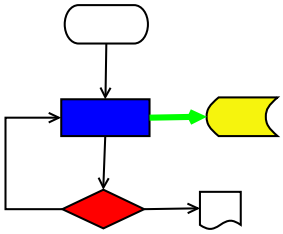
VICI



VISUAL CHART INTERPRETER

Design for libcron

Publication History

Date	Who	What Changes
29 November 2018	Brenton Ross	Initial version.

Table of Contents

1 Introduction

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

1.1 Scope

This document covers the design of libcron which is a library for scheduling scripts to run at a future time.

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 libcron library and associated programs are responsible for allowing the user to schedule a script to be run at some future time, either once or regularly.

It should be possible to define schedules like those of crontab, and also more complex ones like “fortnightly”, or “the 2nd Thursday of each month”, or “the last working day of the month”.

It should be possible to specify what to do if a task cannot be run at the specified time – either skip it or run as soon as possible.

It should be possible to set a priority for the tasks so that several are not started at once.

It should be possible to set several schedules for the same script.

It should be possible to set parameters/arguments to the script for each schedule.

When a script is started it should be possible to have it automatically start, or not.

2.2 Design Approach

The libcron library will provide a user interface for the vici-editor program where the user can specify the schedule.

A vici-cron program will start scripts according to the schedule. This program will be autostarted when the user logs in. A UNIX domain socket will be used to ensure that there is only a single running instance of vici-cron for each user, and provide a means for libcron to notify vici-cron of updates to the schedule.

A scheduling language will be defined. This will be stored on disk as XML and a GUI interface will allow the user to specify a schedule for a script.

3 Scheduling Language

This language will define what schedules can be expressed.

At this stage we will omit constraints such as public holidays, or time off taken by the user.

```
Schedule ::= month_spec date_spec time_spec ;

month_spec ::= “all” | month_list ;
month_list :: month [ “,” month ] ;

time_spec ::= time_list ;
time_list ::= time [“,” time ] ;
time ::= hour ”:” minute ;
minute ::= 00 | 15 | 30 | 45 ;
hour ::= 0...23 ;

date_spec ::= “all” | weekday_list | date_interval | month_date ;
weekday_list ::= weekday [“,” weekday ] ;
weekday ::= 1...7 ; // or names

date_interval ::= date [ interval ] ;
interval ::= 1...366 ;
date ::= day_of_month month year ;
month ::= 1...12 ; // or names or abbreviated names
day_of_month ::= 1...31 ;
year ::= 2018...2100 ;

month_date ::= days_of_month | weekdays_of_month | relative_days ;
days_of_month ::= day_of_month [ “,” day_of_month ] ;
weekdays_of_month ::= weekday_of_month [“,” weekday_of_month ] ;
weekday_of_month ::= ordinal weekday ;
ordinal ::= “first” | “second” | “third” | “fourth” | ... ;

relative_days ::= relative_day [“,” relative_day ];
relative_day ::= [ordinal][“last”][“workday”] ;
```

3.1.1 Data Type Definition

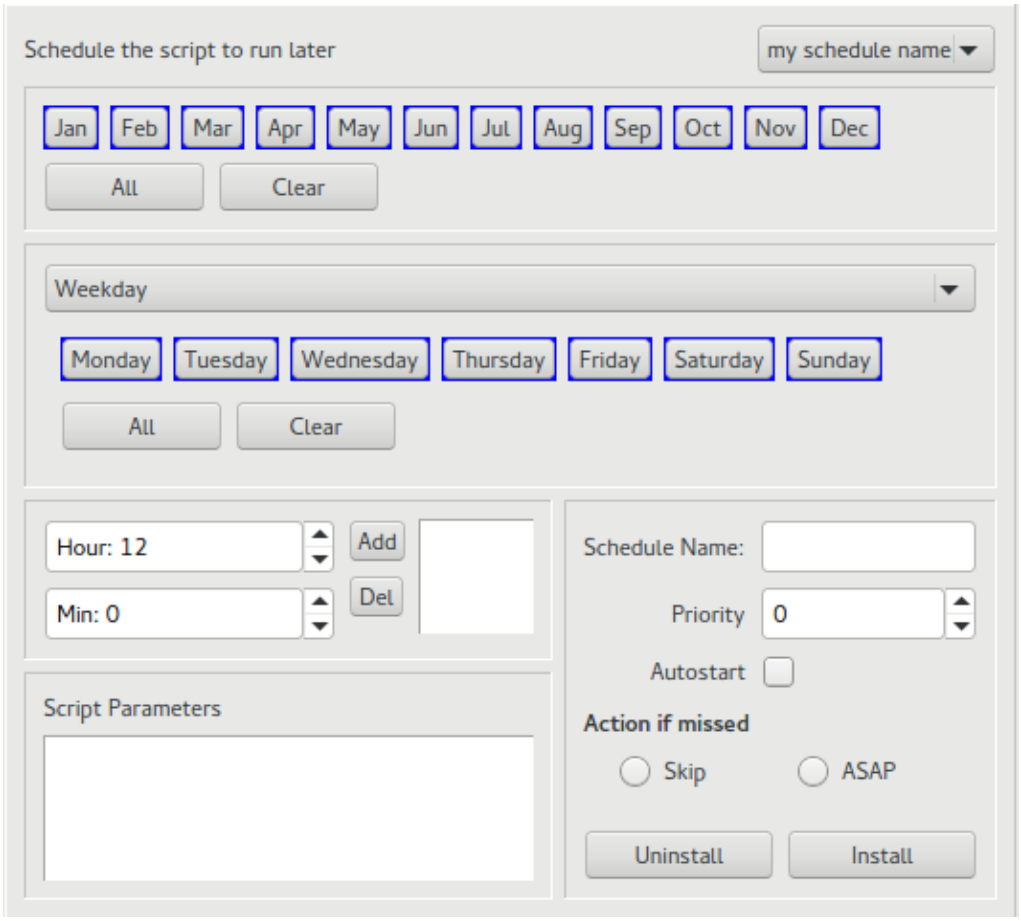
The DTD for the scheduler data:

```
<!ELEMENT VICI-CRON (Schedule*) >
<!ELEMENT Schedule (Name, Path, Param*, Spec) >
<!ATTLIST Schedule  Missed (Skip | ASAP ) #REQUIRED
                    Priority CDATA #IMPLIED
                    AutoStart (true | false ) #REQUIRED
                    >

<!-- A unique name for the schedule -->
<!ELEMENT Name (#PCDATA) >
<!-- The script path to execute -->
<!ELEMENT Path (#PCDATA) >
<!-- One parameter string per element -->
<!ELEMENT Param (#PCDATA) >
<!ATTLIST Param Seq CDATA #REQUIRED >

<!ELEMENT Spec (Months Dates Times ) >
```

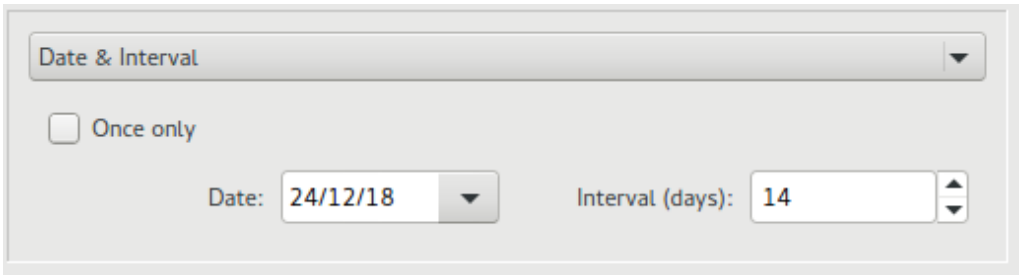
```
<!ELEMENT Months (Month*) >
<!ATTLIST Months all (true | false) #IMPLIED >

<!ELEMENT Month EMPTY >
<!ATTLIST Month name CDATA #REQUIRED >

<!ELEMENT Times (Time+) >
<!ELEMENT Time EMPTY >
<!ATTLIST Time      hour CDATA #REQUIRED
                    min (0 | 15 | 30 | 45 ) #REQUIRED
                    >

<!ELEMENT Dates (Weekdays | Interval | Monthday* )? >
<!ATTLIST Dates all (true | false ) #IMPLIED >

<!ELEMENT Weekdays (DayOfWeek+) >
<!ELEMENT DayOfWeek EMPTY >
<!ATTLIST DayOfWeek name CDATA #REQUIRED >

<!ELEMENT Interval Date >
<!ATTLIST Interval  interval CDATA #IMPLIED >
<!ELEMENT Date EMPTY >
<!ATTLIST Date      day CDATA #REQUIRED
                    month CDATA #REQUIRED
                    year CDATA #IMPLIED
                    >

<!ELEMENT Monthday ( DayOfMonth+ | WeekDayOfMonth+ | RelativeDay+)>
<!ELEMENT DayOfMonth EMPTY >
<!ATTLIST DayOfMonth       day CDATA #REQUIRED >

<!ELEMENT WeekDayOfMonth DayOfWeek >
<!ATTLIST WeekDayOfMonth   ordinal (first | second | third | fourth )
                           last (true | false)
                           >

<!ELEMENT RelativeDay EMPTY >
<!ATTLIST RelativeDay      ordinal (first | second | third | fourth )
                           last (true | false)
                           workday (true | false )
                           >
```

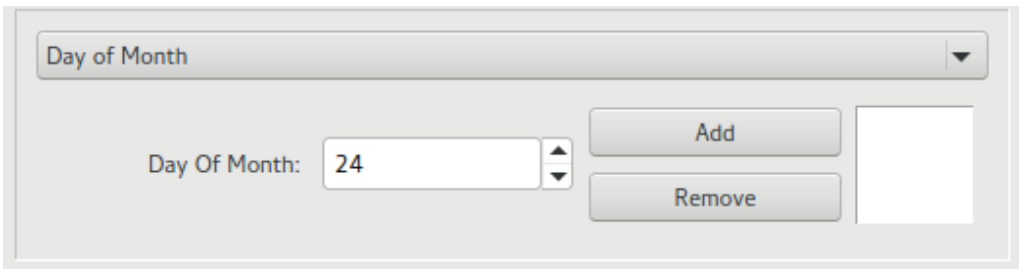
4 User Interface

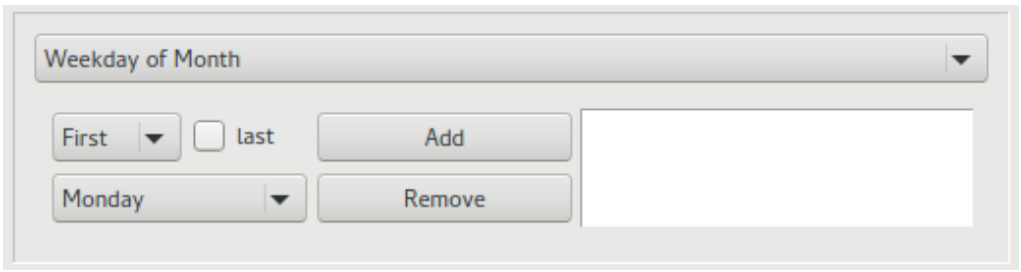
The user interface is a panel on the Install tab.

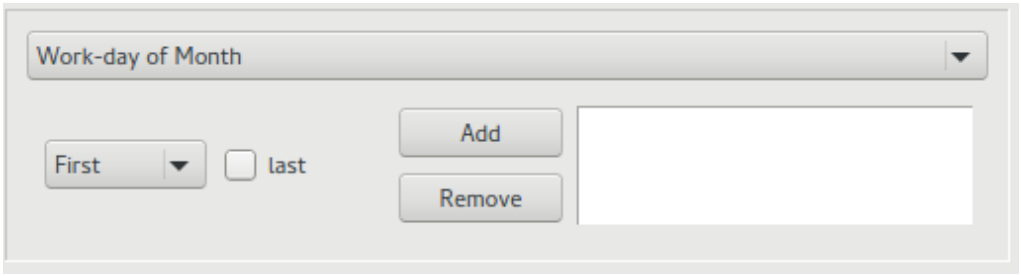


The panel for selecting the day can have the above, or one of the following:



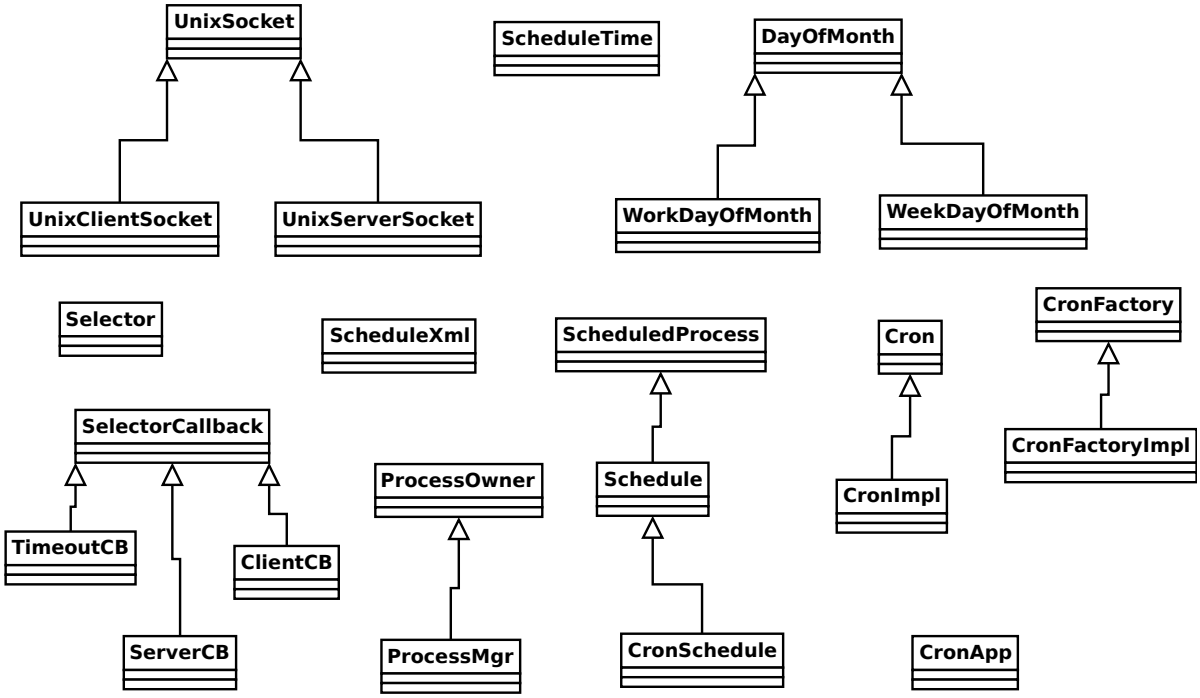



Weekday of Month
First
last
Add
Monday
Remove


Work-day of Month
First
last
Add
Remove

5 Application Design

The classes for the cron library and the vici-cron program are shown in the following diagram.



5.1 CronFactory

This class provides the abstract interface for an object that constructs an instance of Cron.

```
class CronFactory : public VICI::Factory
{
public:
      virtual ~CronFactory() {}
      virtual Cron * makeCron( Window *w) = 0;
};
```

5.2 CronFactoryImpl

This class constructs an instance of the implementation of Cron class.

```
class CronFactoryImpl : public VICI::Cron::CronFactory
{
public:
      virtual Cron * makeCron( Window *w);
};
```

5.3 Cron

This is the abstract class that defines the interface for the class that allows a user to schedule a script.

```
class Cron
{
public:
	virtual ~Cron() {}  ///< virtual destructor
	virtual void setCurrentFile( csr filename ) = 0;
	virtual void show() = 0;
};
```

5.4 CronImpl

This class is responsible for providing the user interface that allows the user to schedule a script. It is also responsible for managing the retrieving and storing of the schedule.

```
class CronImpl : public QObject, public VICI::Cron::Cron
{
public:
	explicit CronImpl(VICI::Window *w);
	virtual void setCurrentFile( csr filename );
	virtual void show(){}

};
```

5.5 ScheduledProcess

This class holds just enough of a schedule to allow the script to be started. It is passed to the ProcessMgr which uses it to launch the script.

```
struct ScheduledProcess
{
	std::string name;
	std::string path;
	bool autoStart;
	VICI::ArgList params;

	ScheduledProcess() : autoStart(false) {}
};
```

5.6 Schedule

This class is used to hold the data for the user interface provided by CronImpl. It is passed to and from the ScheduleXml.

```
class Schedule : public ScheduledProcess
{
public:
	enum DateType { Weekday, DateInterval, DayOfMonth,
		WeekdayOfMonth, WorkdayOfMonth };
public:
	Schedule();

	int priority;
	bool skipIfMissed, asapIfMissed;  // exactly one must be true

	std::vector<int> months;
	DateType dateType;

	std::vector<int> weekdays;
	struct Date
	{
		int day, month, year;
	} date;
	int interval;// zero means no interval
	std::vector<int> daysOfMonth;
	std::vector<WeekDayOfMonth> weekDaysOfMonth;
	std::vector<WorkDayOfMonth> workDaysOfMonth;
	std::vector<ScheduleTime> times;
};
```

5.7 ScheduleTime

This class provides a simple time structure with methods for converting to and from a string.

```
class ScheduleTime
{
public:
	ScheduleTime( int h, int m) : hours(h), minutes(m) {}
	explicit ScheduleTime( const QString & );

	QString toString() const;
	bool operator < ( const ScheduleTime & ) const;
	bool ok() const { return hours >= 0 && hours <= 24; }

	int getHours() const { return hours; }
	int getMins() const { return minutes; }
};
```

5.8 DayOfMonth

This class provides the common data and methods for the WeekDayOfMonth and WorkDayOfMonth classes.

```
class DayOfMonth
{
protected:
	bool last;
	int ordinal;

public:
	DayOfMonth( int ordinal, bool last );
	virtual ~DayOfMonth(){}

	virtual int toNumber() const = 0;
	static int getOrdinal(const QString &);
	static int getOrdinal(const std::string &);
	bool operator < ( const DayOfMonth & ) const;

	bool getLast() const { return last; }
	int  getOrdinal() const { return ordinal; }

	static const std::vector<std::string> ordinals;
	static const std::vector<std::string> days;

};
```

5.9 WeekDayOfMonth

This class is used to represent concepts like “The first Tuesday of the month” or “The second last Friday of the month”.

```
class WeekDayOfMonth : public DayOfMonth
{
	int  day;

public:
	// ordinal and day are 1 based (1..4, 1..7)
	WeekDayOfMonth(int ordinal, bool last, int day );
	explicit WeekDayOfMonth( const QString &);

	// return zero if not valid
	virtual int toNumber() const;
	QString toString() const;

	int getDay(const QString &);
	int getDay() const { return day; }
};
```

5.10 WorkDayOfMonth

This class is used to represent concepts like “The second working day of the month” or “The last working day of the month”.

```
class WorkDayOfMonth : public DayOfMonth
{
public:
	// ordinal is one based
	WorkDayOfMonth( int ordinal, bool last);
	explicit WorkDayOfMonth( const QString &);

	// return zero if not valid
	virtual int toNumber() const;
	QString toString() const;
};
```

5.11 UnixSocket

This class is the parent class for client and server sockets that use the Unix domain. This is used as the communication between libcron and the vici-cron server program, allowing the server to be notified whenever the schedule is updated.

```
class UnixSocket
{
protected:
	Path pathName;
	int fd;
	struct sockaddr_un name;
	bool eof;
public:
	explicit UnixSocket( const Path & );
	explicit UnixSocket( int fd );
	~UnixSocket();

	bool isEof() const { return eof; }
	int getFD() const { return fd; }
};
```

5.12 UnixServerSocket

The vici-cron program uses this to listen for connections from the vici-editor.

It also used to confirm that no other instance of vici-cron is running (for this user).

```
class UnixServerSocket : public UnixSocket
{
public:
	explicit UnixServerSocket( const Path & );
	UnixClientSocketPtr accept();
	~UnixServerSocket();
};
```

5.13 UnixClientSocket

This is used by the libcron library (in vici-editor) to notify the vici-cron program that the schedule has been updated.

```
class UnixClientSocket : public UnixSocket
{
public:
	explicit UnixClientSocket( const Path & );
	explicit UnixClientSocket( int fd );

	std::string read();
	void write( csr );
};
```

5.14 Selector

This class is responsible for waiting until data is available on a file descriptor, or the time-out has been reached. It is used by vici-cron to wait for the next scheduled event, or for connections from vici-editor.

```
class Selector
{
public:
	Selector( int timeoutsecs, SelectorCallback* );
	~Selector();
	void setTimeout( int timeoutsecs );

	void registerCallback( int fd, SelectorCallback *);
	void unregisterCallback(int fd);

	void wait();
	void stop() { stopping = true; }

};
```

5.15 SelectorCallback

This is an abstract function object that is called when an event occurs in the Selector object.

```
class SelectorCallback
{
public:
	virtual ~SelectorCallback(){}
	virtual void operator ()() = 0;
};
```

5.16 TimeoutCB

A function object that is called when there is a time out event.

```
class TimeoutCB : public VICI::SelectorCallback
{
public:
      TimeoutCB( CronApp *x, std::function<void(CronApp *)> f )
            : obj(x), fn(f) {}
      void operator() () { fn(obj); }
};
```

5.17 CronSchedule

This class is responsible for determining the current and next time points for a schedule.

The operation (a bit brute force) involves looping over last year, this year and next year (so as to avoid problems during the end of year transitions), then looping over the selected months (except for the Date & Interval mode), then looping over the dates, and finally looping over the times. Each time point is then compared to the range beginning from when the schedule was last examined and a future time, usually 15 minutes into the future. Any matching time point is added to a list.

The object is then able to respond to requests for the most recent and the next event for the schedule (if they exist).

The CronApp object combines the results from each schedule to determine which processes to start, and how long the timeout should be.

5.18 CronnApp

This class waits for either a timeout or a notification that the schedule has changed and then checks with each CronsSchedule object to find any which need to be started. The ScheduledProcess part is then placed on a priority queue.

A separate thread waits until it is notified and picks the ScheduledProcess from the queue and launches a child process to run the script.

Appendix A