is responsible for getting the options for a selected command. It uses the EBNF parser to create a parse tree which it then extracts the options from.

```
class OptionMgr
{
public:
        OptionMgr();
        void getOptions( VICI::csr ebnf, std::vector< std::string > & options );
};
```

6.5 Binary Class

This class is an open data structure containing the data extracted from the XML file for a particular command.

```
class Binary
{
public:
        Binary( csr name );
        std::string getCommand( csr name ); // name may be an alias

        std::string name;
        std::string description;
        std::string ebnf;
        std::vector< std::string > helpCommands;

        // associative array indexed by the name of the alias
        std::map< std::string, Alias > aliases;
};
```

6.6 Alias Class

This class is also an open data structure containing the data extracted from the XML file for an alias of a command.

```
class Alias
{
public:
        Alias( csr name, Binary * );
        Alias();

       std::string name;
        std::string options;
        std::string description;

        Binary * getOwner() { return owner; }
};
```

6.7 CmndXml Class

This class is responsible for getting the command data from the XML file.

```
class CmndXml : public VICI::Xml
{
public:
        CmndXml();

        // get list of all prepared commands, and all the defined
        // aliases from the command database.
        int getCommandList( std::vector< std::string > & );

        // get the details of a command,
        // name may be either a command or an alias
        bool getCommand( csr name, Binary & );

};
```

6.8 CmndHistory Class

This class is responsible for maintaining the history of options associated with each command. The aim is to make it easier to build scripts that use the same command several times, but with slight variations in the options.

```
class CmndHistory : public QObject
{
 public:
        CmndHistory( QTextEdit * );

        // called when user hits OK
        // adds command options to end of list, removing any duplicates
        void addOptions( const QString &options );
        void setCommand( const QString &cmnd );

public slots:
        void next();
        void prev();
        void edited(); // the ui has been edited

signals:
        void allowNext( bool );
        void allowPrev( bool );
};
```

6.9 CommandFactoryImpl Class

This class is the implementation of the CommandFactory class which is responsible for creating an instance of the Command class.

```
Class CommandFactoryImpl : public CommandFactory
{
public:
        virtual Command * makeCommand( Window *, CommandClient * );
};
```

6.10 CommandImpl Class

This class is the implementation of the Command class which is responsible for collecting a command and its options from the user and attaching them to a symbol.

```
class CommandImpl : public QObject, public Command
{
protected slots:
        void commandSelected();
        void commandEntered();
        void optionAdded(QListWidgetItem *);
        void helpOptionSelected( const QString & );
        void okSelected();
        void cancelSelected();
public:
        CommandImpl( Window *, CommandClient * );
        virtual ~CommandImpl();
        virtual void show();
        virtual void setCommand( csr );
        virtual void selection(VICI::Symbol::Symbol*);

        // needed for the interface, but not of interest to Command
        virtual void symbolAttr(VICI::Symbol::SymbolAttributes&){}
        virtual void textAttr(VICI::Symbol::TextAttributes&){}
};
```

Appendix A