Beginner's Guide to scintilla: Syntax Highlighting & Code Folding for *var'aq*

Andreas Tscharner

April 18, 2014

Abstract

Syntax Highlighting and Code Folding in Scintilla explained by the Klingon scripting language *var'aq*

Contents

1 Introduction

1.1 Yet another tutorial?

There are already three links on the documentation page of scintilla that describe lexing and code folding in the scintilla component. So why another one?

When I started using scintilla I read those documents, as well as the *excellent* documentation of the scintilla "functions". I also had a look at other lexers of languages I knew. All these helped *a little*. But what I really missed was a guide that took me step by step into the mysteries of lexing and code folding.

So, yes. This is yet another tutorial, but one that goes step by step.

1.2 What is that var'aq thing?

Question 1 in the FAQ: What is Var'aq?

Var'aq is sort of a fanfic programming language based on the Klingon language used on the Star Trek television series and movies. Klingon, created by Marc Okrand and "maintained", in a sense, by the Klingon Language Institute independently of Paramount's auspices, is really its own separate deal; this is merely a fan's attempt to give a little more richness to the culture as well as exercising a love of languages and a desire to learn more Perl.

1.2.1 Why a Klingon programming language?

A few reasons; the first and most important: There was no lexer for that language yet.

Furthermore I always liked slightly obscure languages and I am a fan of StarTrek so this was the "logical" choice.

1.2.2 Links for more information

- The official var'aq webpage: http://www.reocities.com/connorbd/varaq/
- An introduction to var'aq: https://www.mjstemen.com/varaq_site/varaq_main.html

1.3 Organisational Stuff

1.3.1 Latest version of this tutorial

You'll find the latest version of this tutorial on Bitbucket

1.3.2 Bugs, mistakes and typos

If you find any bugs in the code or mistakes and typos in the text, please let me know.

You can either open a new issue in the Bitbucket issue tracker or send me an e-mail. Patches are welcome!

2 Preparations

2.1 How syntax highlighting is done in scintilla

The so called "Lexer" is parsing the given document. If it finds something that could be highlighted it sets a pre-defined style.

For example: if the C/C++ lexer finds a number it sets for each position of the number the `SCE_C_NUMBER` style.

It is up to the controlling application to set a color for a given style.

2.2 How the files are organized

All "Lexer" files are stored in the `lexers` subdirectory. They start with `Lex` and the name of the programming language they are responsible for. So our var'aq lexer will be called: `LexVaraq.cxx`.

The different styles for all languages are stored in `SciLexer.h` in the `include` subdirectory. This file is automatically generated. We have to insert the values for our styles in the `Scintilla.iface` file which is also in the `include` directory.

Finally we have to regenerate some Makefiles (or parts of it), header files etc. by running some Python scripts (they can be found in the `scripts`) directory.

2.3 Setting up the styles

The var'aq lexer will have four styles: *Comments*, *Strings*, *Numbers* and *Keywords*. Along with them we also need to define a style for the standard/default text.

So we have to define styles (constants) in `Scintilla.iface` and a constant for the new language as well.

A new language is simple:

```
val SCLEX_VARAQ=114
```

right after the line for the *Unix Assembler*: `val SCLEX_AS=113`

We define the constants for our new styles after the constants for *DMAP*[1] (after `val SCE_DMAP_WORD3=10`; usually we would do that after the *Unix Assembler*, but in this case, the new assembler lexing is just a variation of an already existing):

```
# Lexical states for SCLEX_VARAQ
lex Varaq=SCLEX_VARAQ SCE_VARAQ_
val SCE_VARAQ_DEFAULT=0
val SCE_VARAQ_COMMENT=1
val SCE_VARAQ_STRING=2
val SCE_VARAQ_NUMBER=3
val SCE_VARAQ_IDENTIFIER=4
val SCE_VARAQ_KEYWORD=5
```

The `SCE_VARAQ_IDENTIFIER` is a special case. It is (internally) used to match any possible identifier of the language. This is necessary if we have the final word later and cannot decide at the beginning of the word.

[1]Generally speaking: The constant for the new language and the constants for the different styles within this new language are simply written at the end of the current section

After having run `HFacer.py` from the `scripts` directory we notice that the updated `include\SciLexer.h` contains our newly defined constants.

3 Lexer

After having defined our styles we have to implement our var'aq lexer. To do so we have to implement the `ILexer` interface or the `ILexerWithSubStyles` interface. Both are defined in `ILexer.h` and `ILexerWithSubStyles` is derived from `ILexer`.

3.1 ILexer and ILexerWithSubStyles interface

3.1.1 ILexer

```
class ILexer {
public:
    virtual int SCI_METHOD Version() const = 0;
    virtual void SCI_METHOD Release() = 0;
    virtual const char * SCI_METHOD PropertyNames() = 0;
    virtual int SCI_METHOD PropertyType(const char *name) = 0;
    virtual const char * SCI_METHOD DescribeProperty(const char *name) = 0;
    virtual int SCI_METHOD PropertySet(const char *key, const char *val) = 0;
    virtual const char * SCI_METHOD DescribeWordListSets() = 0;
    virtual int SCI_METHOD WordListSet(int n, const char *wl) = 0;
    virtual void SCI_METHOD Lex(unsigned int startPos, int lengthDoc,
          int initStyle, IDocument *pAccess) = 0;
    virtual void SCI_METHOD Fold(unsigned int startPos, int lengthDoc,
          int initStyle, IDocument *pAccess) = 0;
    virtual void * SCI_METHOD PrivateCall(int operation, void *pointer) = 0;
};
```

3.1.2 ILexerWithSubStyles

```
class ILexerWithSubStyles : public ILexer {
public:
    virtual int SCI_METHOD LineEndTypesSupported() = 0;
    virtual int SCI_METHOD AllocateSubStyles(int styleBase, int numberStyles) = 0;
    virtual int SCI_METHOD SubStylesStart(int styleBase) = 0;
    virtual int SCI_METHOD SubStylesLength(int styleBase) = 0;
    virtual void SCI_METHOD FreeSubStyles() = 0;
    virtual void SCI_METHOD SetIdentifiers(int style, const char *identifiers) = 0;
    virtual int SCI_METHOD DistanceToSecondaryStyles() = 0;
    virtual const char * SCI_METHOD GetSubStyleBases() = 0;
};
```

3.2 Properties

We notice a few *Property* methods in the `ILexer` interface. Scintilla allows to define properties which can be set/unset while the application is running.

We don't need them at the moment and don't implement the corresponding methods.

3.3 Creating the lexer file

As mentioned above our file will be called `LexVaraq.cxx` and will be placed in the `lexers` directory as all the other lexers.

We start by implementing the `ILexer` interface. There is no problem in changing to the `ILexerWithSubStyles` interface later if we need it as it is derived from `ILexer`.

The selection of the implemented interface also defines the return value of the virtual int SCI_METHOD Version() method (see below). We return `lvOriginal` for the `ILexer` interface and `lvSubStyles` for `ILexerWithSubStyles`.

We start with the absolute minimum:

```
// Scintilla source code edit control
/** @file LexVaraq.cxx
 ** Lexer for the klingon scripting language var'aq
 ** Used for the Scintilla-var'aq-Tutorial
 **/
// Copyright 2013-2014 by Andreas Tscharner <andy@vis.ethz.ch>
// The License.txt file describes the conditions under which this
// software may be distributed.


#include "ILexer.h"


#ifdef SCI_NAMESPACE
using namespace Scintilla;
#endif

class LexerVaraq : public ILexer
{
};
```

3.4 The methods of the interface

Most methods of the interface are straight forward to implement. We discuss them here shortly.

- virtual int SCI_METHOD Version()

  The virtual int SCI_METHOD Version() method defines which version of the interface should be used. For the ILexer interface lvOriginal should be returned, for ILexerWithSubStyles the value lvSubStyles is expected.

- virtual void SCI_METHOD Release()

- static ILexer *LexerFactoryVaraq()

  virtual void SCI_METHOD Release() is called to destroy the lexer object that was created by the factory method static ILexer *LexerFactoryVaraq()

- virtual void SCI_METHOD Lex(...)

  We ignore the `property` and `WordList` methods for the moment and have a look at the most important method: virtual void SCI_METHOD Lex(...). The `Lex` method has to be implemented for syntax highlighting.

- virtual void SCI_METHOD Fold(...)

  The last method is important as well (later on in this tutorial): virtual void SCI_METHOD Fold(...) for the so called *Code Folding*

So the first code in our class looks like the following:

```
class LexerVaraq : public ILexer
{
        public:
                LexerVaraq() {}
                virtual ~LexerVaraq() {}

                int SCI_METHOD Version() const {
                        return lvOriginal;
                }
                void SCI_METHOD Release() {
                        delete this;
                }

                const char * SCI_METHOD PropertyNames() {
                        return NULL;
                }
                int SCI_METHOD PropertyType(const char *name) {
                        return -1;
                }
                const char * SCI_METHOD DescribeProperty(const char *name) {
                        return NULL;
                }
                int SCI_METHOD PropertySet(const char *key, const char *val) {
                        return -1;
                }

                const char * SCI_METHOD DescribeWordListSets() {
                        return NULL;
                }
                int SCI_METHOD WordListSet(int n, const char *wl) {
                        return -1;
                }

                void SCI_METHOD Lex(unsigned int startPos, int lengthDoc,
                  int initStyle, IDocument *pAccess);
                void SCI_METHOD Fold(unsigned int startPos, int lengthDoc,
                  int initStyle, IDocument *pAccess) {}

                void * SCI_METHOD PrivateCall(int operation, void *pointer) {
                        return NULL;
                }

                static ILexer *LexerFactoryVaraq() {
                        return new LexerVaraq();
                }
};
```

3.5 Comments

Comments in var'aq are like the old style Pascal comments: The comment is enclosed in `(*` and `*)`.

So we start to find such tokens in the document and add the COMMENT style to the range from start to the end.

```
void SCI_METHOD LexerVaraq::Lex(unsigned int startPos, int lengthDoc,
  int initStyle, IDocument *pAccess)
{
        LexAccessor styler(pAccess);
        StyleContext scCTX(startPos, lengthDoc, initStyle, styler);


        for (; scCTX.More() ; scCTX.Forward()) {
                switch (scCTX.state) {
                        case SCE_VARAQ_DEFAULT :
                                if (scCTX.Match('(', '*')) {
                                        scCTX.SetState(SCE_VARAQ_COMMENT);
                                        scCTX.Forward();   // Move over the *
                                };
                                break;

                        case SCE_VARAQ_COMMENT :
                                if (scCTX.Match('*', ')')) {
                                        scCTX.Forward();   // Move over the )
                                        scCTX.ForwardSetState(SCE_VARAQ_DEFAULT);
                                };
                                break;
                };
        };
        scCTX.Complete();
}
```

These few lines are already enough to highlight the comment in a var'aq program. Let's have a closer look at them.

The `styler` of type `LexAccessor` provides access to the document with several useful functions like `GetLine` or `GetLineState`.

With the StyleContext scCTX variable we get the context of the style.

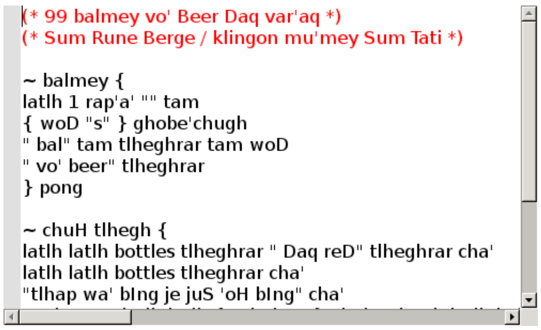
Now we iterate through the defined range:
for (; scCTX.More() ; scCTX.Forward())
and check each character. If we find the start of a comment and are in the default style, we set the new `COMMENT` style. On the other hand if we are in the comment style and find the end of the comment, we return to the default style.

To use the lexer we need to create an instance of a `LexerModule`. This is the last line in the file:

```
LexerModule lmVaraq(SCLEX_VARAQ, LexerVaraq::LexerFactoryVaraq, "varaq", VaraqWordListDesc);
```

This `LexerModule` is added to `Catalogue.cxx` using the `LexGen.py` script in the corresponding directory.

Now if we load a *var'aq* program into scintilla the comment is already higlighted:



```
(* 99 balmey vo' Beer Daq var'aq *)
(* Sum Rune Berge / klingon mu'mey Sum Tati *)

~ balmey {
latlh 1 rap'a' "" tam
{ woD "s" } ghobe'chugh
" bal" tam tlheghrar tam woD
" vo' beer" tlheghrar
} pong

~ chuH tlhegh {
latlh latlh bottles tlheghrar " Daq reD" tlheghrar cha'
latlh latlh bottles tlheghrar cha'
"tlhap wa' bIng je juS 'oH bIng" cha'
```

The application defines the colors of the highlighted text. To show the styled comment we've used red, but later the comment will become grey.

3.6 Strings

Strings in *var'aq* are marked as in many other languages with double quotes.

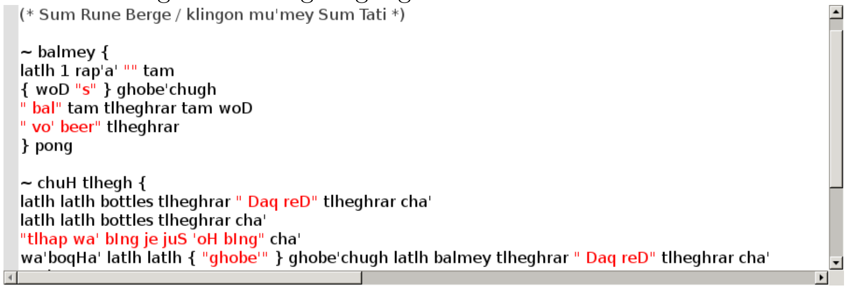
Therefore the string highlighting is simple: we add a check in the *var'aq* default block to switch to *var'aq* string style:

```
if (scCTX.Match('\"')) {
        scCTX.SetState(SCE_VARAQ_STRING);
        break;
};
```

and we add a new block in case of a string and switch back to the default style:

```
case SCE_VARAQ_STRING :
        if (scCTX.Match('\"')) {
                scCTX.ForwardSetState(SCE_VARAQ_DEFAULT);
        };
        break;
```

That's all to get the strings highlighted:



```
(* Sum Rune Berge / klingon mu'mey Sum Tati *)

~ balmey {
latlh 1 rap'a' "" tam
{ woD "s" } ghobe'chugh
" bal" tam tlheghrar tam woD
" vo' beer" tlheghrar
} pong

~ chuH tlhegh {
latlh latlh bottles tlheghrar " Daq reD" tlheghrar cha'
latlh latlh bottles tlheghrar cha'
"tlhap wa' bIng je juS 'oH bIng" cha'
wa'boqHa' latlh latlh { "ghobe'" } ghobe'chugh latlh balmey tlheghrar " Daq reD" tlheghrar cha'
```

3.7 Numbers

For a first simple check for numbers, we check for a starting digit or negative sign, more digits and optionally one point as a decimal separator.

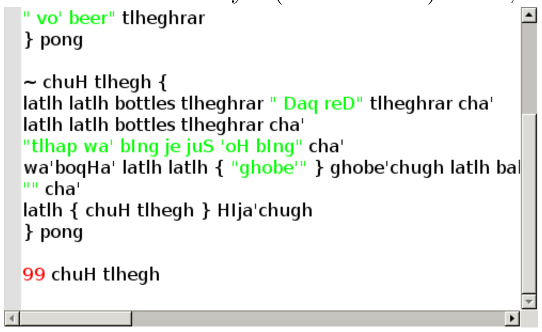
To start with the SCE_VARAQ_NUMBER style we check for a negative sign followed by a digit or just for a digit:

```
if (IsADigit(scCTX.ch) || (scCTX.Match('-') && IsADigit(scCTX.chNext))) {
        scCTX.SetState(SCE_VARAQ_NUMBER);
        pointInNumberUsed = false;
        break;
};
```

We can keep the number style as long as digits follow or at maximum one point as a decimal separator. pointInNumberUsed is a variable of type bool:

```
case SCE_VARAQ_NUMBER :
        if (scCTX.Match('.')) {
                if (pointInNumberUsed) {
                        scCTX.SetState(SCE_VARAQ_DEFAULT);
                } else {
                        pointInNumberUsed = true;
                };
        } else {
                if (!IsADigit(scCTX.ch)) {
                        scCTX.SetState(SCE_VARAQ_DEFAULT);
                };
        };
        break;
```

As usual the new style (the numbers) is red; the strings have become green:



```
" vo' beer" tlheghrar
} pong

~ chuH tlhegh {
latlh latlh bottles tlheghrar " Daq reD" tlheghrar cha'
latlh latlh bottles tlheghrar cha'
"tlhap wa' bIng je juS 'oH bIng" cha'
wa'boqHa' latlh latlh { "ghobe'" } ghobe'chugh latlh bal
"" cha'
latlh { chuH tlhegh } HIja'chugh
} pong

99 chuH tlhegh
```

3.8 Keywords

At last we want to highlight the keywords of *var'aq*. Checking for these words works a little different than lexing the other parts of the syntax. First, let's define these words:

```
static const char *const VaraqKeywords = {
  "woD␣latlh␣tam␣chImmoH␣qaw␣qawHa\'␣Hotlh␣pong␣cher␣HIja\'chugh␣ghobe\'chugh␣wIv"
  "chov␣nargh␣vangqa\'␣SIj␣muv␣ghorqu\'␣chIm\'a\'␣tlheghrar␣naQmoH␣tlheghrap\'a\'"
  "tlheghpe\'␣tlheghjuv␣jor␣boq␣boqHa\'␣boq\'egh␣boqHa\'\'egh␣HabboqHa\'\'egh"
  "chuv␣boqHa\'qa\'␣loS\'ar␣wa\'boq␣wa\'boqHa\'␣joq␣joqHa\'␣qojmI\'␣qojHa\'"
  "ghurtaH␣maHghurtaH␣wejghurtaH␣poD␣Hab␣\'ar␣mIScher␣mIS␣HeHmI\'␣ghurmI\'"
  "HabmI\'\'a\'␣mI\'\'a\'␣mI\'moH␣mobmoH␣DuD␣tlhoch␣Qo\'moH␣nIHghoS␣poSghoS"
  "law\'\'a\'␣puS\'a\'␣rap\'a\'␣law\'rap\'a\'␣puSrap\'a\'␣rapbe\'a\'␣pagh\'a\'"
  "taH\'a\'␣je␣joq␣ghap␣ghobe\'␣cha\'␣\'Ij␣bep␣chu\'DonwI\'␣chu\'tut␣nuqDaq_jIH"
  "pongmI\'␣taghDe\'"
};
```

Scintilla has a WordList class that allows simple checking for these words in that one big string:

```
        WordList vqKeywords;
        vqKeywords.Set(VaraqKeywords);
```

We will use these keywords later to check if the found word is a *var'aq* keyword that has to be styled as `VARAQ_KEYWORD`.

We also need a character set which contains all possible characters a keyword can have. If we look at the keywords above these characters are all letters (upper and lower case) and the single quote.

```
        CharacterSet setKeywordChars(CharacterSet::setAlpha, "\'");
```

If we find one of these characters the next word is possible a keyword (but it might be something else as well), so we set the state to our helping state SCE_VARAQ_IDENTIFIER:

```
switch (scCTX.state) {
        case SCE_VARAQ_DEFAULT :
                if (setKeywordChars.Contains(scCTX.ch)) {
                        scCTX.SetState(SCE_VARAQ_IDENTIFIER);
                        break;
                };
};
```

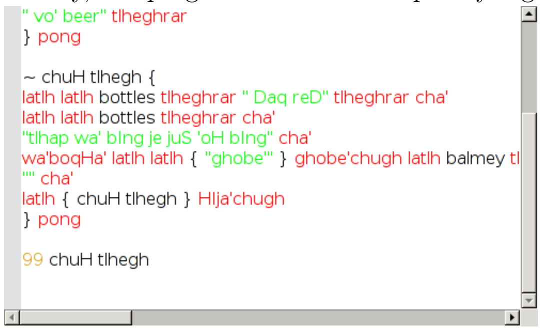
Now all possible words have the SCE_VARAQ_IDENTIFIER state. If we are in this state and reach a character that is not part of a keyword, we have to handle the state. If the word defined as SCE_VARAQ_IDENTIFIER is a *var'aq* keyword, we change the state to SCE_VARAQ_KEYWORD. If it is not, we don't do anything. In both cases, we have to set the default state again.

```
case SCE_VARAQ_IDENTIFIER :
        if (!setKeywordChars.Contains(scCTX.ch)) {
                tmpStr = new char[MAX_STR_LEN];
                memset(tmpStr, 0, sizeof(char)*MAX_STR_LEN);
                scCTX.GetCurrent(tmpStr, MAX_STR_LEN);

                if (vqKeywords.InList(tmpStr)) {
                        scCTX.ChangeState(SCE_VARAQ_KEYWORD);
                };
                scCTX.SetState(SCE_VARAQ_DEFAULT);

                delete[] tmpStr;
        };
        break;
```

Finally, our program is almost completely highlighted (keywords are red):



```
" vo' beer" tlheghrar
} pong

~ chuH tlhegh {
latlh latlh bottles tlheghrar " Daq reD" tlheghrar cha'
latlh latlh bottles tlheghrar cha'
"tlhap wa' bIng je juS 'oH bIng" cha'
wa'boqHa' latlh latlh { "ghobe'" } ghobe'chugh latlh balmey tl
"" cha'
latlh { chuH tlhegh } HIja'chugh
} pong

99 chuH tlhegh
```

4 Code Folding

Code folding is done a similar way like lexing: Stepping through the requested area character by character and if a starting character for code folding is found, the level is incremented, if a closing character is found, it is decremented.

In *var'aq* the characters for code folding are (like in C/C++) the curly braces ({ and }). The code is a little bit different as there is no StyleContext this time.

```
void SCI_METHOD LexerVaraq::Fold(unsigned int startPos, int lengthDoc,
  int initStyle, IDocument *pAccess)
{
        LexAccessor styler(pAccess);
        unsigned int endPos = startPos + lengthDoc;
        char chNext = styler[startPos];
        int lineCurrent = styler.GetLine(startPos);
        int levelPrev = styler.LevelAt(lineCurrent) & SC_FOLDLEVELNUMBERMASK;
        int levelCurrent = levelPrev;
        char ch;
        bool atEOL;


        for (unsigned int i = startPos; i < endPos; i++) {
                ch = chNext;
                chNext = styler.SafeGetCharAt(i+1);

                atEOL = ((ch == '\r' && chNext != '\n') || (ch == '\n'));

                if (ch == '{') {
                        levelCurrent++;
                };
                if (ch == '}') {
                        levelCurrent--;
                };

                if (atEOL || (i == (endPos-1))) {
                        int lev = levelPrev;

                        if (levelCurrent > levelPrev) {
                                lev |= SC_FOLDLEVELHEADERFLAG;
                        };

                        if (lev != styler.LevelAt(lineCurrent)) {
                                styler.SetLevel(lineCurrent, lev);
                        };
                        lineCurrent++;
                        levelPrev = levelCurrent;
                };
        };
}
```

The implementation is straight forward: We iterate from start to the end. Each character is tested: if it is a opening curly brace, the level is incremented; if it is a closing curly brace, it is decremented again. At the end of every line, the current level is checked and if it is greater than the level in the line before, the SC_FOLDLEVELHEADERFLAG is set. This is an indicator for Scintilla that on this line is a folding point.

What is left now? We also have to set up some stuff in the controlling application:

```
#define MARGIN_SCRIPT_FOLD_INDEX 1


SSM(SCI_SETMARGINTYPEN, MARGIN_SCRIPT_FOLD_INDEX, SC_MARGIN_SYMBOL);
SSM(SCI_SETMARGINMASKN, MARGIN_SCRIPT_FOLD_INDEX, SC_MASK_FOLDERS);
SSM(SCI_SETMARGINWIDTHN, MARGIN_SCRIPT_FOLD_INDEX, 20);
SSM(SCI_SETMARGINSENSITIVEN, MARGIN_SCRIPT_FOLD_INDEX, 1);

SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDER, SC_MARK_BOXPLUS);
SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDEROPEN, SC_MARK_BOXMINUS);
SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDEREND, SC_MARK_EMPTY);
SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDERMIDTAIL, SC_MARK_EMPTY);
SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDEROPENMID, SC_MARK_EMPTY);
SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDERSUB, SC_MARK_EMPTY);
SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDERTAIL, SC_MARK_EMPTY);

SSM(SCI_SETFOLDFLAGS, SC_FOLDFLAG_LINEAFTER_CONTRACTED, 0);
```

These calls set up a margin for symbols which is 20 pixels wide, and defines a few (internally stored) symbols for the folding points.

The SCI_SETMARGINSENSITIVEN call also sets the margin symbols for the given margins (MARGIN_SCRIPT_FOLD_INDEX) sensitive for mouse clicks. Therefore a callback function for Scintilla notifications has to be set up...

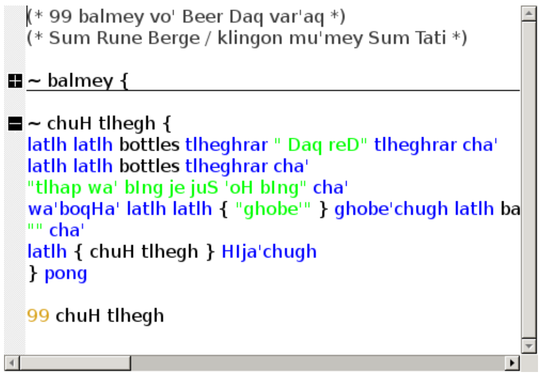
```
static void scintilla_notify(GtkWidget *sciWidget, gint ctrlID,
  struct SCNotification *notifyData, gpointer userData)
{
        int lineNr = scintilla_send_message(SCINTILLA(sciWidget),
                SCI_LINEFROMPOSITION, (uptr_t)notifyData->position, 0);


        switch (notifyData->margin) {
                case MARGIN_SCRIPT_FOLD_INDEX:
                        scintilla_send_message(SCINTILLA(sciWidget),
                          SCI_TOGGLEFOLD, lineNr, 0);
                        break;
        };
}
```

... and connected:

```
gtk_signal_connect(GTK_OBJECT(editor), "sci-notify",
  G_CALLBACK(scintilla_notify), editor);
```

The result is here (the balmey function has been folded):



```
(* 99 balmey vo' Beer Daq var'aq *)
(* Sum Rune Berge / klingon mu'mey Sum Tati *)

~ balmey {

~ chuH tlhegh {
latlh latlh bottles tlheghrar " Daq reD" tlheghrar cha'
latlh latlh bottles tlheghrar cha'
"tlhap wa' bIng je juS 'oH bIng" cha'
wa'boqHa' latlh latlh { "ghobe'" } ghobe'chugh latlh ba
"" cha'
latlh { chuH tlhegh } HIja'chugh
} pong

99 chuH tlhegh
```

A The complete *var'aq* lexing and folding code

```
// Scintilla source code edit control
/** @file LexVaraq.cxx
 ** Lexer for the klingon scripting language var'aq
 ** Used for the Scintilla-var'aq-Tutorial
 **/
// Copyright 2013 by Andreas Tscharner <andy@vis.ethz.ch>
// The License.txt file describes the conditions under which this software may be distributed.


#include <cstdlib>
#include <cassert>
#include <cstring>

#include "ILexer.h"
#include "Scintilla.h"
#include "SciLexer.h"

#include "LexAccessor.h"
#include "StyleContext.h"
#include "LexerModule.h"
#include "CharacterSet.h"
#include "WordList.h"


#ifdef SCI_NAMESPACE
using namespace Scintilla;
#endif


static const char *const VaraqWordListDesc[] = {
        "Var\'aq␣keywords",
        0
};

static const char *const VaraqKeywords = {
        "woD␣latlh␣tam␣chImmoH␣qaw␣qawHa\'␣Hotlh␣pong␣cher␣HIja\'chugh␣ghobe\'chugh␣wIv␣"
        "chov␣nargh␣vangqa\'␣SIj␣muv␣ghorqu\'␣chIm\'a\'␣tlheghrar␣naQmoH␣tlheghrap\'a\'␣"
        "tlheghpe\'␣tlheghjuv␣jor␣boq␣boqHa\'␣boq\'egh␣boqHa\'\'egh␣HabboqHa\'\'egh␣"
        "chuv␣boqHa\'qa\'␣loS\'ar␣wa\'boq␣wa\'boqHa\'␣joq␣joqHa\'␣qojmI\'␣qojHa\'␣"
        "ghurtaH␣maHghurtaH␣wejghurtaH␣poD␣Hab␣\'ar␣mIScher␣mIS␣HeHmI\'␣ghurmI\'␣"
        "HabmI\'\'a\'␣mI\'\'a\'␣mI\'moH␣mobmoH␣DuD␣tlhoch␣Qo\'moH␣nIHghoS␣poSghoS␣"
        "law\'\'a\'␣puS\'a\'␣rap\'a\'␣law\'rap\'a\'␣puSrap\'a\'␣rapbe\'a\'␣pagh\'a\'␣"
        "taH\'a\'␣je␣joq␣ghap␣ghobe\'␣cha\'␣\'Ij␣bep␣chu\'DonwI\'␣chu\'tut␣nuqDaq_jIH␣"
        "pongmI\'␣taghDe\'"
};


class LexerVaraq : public ILexer
{
        public:
                LexerVaraq() {}
                virtual ~LexerVaraq() {}

                int SCI_METHOD Version() const {
                        return 1;
                }
                void SCI_METHOD Release() {
                        delete this;
                }

                const char * SCI_METHOD PropertyNames() {
                        return NULL;
                }
                int SCI_METHOD PropertyType(const char *name) {
                        return -1;
                }
                const char * SCI_METHOD DescribeProperty(const char *name) {
                        return NULL;
                }
                int SCI_METHOD PropertySet(const char *key, const char *val) {
                        return -1;
                }

                const char * SCI_METHOD DescribeWordListSets() {
                        return VaraqWordListDesc[0];
                }
                int SCI_METHOD WordListSet(int n, const char *wl) {
                        return -1;
                }

                void SCI_METHOD Lex(unsigned int startPos, int lengthDoc, int initStyle, IDocument *pAccess);
                void SCI_METHOD Fold(unsigned int startPos, int lengthDoc, int initStyle, IDocument *pAccess);

                void * SCI_METHOD PrivateCall(int operation, void *pointer) {
                        return NULL;
                }

                static ILexer *LexerFactoryVaraq() {
                        return new LexerVaraq();
                }
};


void SCI_METHOD LexerVaraq::Lex(unsigned int startPos, int lengthDoc, int initStyle, IDocument *pAccess)
{
        const unsigned int MAX_STR_LEN = 100;


        LexAccessor styler(pAccess);
        StyleContext scCTX(startPos, lengthDoc, initStyle, styler);
        bool pointInNumberUsed = false;
        WordList vqKeywords;
        char *tmpStr;
        CharacterSet setKeywordChars(CharacterSet::setAlpha, "\'");


        vqKeywords.Set(VaraqKeywords);
        tmpStr = new char[MAX_STR_LEN];

        for (; scCTX.More() ; scCTX.Forward()) {
                switch (scCTX.state) {
                        case SCE_VARAQ_DEFAULT :
                                if (scCTX.Match('(', '*')) {
                                        scCTX.SetState(SCE_VARAQ_COMMENT);
                                        scCTX.Forward();
                                        break;
                                };
                                if (scCTX.Match('\"')) {
                                        scCTX.SetState(SCE_VARAQ_STRING);
                                        break;
                                };
                                if (IsADigit(scCTX.ch) || (scCTX.Match('-') && IsADigit(scCTX.chNext))) {
                                        scCTX.SetState(SCE_VARAQ_NUMBER);
                                        pointInNumberUsed = false;
                                        break;
                                };
                                if (setKeywordChars.Contains(scCTX.ch)) {
                                        scCTX.SetState(SCE_VARAQ_IDENTIFIER);
                                        break;
                                };
                                break;

                        case SCE_VARAQ_COMMENT :
                                if (scCTX.Match('*', ')')) {
                                        scCTX.Forward();
                                        scCTX.ForwardSetState(SCE_VARAQ_DEFAULT);
                                };
                                break;

                        case SCE_VARAQ_STRING :
                                if (scCTX.Match('\"')) {
                                        scCTX.ForwardSetState(SCE_VARAQ_DEFAULT);
                                };
                                break;

                        case SCE_VARAQ_NUMBER :
                                if (scCTX.Match('.')) {
                                        if (pointInNumberUsed) {
                                                scCTX.SetState(SCE_VARAQ_DEFAULT);
                                        } else {
                                                pointInNumberUsed = true;
                                        };
                                } else {
                                        if (!IsADigit(scCTX.ch)) {
                                                scCTX.SetState(SCE_VARAQ_DEFAULT);
                                        };
                                };
                                break;

                        case SCE_VARAQ_IDENTIFIER :
                                if (!setKeywordChars.Contains(scCTX.ch)) {
                                        memset(tmpStr, 0, sizeof(char)*MAX_STR_LEN);
                                        scCTX.GetCurrent(tmpStr, MAX_STR_LEN);

                                        if (vqKeywords.InList(tmpStr)) {
                                                scCTX.ChangeState(SCE_VARAQ_KEYWORD);
                                        };
                                        scCTX.SetState(SCE_VARAQ_DEFAULT);
                                };
                                break;
                };
        };
        scCTX.Complete();

        delete[] tmpStr;
}

void SCI_METHOD LexerVaraq::Fold(unsigned int startPos, int lengthDoc, int initStyle, IDocument *pAccess)
{
        LexAccessor styler(pAccess);
        unsigned int endPos = startPos + lengthDoc;
        char chNext = styler[startPos];
        int lineCurrent = styler.GetLine(startPos);
        int levelPrev = styler.LevelAt(lineCurrent) & SC_FOLDLEVELNUMBERMASK;
        int levelCurrent = levelPrev;
        char ch;
        bool atEOL;


        for (unsigned int i = startPos; i < endPos; i++) {
                ch = chNext;
                chNext = styler.SafeGetCharAt(i+1);

                atEOL = ((ch == '\r' && chNext != '\n') || (ch == '\n'));
                if (ch == '{') {
                        levelCurrent++;
                };
                if (ch == '}') {
                        levelCurrent--;
                };

                if (atEOL || (i == (endPos-1))) {
                        int lev = levelPrev;

                        if (levelCurrent > levelPrev) {
                                lev |= SC_FOLDLEVELHEADERFLAG;
                        };

                        if (lev != styler.LevelAt(lineCurrent)) {
                                styler.SetLevel(lineCurrent, lev);
                        };
                        lineCurrent++;
                        levelPrev = levelCurrent;
                };
        };
}


LexerModule lmVaraq(SCLEX_VARAQ, LexerVaraq::LexerFactoryVaraq, "varaq", VaraqWordListDesc);
```

B Sample GTK code for displaying the scintilla window

This small application is an adapted version of the sample application from the Scintilla page `bait.c`.

```
#include <gtk/gtk.h>

#include <Scintilla.h>
#include <SciLexer.h>
#define PLAT_GTK 1
#include <ScintillaWidget.h>

#define MARGIN_SCRIPT_FOLD_INDEX 1


static int exit_app(GtkWidget*w, GdkEventAny*e, gpointer p) {
        gtk_main_quit();
        return w||e||p||1;      // Avoid warnings
}

static void scintilla_notify(GtkWidget *sciWidget, gint ctrlID, struct SCNotification *notifyData, gpointer userData)
{
        int lineNr = scintilla_send_message(SCINTILLA(sciWidget), SCI_LINEFROMPOSITION, (uptr_t)notifyData->position, 0);


        switch (notifyData->margin) {
                case MARGIN_SCRIPT_FOLD_INDEX:
                        scintilla_send_message(SCINTILLA(sciWidget), SCI_TOGGLEFOLD, lineNr, 0);
                        break;
        };
}

int main(int argc, char **argv) {
        GtkWidget *app;
        GtkWidget *editor;
        ScintillaObject *sci;

        gtk_init(&argc, &argv);
        app = gtk_window_new(GTK_WINDOW_TOPLEVEL);
        editor = scintilla_new();
        sci = SCINTILLA(editor);

        gtk_container_add(GTK_CONTAINER(app), editor);
        gtk_signal_connect(GTK_OBJECT(app), "delete_event",
          GTK_SIGNAL_FUNC(exit_app), 0);

        scintilla_set_id(sci, 0);
        gtk_signal_connect(GTK_OBJECT(editor), "sci-notify", G_CALLBACK(scintilla_notify), editor);
        gtk_widget_set_usize(editor, 500, 300);

#define SSM(m, w, l) scintilla_send_message(sci, m, w, l)

        SSM(SCI_STYLECLEARALL, 0, 0);
        SSM(SCI_SETLEXER, SCLEX_VARAQ, 0);
        SSM(SCI_STYLESETFORE, SCE_VARAQ_COMMENT, 0x404040);
        SSM(SCI_STYLESETFORE, SCE_VARAQ_STRING, 0x00FF00);
        SSM(SCI_STYLESETFORE, SCE_VARAQ_NUMBER, 0x20A5DA);
        SSM(SCI_STYLESETFORE, SCE_VARAQ_KEYWORD, 0xFF0000);

        SSM(SCI_SETMARGINTYPEN, MARGIN_SCRIPT_FOLD_INDEX, SC_MARGIN_SYMBOL);
        SSM(SCI_SETMARGINMASKN, MARGIN_SCRIPT_FOLD_INDEX, SC_MASK_FOLDERS);
        SSM(SCI_SETMARGINWIDTHN, MARGIN_SCRIPT_FOLD_INDEX, 20);
        SSM(SCI_SETMARGINSENSITIVEN, MARGIN_SCRIPT_FOLD_INDEX, 1);

        SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDER, SC_MARK_BOXPLUS);
        SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDEROPEN, SC_MARK_BOXMINUS);
        SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDEREND, SC_MARK_EMPTY);
        SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDERMIDTAIL, SC_MARK_EMPTY);
        SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDEROPENMID, SC_MARK_EMPTY);
        SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDERSUB, SC_MARK_EMPTY);
        SSM(SCI_MARKERDEFINE, SC_MARKNUM_FOLDERTAIL, SC_MARK_EMPTY);

        SSM(SCI_SETFOLDFLAGS, SC_FOLDFLAG_LINEAFTER_CONTRACTED, 0);

        SSM(SCI_INSERTTEXT, 0, (sptr_t)
         "(*␣99␣balmey␣vo'␣Beer␣Daq␣var'aq␣*)\n"
         "(*␣Sum␣Rune␣Berge␣/␣klingon␣mu'mey␣Sum␣Tati␣*)\n"
         "\n"
         "~␣balmey␣{\n"
         "latlh␣1␣rap\'a\'␣\"\"␣tam\n"
         "{␣woD␣\"s\"␣}␣ghobe\'chugh\n"
         "\"␣bal\"␣tam␣tlheghrar␣tam␣woD\n"
         "\"␣vo\'␣beer\"␣tlheghrar\n"
         "}␣pong\n"
         "\n"
         "~␣chuH␣tlhegh␣{\n"
         "latlh␣latlh␣bottles␣tlheghrar␣\"␣Daq␣reD\"␣tlheghrar␣cha\'\n"
         "latlh␣latlh␣bottles␣tlheghrar␣cha\'\n"
         "\"tlhap␣wa\'␣bIng␣je␣juS␣\'oH␣bIng\"␣cha\'\n"
         "wa\'boqHa\'␣latlh␣latlh␣{␣\"ghobe\'\"␣}␣ghobe'chugh␣latlh␣balmey␣tlheghrar␣\"␣Daq␣reD\"␣tlheghrar␣cha\'\n"
         "\"\"␣cha\'\n"
         "latlh␣{␣chuH␣tlhegh␣}␣HIja\'chugh\n"
         "}␣pong\n"
         "\n"
         "99␣chuH␣tlhegh\n"
        );
        SSM(SCI_COLOURISE, 0, -1);

        gtk_widget_show_all(app);
        gtk_widget_grab_focus(GTK_WIDGET(editor));
        gtk_main();

        return 0;
}
```