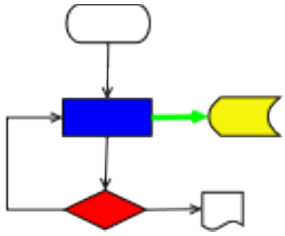
VICI



VISUAL CHART INTERPRETER

Design of libsymbol

Publication History

Date	Who	What Changes
18 September 2014	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 detailed design of the symbol component. This component is responsible for displaying a palette of flow chart symbols.

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

This design is for increment #1.

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

This component provides a palette of flow chart symbols that user selects from to place onto the flow chart.

It addresses the following responsibilities:

- T3.1: Display a set of flowchart symbols in a palette. See Appendix C of the Requirements document.
- T3.2: Indicate a default symbol on the palette.
- T17.1: Provide a logical to actual mapping for the symbol attributes.
- T17.2: Provide a palette of colours for the symbol attributes.
- T17.3: Provide a palette of patterns for the arrows and lines.
- T17.4: Provide a palette of textures for the area symbols
- T9.1: Provide a palette of text attributes.

For the first iteration we will be omitting T17.4.

2.2 Interfaces

2.2.1 IF03 Symbol Palette

This is the graphical interface that allows a user to select and configure the symbols used on the flowchart diagrams.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:

1. Palette of flowchart symbols. (O)
2. Indication of currently selected symbol. (O)
3. Palette of colours. (O)
4. Palette of line patterns. (O)
5. Palette of textures. (O)
6. Palette of text attributes. (O)
7. User selected colour. (I)
8. User selected line pattern. (I)
9. User selected texture. (I)
10. User selected text attributes. (I)

2.2.2 IF18 Vici-ed using Symbol

This is the interface between then vici-ed program and the component responsible for managing and displaying the symbols used on the flowchart.

Transport Medium: Memory

Protocol: C++ function calls, Qt signals and slots.

Content:
1. Sub-window in which it the symbol library uses to display its visual components. (I)
2. Signal that indicates a change of selected symbol. (I/O)
3. Attributes of text and symbols. (O)

2.2.3 IF19 Canvas using Symbol

This is the interface between the Canvas component and the Symbol component.

Transport Medium: Memory

Protocol: C++ function calls., Qt signals and slots

Content:
1. Drawing surface (I)
2. Position of required symbol (I)
3. Currently selected symbol (O)
4. Attributes of text and symbols. (O)

2.2.4 IF29 Symbol Library using libQtGui

This interface allows the symbol component to display the symbol and text palettes.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Symbol palette
2. Text attributes

2.3 Design Approach

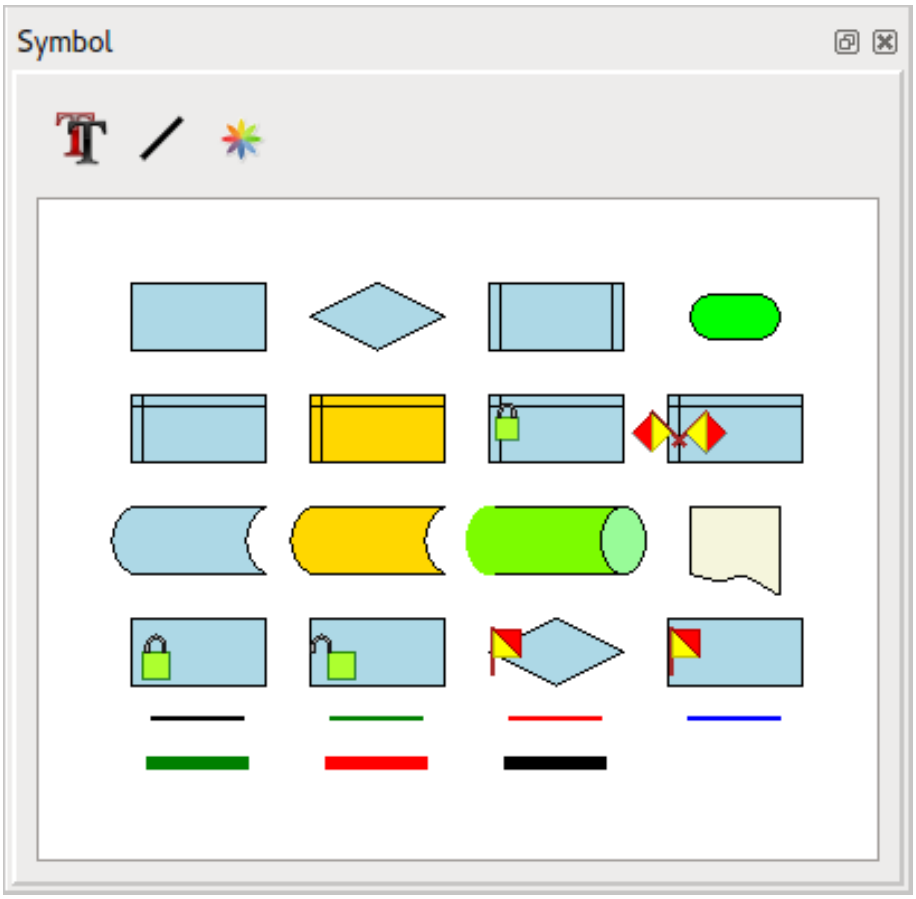
The symbols are drawn using the Qt library's QGraphicsScene and QGraphicsItem classes. This is an efficient library for drawing the geometric shapes involved.

An alternative would have been to use SVG and Inkscape to draw the shapes and then use the Qt SVG library to render them. This may be a viable approach if the number of symbols needs to be expanded or the level of detail on the symbols increased.

Most of the complexity arises from the need to keep Qt out of the high level interface descriptions, as defined in vici.h.

3 User Interface

The following diagram illustrates the user interface for the symbol component.

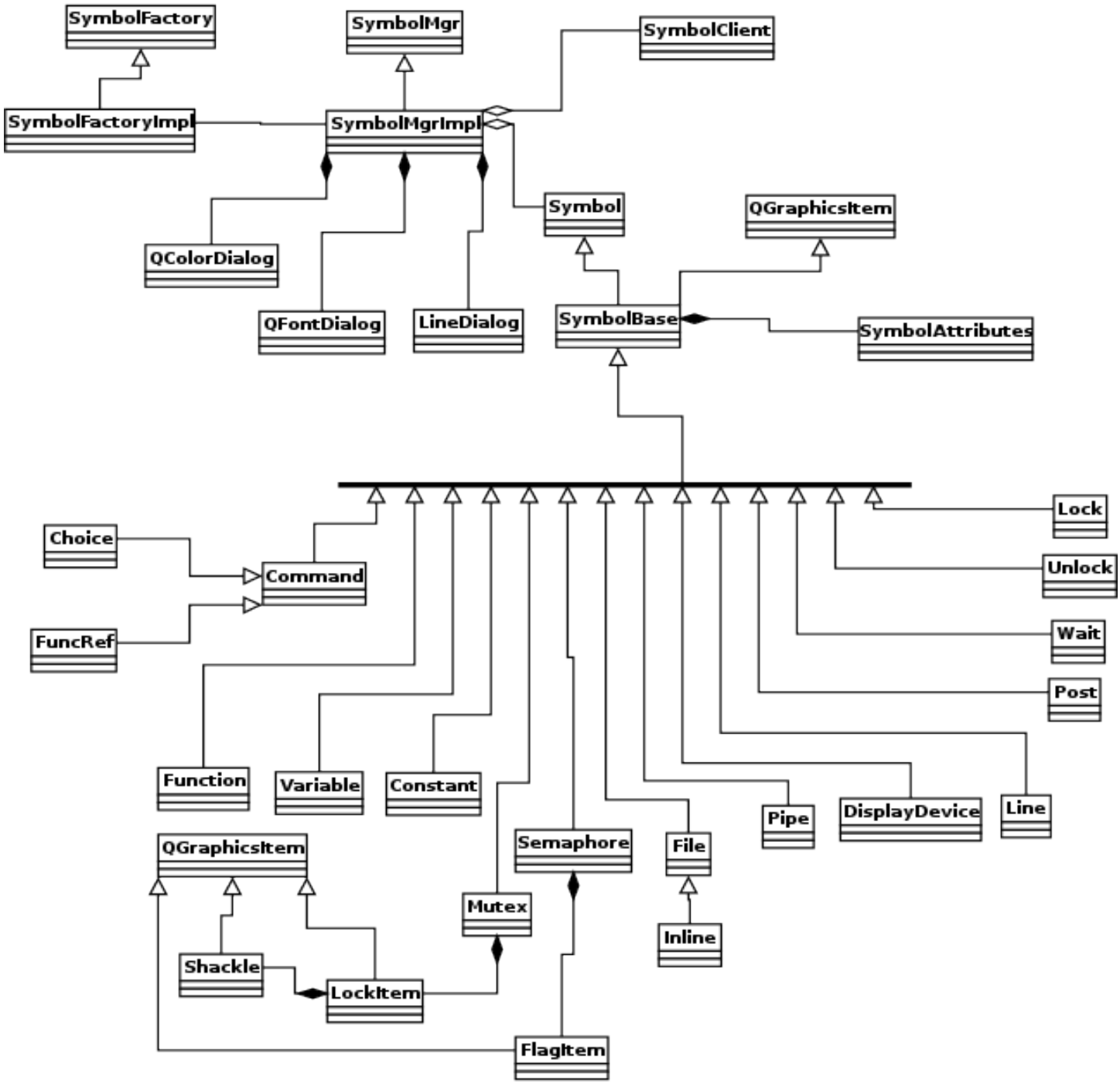


The tool bar at the top allows the user to change the text font, the line style and the colours for lines and symbols. (This will be implemented later.)

The main area displays the symbols. The selected symbol is highlighted with a surrounding dotted line. A tool tip displays the type of the objects:

4 Application Design

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

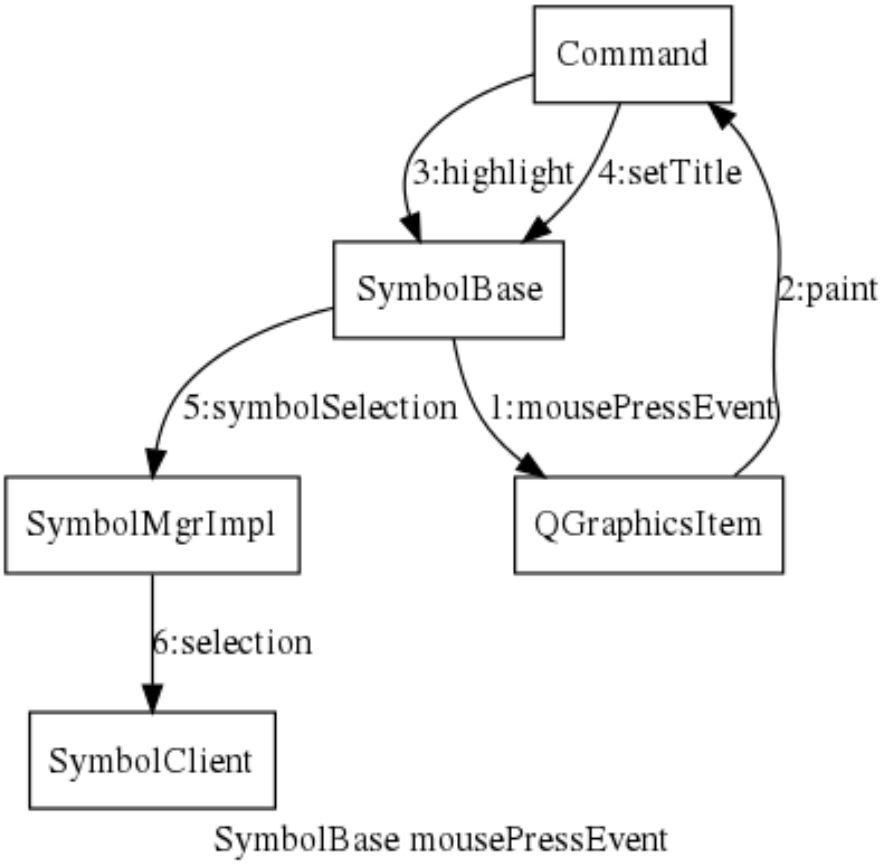


5 Collaboration Diagrams

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

5.1 Select a Symbol

The following sequence occurs when the user selects a symbol on the palette.



SymbolBase mousePressEvent

6 Class Designs

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

6.1 SymbolBase Class

This class provides the common implementation of the Symbol class defined in the interface. It is the parent of each of the actual symbol classes.

```
class SymbolBase : public QObject, public Symbol, public QGraphicsItem
{
	Q_OBJECT
protected:
	SymbolAttributes attribs;
	QPen pen;
	QBrush brush;

	ArgList args;

	void highlight(QPainter *painter, const QStyleOptionGraphicsItem * option);
	void setTitle(QPainter *painter, const QRectF );
	void mousePressEvent ( QGraphicsSceneMouseEvent * event );
public:
	SymbolBase();

	virtual void setAttributes( SymbolAttributes &attr ) { attribs = attr; }
	virtual bool isCommand() { return false; }
	virtual void draw( Scene *, double x, double y, double scale );
	virtual void attachCommand( const ArgList & );

signals:
	void selected( Symbol *);
};
```

6.2 Command Class

This is the symbol for commands.

```
class Command : public SymbolBase
{
private:
	QRectF shape;
public:
	Command();
	Symbol *clone() { return new Command; }
	virtual bool isCommand() { return true; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymCommand; }
};
```

6.3 Choice Class

This is the symbol for commands that have two or more expected exit statuses.

```
class Choice : public Command
{
private:
	QPolygonF shape;
public:
	Choice();
	Symbol *clone() { return new Choice; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymChoice; }
};
```

6.4 FuncRef Class

This is the symbol used to mark a call to a function.

```
class FuncRef : public Command
{
private:
	QRectF shape;
	QLineF lineLeft, lineRight;
public:
	FuncRef();
	Symbol *clone() { return new FuncRef; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymFuncRef; }
};
```

6.5 Function Class

This is the symbol used to start each function.

```
class Function : public SymbolBase
{
private:
	QRectF shape;
	double radius;
public:
	Function();
	Symbol *clone() { return new Function; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymFunc; }
};
```

6.6 Variable Class

This is the symbol used to represent a variable.

```
class Variable : public SymbolBase
{
private:
	QRectF shape;
	QLineF lineLeft, lineTop;
public:
	Variable();
	Symbol *clone() { return new Variable; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymVar; }
};
```

6.7 Constant Class

This is the symbol used to represent a Constant.

```
class Constant : public SymbolBase
{
private:
	QRectF shape;
	QLineF lineLeft, lineTop;
public:
	Constant();
	Symbol *clone() { return new Constant; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymConst; }
};
```

6.8 Shackle Class

This is a small class derived from QGraphicsItem to draw the shackle of a padlock.

```
class Shackle : public QGraphicsItem
{
public:
	Shackle(){}
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
};
```

6.9 LockItem Class

This is a small class derived from QGraphicsItem to draw a padlock.

```
class LockItem : public QGraphicsItem
{
private:
	bool isOpen;
	Shackle *shackle;
	QRectF shape;
public:
	LockItem( bool open );
	void paint( QPainter *painter, const QStyleOptionGraphicsItem *
option, QWidget * widget );
	QRectF boundingRect() const;
};
```

6.10 Mutex Class

This is the symbol for a mutex variable.

```
class Mutex : public SymbolBase
{
private:
	QRectF shape;
	QLineF lineLeft, lineTop;
	LockItem *lock;
public:
	Mutex();
	Symbol *clone() { return new Mutex; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymMutex; }
};
```

6.11 FlagItem Class

This is a small class derived from QGraphicsItem to draw a semaphore flag.

```
class FlagItem : public QGraphicsItem
{
private:
	QPolygonF t1, t2;
public:
	FlagItem();
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
};
```

6.12 Semaphore Class

This is the symbol for a semaphore variable.

```
class Semaphore : public SymbolBase
{
private:
	QRectF shape;
	QLineF lineLeft, lineTop;
	FlagItem *flag1, *flag2;
public:
	Semaphore();
	Symbol *clone() { return new Semaphore; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymSem; }
};
```

6.13 File Class

This is the symbol for a file.

```
class File : public SymbolBase
{
protected:
	QPainterPath path;
public:
	File();
	Symbol *clone() { return new File; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymFile; }
};
```

6.14 Inline Class

This is the symbol for an in-line file.

```
class Inline : public File
{
public:
	Inline();
	Symbol *clone() { return new Inline; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	// QRectF boundingRect() const;
	Style getStyle() { return SymInline; }
};
```

6.15 Pipe Class

This is the symbol for a named pipe.

```
class Pipe : public SymbolBase
{
private:
	QRectF body, leftEllipse, rightEllipse;

public:
	Pipe();
	Symbol *clone() { return new Pipe; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymPipe; }
};
```

6.16 DisplayDevice Class

This is the symbol for output displays. (It was changed from Display since that seems to conflict with an existing item.)

```
class DisplayDevice : public SymbolBase
{
private:
	QPainterPath path;
public:
	DisplayDevice();
	Symbol *clone() { return new DisplayDevice; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymDisplay; }
};
```

6.17 Lock Class

This is the symbol for the mutex locking command.

```
class Lock : public SymbolBase
{
private:
	QRectF shape;
	LockItem *lock;
public:
	Lock();
	Symbol *clone() { return new Lock; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymLock; }
};
```

6.18 Unlock Class

This is the symbol for the mutex unlock command.

```
class Unlock : public SymbolBase
{
private:
	QRectF shape;
	LockItem *lock;
public:
	Unlock();
	Symbol *clone() { return new Unlock; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymUnlock; }
};
```

6.19 Wait Class

This is the symbol for the semaphore wait command.

```
class Wait : public SymbolBase
{
private:
	QPolygonF shape;
	FlagItem *flag;
public:
	Wait();
	Symbol *clone() { return new Wait; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymWait; }
};
```

6.20 Post Class

This is the symbol for the semaphore post command.

```
class Post : public SymbolBase
{
private:
	QRectF shape;
	FlagItem *flag;
public:
	Post();
	Symbol *clone() { return new Post; }
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
	Style getStyle() { return SymPost; }
};
```

6.21 Line Class

This is the class for all the lines. It uses the style to determine which pen to use.

```
class Line : public SymbolBase
{
protected:
	Style style;
public:
	Line( Style );
	Symbol *clone() { return new Line(style); }
	Style getStyle() { return style; }
	// virtual void draw( Scene *, double x, double y, double scale );
	void paint( QPainter *painter, const QStyleOptionGraphicsItem * option,
		QWidget * widget );
	QRectF boundingRect() const;
};
```

6.22 LineDialog Class

This presents a dialog allowing the user to select a new line style.

```
class LineDialog : public QDialog
{
private:
	QComboBox *styleSelector;
public:
	LineDialog( QWidget *parent);
	~LineDialog();
};
```

6.23 SymbolFactoryImpl Class

This is responsible for creating an instance of the SymbolMgrImpl.

```
class SymbolFactoryImpl : public SymbolFactory
{
public:
	SymbolFactoryImpl(){}
	SymbolMgr * makeSymbolMgr( Window * );
};
```

6.24 SymbolMgrImpl Class

This class implements the interface for the symbol library.

```
class SymbolMgrImpl : public QObject, public SymbolMgr
{
      Q_OBJECT
private:
      std::vector< SymbolClient * > clients;
      std::vector< Symbol * > symbols;
      GWindow *window;
      QToolBar * symbolToolBar;
      QGraphicsView * view;
      CanvasScene *scene;
      QAction *fontAction;
      QAction *lineAction;
      QAction *colourAction;

      QFontDialog *fontDialog;
      QColorDialog *colourDialog;
      LineDialog *lineDialog;

      void makeToolBar();
      void showSymbols();
      void setupSymbol( SymbolBase *, double x, double y);
private slots:
      void doFont();
      void doLine();
      void doColour();
      void symbolSelection( Symbol *);
public:
      SymbolMgrImpl( Window *);

      virtual void addClient( SymbolClient * );

      virtual SymbolAttributes getDefaultAttr();
      virtual TextAttributes   getDefaultTextAttr();
};
```

Appendix A