My Project

Generated by Doxygen 1.12.0

Chapter 1

Mini Browser

A simple mini browser built using Qt Widgets, Flex, and Bison. This project is designed to fetch and display simple HTML pages while supporting a limited set of HTML tags. It includes features for caching pages locally and allows for easy navigation.

1.1 Features

- Fetches simple HTML pages via HTTP requests.
- Caches pages locally to reduce network requests.
- Supports a restricted set of HTML tags, including:
  - `html`
  - `head`
  - `title`
  - `body`
  - `nav`
  - `ul`, `li`
  - `h1` - `h5`
  - `p`
  - `section`
  - `article`
  - `aside`
  - `footer`
  - `img`
  - `strong`, `em`, `u`, `small`
  - `blockquote`
  - `pre`
  - `code`
  - `ol`

1.2 Getting Started

1.2.1 Prerequisites

Make sure you have the following installed on your system:

- **Qt**: Download Qt
- **Flex**: For generating the lexer.
- **Bison**: For generating the parser.

1.2.2 Installation

1. **Clone the Repository**
   ```
   git clone https://github.com/Nerdy-Byte/mini-browser.git
   cd minibrowser
   ```
2. **Build the Project**

   You can use the provided Makefile or a Qt project file (.pro) to build the application. If you're using qmake, navigate to the project directory and run:
   ```
   cmake . build
   make
   ```

1.3 Usage

1. Enter a URL in the input field and press Enter or click the Go button to fetch the corresponding webpage.
2. The fetched page will be displayed in the main window, rendered according to the supported HTML tags.
3. Navigate through the cached pages using the back and forward buttons.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 DOMNode Class Reference

Public Member Functions

- **DOMNode** (TagType t)
- **DOMNode** (TagType t, const std::string &c)
- **DOMNode** (const DOMNode &)=delete
- DOMNode & **operator=** (const DOMNode &)=delete
- **DOMNode** (DOMNode &&other) noexcept
- DOMNode & **operator=** (DOMNode &&other) noexcept
- void **appendChildren** (const std::vector< DOMNode * > &childList)
- void **setAttribute** (const std::string &name, const std::string &value)
- std::string **getAttribute** (const std::string &name) const
- void **print** (int depth=0) const
- const std::string & **getName** () const
- const std::string & **getTextContent** () const
- const std::vector< DOMNode * > & **getChildren** () const

The documentation for this class was generated from the following file:

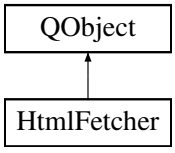
- dom_tree.h

5.2 HtmlFetcher Class Reference

A class that fetches HTML content from a URL.

`#include <dom_creater.h>`

Inheritance diagram for HtmlFetcher:



Signals

- void **fetchFinished** (const QString &content)
- void **errorOccurred** (const QString &error)

Public Member Functions

- **HtmlFetcher** (const QString &url)
- void **fetchAsync** ()
- std::string **fetchSync** ()

5.2.1 Detailed Description

A class that fetches HTML content from a URL.

This class provides methods to asynchronously or synchronously fetch HTML content from a specified URL. The asynchronous method uses signals and slots, while the synchronous method returns the HTML content as a string.

The documentation for this class was generated from the following file:

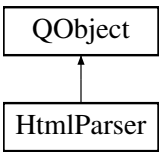
- dom_creater.h

5.3 HtmlParser Class Reference

A class to parse HTML content into a DOM tree.

`#include <dom_creater.h>`

Inheritance diagram for HtmlParser:



Signals

- void **parsingFinished** (DOMNode *root, const std::string &titleText)

Public Member Functions

- **HtmlParser** (const std::string &html_content)
- void **parseAsync** ()
- std::pair< DOMNode *, std::string > **parseSync** ()

5.3.1 Detailed Description

A class to parse HTML content into a DOM tree.

This class provides methods to parse HTML content into a DOM tree structure. It supports both asynchronous and synchronous parsing.

The documentation for this class was generated from the following file:

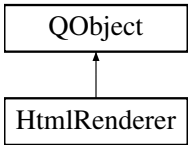
- dom_creater.h

5.4 HtmlRenderer Class Reference

A class to render the DOM tree into a GUI layout.

`#include <dom_creater.h>`

Inheritance diagram for HtmlRenderer:



Public Member Functions

- **HtmlRenderer** (QTabWidget *tabWidget, int tabIndex)
- void **render** (DOMNode *root, const std::string &titleText)

5.4.1 Detailed Description

A class to render the DOM tree into a GUI layout.

This class is responsible for rendering the DOM tree into a specific tab within a QTabWidget. It processes the DOM tree and renders it into a QVBoxLayout, updating the tab with the title text.

The documentation for this class was generated from the following file:

- dom_creater.h

5.5 TaskData Struct Reference

Structure to pass data between threads for fetching and parsing.

Public Attributes

- QString **url**
- std::string **content**
- DOMNode * **root** = nullptr
- std::string **title**
- pthread_mutex_t **mutex** = PTHREAD_MUTEX_INITIALIZER
- pthread_cond_t **condition** = PTHREAD_COND_INITIALIZER
- bool **isComplete** = false

5.5.1 Detailed Description

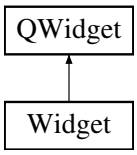
Structure to pass data between threads for fetching and parsing.

The documentation for this struct was generated from the following file:

- main.cpp

5.6 Widget Class Reference

Inheritance diagram for Widget:



Public Member Functions

- **Widget** (QWidget *parent=nullptr)

The documentation for this class was generated from the following files:

- widget.h
- widget.cpp

5.7 yy_buffer_state Struct Reference

Public Attributes

- FILE * **yy_input_file**
- char * **yy_ch_buf**
- char * **yy_buf_pos**
- int **yy_buf_size**
- yy_size_t **yy_n_chars**
- int **yy_is_our_buffer**
- int **yy_is_interactive**
- int **yy_at_bol**
- int yy_bs_lineno
- int yy_bs_column
- int **yy_fill_buffer**
- int **yy_buffer_status**

5.7.1 Member Data Documentation

5.7.1.1 yy_bs_column

```
int yy_buffer_state::yy_bs_column
```

The column count.

5.7.1.2 yy_bs_lineno

```
int yy_buffer_state::yy_bs_lineno
```

The line count.

The documentation for this struct was generated from the following file:

- lexer.cpp

5.8 yy_trans_info Struct Reference

Public Attributes

- flex_int32_t **yy_verify**
- flex_int32_t **yy_nxt**

The documentation for this struct was generated from the following file:

- lexer.cpp

5.9 yyalloc Union Reference

Public Attributes

- yy_state_t **yyss_alloc**
- YYSTYPE **yyvs_alloc**

The documentation for this union was generated from the following file:

- parser.cpp

5.10 YYSTYPE Union Reference

Public Attributes

- DOMNode * **domNode**
- DOMNodeList * **domNodeList**
- char * **text**

The documentation for this union was generated from the following file:

- parser.hpp

Chapter 6

File Documentation

6.1 dom_creater.cpp File Reference

This file defines the DOM creation process by parsing an input string into a DOM tree.

```
#include "dom_creater.h"
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <string>
#include <thread>
#include <mutex>
```

Functions

- int **yyparse** ()

  Function to initiate the parsing process.
- void **yy_scan_string** (const char ∗str)

  Function to scan a string input for parsing.
- void **yy_delete_buffer** (void ∗buffer)

  Function to delete the buffer after scanning.
- DOMNode ∗ dom_creater_string (const std::string &input)

  Parses an input string to create a DOM tree.

Variables

- DOMNode ∗ **root**

  External pointer to the root DOM node after parsing.
- FILE ∗ **yyin** = NULL

  File pointer for the input stream for parsing.
- std::mutex **dom_mutex**

  Mutex to ensure thread safety during DOM creation and parsing.

6.1.1 Detailed Description

This file defines the DOM creation process by parsing an input string into a DOM tree.

6.1.2 Function Documentation

6.1.2.1 dom_creater_string()

```
DOMNode * dom_creater_string (
            const std::string & input)
```

Parses an input string to create a DOM tree.

Creates a DOM tree from a file.

This function uses the `yy_scan_string` to scan the input string, and `yyparse` to parse it into a DOM tree. Parsing is done in a separate thread to allow asynchronous processing. A lock is applied to ensure thread safety during the DOM creation process.

Parameters

input	The string to be parsed into a DOM tree.

Returns

A pointer to the root node of the parsed DOM tree, or nullptr if parsing fails.

< Lock to ensure thread safety

< Scan the input string for parsing

< Initialize the root of the DOM

< Initiates the parsing process

< Set the parsed root to the external root

< Wait for the parsing thread to complete

6.2 dom_creater.h

```
00001 #ifndef DOM_CREATER_H
00002 #define DOM_CREATER_H
00003
00004 #include <QApplication>
00005 #include <QTextEdit>
00006 #include <QVBoxLayout>
00007 #include <QLabel>
00008 #include <QWidget>
00009 #include <QListWidget>
00010 #include <QTabWidget>
00011 #include <string>
00012 #include <cstdlib>
00013 #include <QNetworkAccessManager>
00014 #include <QNetworkReply>
00015 #include <QEventLoop>
00016 #include <QObject>
00017 #include <QString>
00018
00019 #include "dom_tree.h"
00020
00031 DOMNode* dom_creater_main(char*);
00032
00043 DOMNode* dom_creater_string(const std::string& input);
00044
00056 void renderDOMNode(DOMNode* node, QVBoxLayout* layout, QWidget* mainWindow = nullptr);
00057
00069 void renderDOMTree(DOMNode* root, QVBoxLayout* layout, QTabWidget* tabWidget);
00070
00081 std::string findTitle(DOMNode* node);
00082
00083 // HtmlFetcher class to fetch HTML content
00091 class HtmlFetcher : public QObject {
00092     Q_OBJECT
00093
00094 public:
00095     HtmlFetcher(const QString& url)
00096         : m_url(url) {}
00097
00098     void fetchAsync() {
00099         QNetworkAccessManager* manager = new QNetworkAccessManager(this);
00100         QObject::connect(manager, &QNetworkAccessManager::finished, this,
      &HtmlFetcher::onFetchFinished);
00101         manager->get(QNetworkRequest(QUrl(m_url)));
00102     }
00103
00104     std::string fetchSync() {
00105         QNetworkAccessManager manager;
00106         QEventLoop loop;
00107
00108         // Connect manager's finished signal to the event loop's quit slot
00109         QObject::connect(&manager, &QNetworkAccessManager::finished, &loop, &QEventLoop::quit);
00110
00111         // Make the request
00112         QNetworkReply* reply = manager.get(QNetworkRequest(QUrl(m_url)));
00113         loop.exec(); // Wait for the request to finish
00114
00115         // Handle the reply
00116         if (reply->error() != QNetworkReply::NoError) {
00117             QString error = reply->errorString();
00118             reply->deleteLater();
00119             throw std::runtime_error(error.toStdString());
00120         }
00121
00122         QString htmlContent = reply->readAll();
00123         reply->deleteLater();
00124         return htmlContent.toStdString(); // Convert to std::string
00125     }
00126
00127
00128 signals:
00129     void fetchFinished(const QString& content);
00130     void errorOccurred(const QString& error);
00131
00132 private slots:
00133     void onFetchFinished(QNetworkReply* reply) {
00134         if (reply->error() != QNetworkReply::NoError) {
00135             emit errorOccurred(reply->errorString());
00136             reply->deleteLater();
00137             return;
00138         }
00139
00140         QString htmlContent = reply->readAll();
00141         reply->deleteLater();
00142         emit fetchFinished(htmlContent);
00143     }
```

```
00144
00145 private:
00146     QString m_url;
00147 };
00148
00149 // HtmlParser class to parse HTML content into a DOM tree
00157 class HtmlParser : public QObject {
00158     Q_OBJECT
00159
00160 public:
00161     HtmlParser(const std::string& html_content)
00162         : m_html_content(QString::fromStdString(html_content)) {} // Convert std::string to QString
00163
00164     void parseAsync() {
00165         DOMNode* root = dom_creater_string(m_html_content.toStdString());
00166         if (!root) {
00167             emit parsingFinished(nullptr, "");
00168             return;
00169         }
00170
00171         std::string titleText = findTitle(root);
00172         emit parsingFinished(root, titleText);
00173     }
00174
00175     std::pair<DOMNode*, std::string> parseSync() {
00176         DOMNode* root = dom_creater_string(m_html_content.toStdString());
00177         if (!root) {
00178             throw std::runtime_error("Failed to parse HTML content.");
00179         }
00180
00181         std::string titleText = findTitle(root);
00182         return {root, titleText};
00183     }
00184
00185 signals:
00186     void parsingFinished(DOMNode* root, const std::string& titleText);
00187
00188 private:
00189     QString m_html_content;
00190 };
00191
00192
00193 // HtmlRenderer class to render the DOM tree into the GUI
00202 class HtmlRenderer : public QObject {
00203     Q_OBJECT
00204
00205 public:
00206     HtmlRenderer(QTabWidget* tabWidget, int tabIndex)
00207         : m_tabWidget(tabWidget), m_tabIndex(tabIndex) {}
00208
00209     void render(DOMNode* root, const std::string& titleText) {
00210         QWidget* targetTab = m_tabWidget->widget(m_tabIndex);
00211         if (targetTab) {
00212             QVBoxLayout* tabLayout = qobject_cast<QVBoxLayout*>(targetTab->layout());
00213             if (tabLayout) {
00214                 // Clear the previous content
00215                 QLayoutItem* item;
00216                 while ((item = tabLayout->takeAt(0)) != nullptr) {
00217                     delete item->widget();
00218                     delete item;
00219                 }
00220
00221                 // Render the DOM tree
00222                 renderDOMTree(root, tabLayout, m_tabWidget);
00223
00224                 // Set the tab name
00225                 QString tabName = QString::fromStdString(titleText.empty() ? "Untitled" : titleText);
00226                 m_tabWidget->setTabText(m_tabIndex, tabName);
00227             }
00228         }
00229     }
00230
00231 private:
00232     QTabWidget* m_tabWidget;
00233     int m_tabIndex;
00234 };
00235
00236 #endif
```

6.3 dom_tree.h

```
00001 #ifndef DOM_TREE_H
00002 #define DOM_TREE_H
```

```
00003
00004 #include <iostream>
00005 #include <vector>
00006 #include <map>
00007 #include <string>
00008
00009 // Enumeration for HTML tags
00010 enum TagType {
00011     ROOT,
00012     HTML,
00013     HEAD,
00014     TITLE,
00015     BODY,
00016     DIV,
00017     P,
00018     H1,
00019     H2,
00020     H3,
00021     H4,
00022     H5,
00023     SECTION,
00024     ARTICLE,
00025     ASIDE,
00026     IMG,
00027     A,
00028     STRONG,
00029     EM,
00030     U,
00031     SMALL,
00032     BLOCK_QUOTE,
00033     PRE,
00034     CODE,
00035     NAV,
00036     OL,
00037     UL,
00038     LI,
00039     HEADER,
00040     FOOTER,
00041     SRC,
00042     ALT,
00043     TXT,
00044     ERROR
00045 };
00046
00047
00048 inline std::string tagTypeToString(TagType tagType) {
00049     switch (tagType) {
00050     case TagType::HTML: return "html";
00051     case TagType::HEAD: return "head";
00052     case TagType::TITLE: return "title";
00053     case TagType::BODY: return "body";
00054     case TagType::NAV: return "nav";
00055     case TagType::DIV: return "div";
00056     case TagType::P: return "p";
00057     case TagType::H1: return "h1";
00058     case TagType::ROOT: return "root";
00059     case TagType::H2: return "h2";
00060     case TagType::H3: return "h3";
00061     case TagType::H4: return "h4";
00062     case TagType::H5: return "h5";
00063     case TagType::SECTION: return "section";
00064     case TagType::EM: return "em";
00065     case TagType::ARTICLE: return "article";
00066     case TagType::ASIDE: return "aside";
00067     case TagType::IMG: return "img";
00068     case TagType::A: return "a";
00069     case TagType::STRONG: return "strong";
00070     case TagType::BLOCK_QUOTE: return "block_quote";
00071     case TagType::U: return "u";
00072     case TagType::SMALL: return "small";
00073     case TagType::PRE: return "pre";
00074     case TagType::OL: return "ol";
00075     case TagType::UL: return "ul";
00076     case TagType::LI: return "li";
00077     case TagType::HEADER: return "header";
00078     case TagType::FOOTER: return "footer";
00079     case TagType::CODE: return "code";
00080     case TagType::SRC: return "src";
00081     case TagType::ALT: return "alt";
00082     case TagType::TXT: return "text";
00083
00084     default: return "unknown";
00085     }
00086 }
00087
00088 inline TagType getTagType(const std::string& tagName) {
00089     if (tagName == "html") return TagType::HTML;
```

```
00090     if (tagName == "head") return TagType::HEAD;
00091     if (tagName == "title") return TagType::TITLE;
00092     if (tagName == "body") return TagType::BODY;
00093     if (tagName == "div") return TagType::DIV;
00094     if (tagName == "p") return TagType::P;
00095     if (tagName == "h1") return TagType::H1;
00096     if (tagName == "h2") return TagType::H2;
00097     if (tagName == "h3") return TagType::H3;
00098     if (tagName == "h4") return TagType::H4;
00099     if (tagName == "h5") return TagType::H5;
00100     if (tagName == "section") return TagType::SECTION;
00101     if (tagName == "article") return TagType::ARTICLE;
00102     if (tagName == "aside") return TagType::ASIDE;
00103     if (tagName == "img") return TagType::IMG;
00104     if (tagName == "a") return TagType::A;
00105     if (tagName == "strong") return TagType::STRONG;
00106     if (tagName == "em") return TagType::EM;
00107     if (tagName == "u") return TagType::U;
00108     if (tagName == "small") return TagType::SMALL;
00109     if (tagName == "block_quote") return TagType::BLOCK_QUOTE;
00110     if (tagName == "pre") return TagType::PRE;
00111     if (tagName == "code") return TagType::CODE;
00112     if (tagName == "nav") return TagType::NAV;
00113     if (tagName == "ol") return TagType::OL;
00114     if (tagName == "ul") return TagType::UL;
00115     if (tagName == "li") return TagType::LI;
00116     if (tagName == "header") return TagType::HEADER;
00117     if (tagName == "footer") return TagType::FOOTER;
00118     if (tagName == "root") return TagType::ROOT;
00119     if (tagName == "code") return TagType::CODE;
00120     if (tagName == "src") return TagType::SRC;
00121     if (tagName == "alt") return TagType::ALT;
00122     if (tagName == "text") return TagType::TXT;
00123     return TagType::ERROR;
00124 }
00125
00126
00127 class DOMNode {
00128 public:
00129     DOMNode(TagType t) : tag(tagTypeToString(t)), content("") {}
00130     DOMNode(TagType t, const std::string& c) : tag(tagTypeToString(t)), content(c) {}
00131
00132     ~DOMNode() {
00133         for (auto* child : children) {
00134             delete child;
00135         }
00136     }
00137
00138     DOMNode(const DOMNode&) = delete;
00139     DOMNode& operator=(const DOMNode&) = delete;
00140
00141     DOMNode(DOMNode&& other) noexcept : tag(std::move(other.tag)), content(std::move(other.content)),
    children(std::move(other.children)), attributes(std::move(other.attributes)) {
00142         other.children.clear();
00143     }
00144
00145     DOMNode& operator=(DOMNode&& other) noexcept {
00146         if (this != &other) {
00147             for (auto* child : children) {
00148                 delete child;
00149             }
00150             tag = std::move(other.tag);
00151             content = std::move(other.content);
00152             children = std::move(other.children);
00153             attributes = std::move(other.attributes);
00154             other.children.clear();
00155         }
00156         return *this;
00157     }
00158
00159     // Method to append children nodes
00160     void appendChildren(const std::vector<DOMNode*>& childList) {
00161         children.insert(children.end(), childList.begin(), childList.end());
00162     }
00163
00164     // Method to set an attribute
00165     void setAttribute(const std::string& name, const std::string& value) {
00166         attributes[name] = value;
00167     }
00168
00169     // Method to get an attribute
00170     std::string getAttribute(const std::string& name) const {
00171         auto it = attributes.find(name);
00172         if (it != attributes.end()) {
00173             return it->second;
00174         }
00175         return "";
```

```
00176     }
00177
00178     // Method to print the DOM tree (recursive)
00179     void print(int depth = 0) const {
00180         std::cout « std::string(depth * 2, ' ');
00181         std::cout « tag;
00182         if (!content.empty()) {
00183             std::cout « ": " « content;
00184         }
00185
00186         // Print attributes
00187         for (const auto& [key, value] : attributes) {
00188             std::cout « " (" « key « "=\"" « value « "\")";
00189         }
00190         std::cout « std::endl;
00191
00192         for (const auto* child : children) {
00193             child->print(depth + 1);
00194         }
00195     }
00196
00197     // Getter for the tag name
00198     const std::string& getName() const {
00199         return tag;
00200     }
00201
00202     // Getter for the text content
00203     const std::string& getTextContent() const {
00204         return content;
00205     }
00206
00207     // Getter for child nodes
00208     const std::vector<DOMNode*>& getChildren() const {
00209         return children;
00210     }
00211
00212 private:
00213     std::string tag;  // Tag name
00214     std::string content;  // Text content of the node
00215     std::vector<DOMNode*> children;  // Children of this node
00216     std::map<std::string, std::string> attributes;  // Attributes
00217 };
00218
00219 typedef std::vector<DOMNode*> DOMNodeList;
00220
00221
00222 #endif
```

6.4 main.cpp File Reference

Entry point for the DOM Browser application using pthread for concurrency.

```
#include "dom_creater.h"
#include <QApplication>
#include <QPushButton>
#include <QVBoxLayout>
#include <QTabWidget>
#include <QLineEdit>
#include <QMap>
#include <pthread.h>
#include <queue>
#include <utility>
```

Classes

- struct TaskData

  Structure to pass data between threads for fetching and parsing.

Functions

- void * fetchHtmlThread (void *arg)

  Thread function for fetching HTML content.
- void * parseHtmlThread (void *arg)

  Thread function for parsing HTML content.
- void renderContent (const QString &url, QTabWidget *tabWidget, int tabIndex)

  Renders HTML content in a browser tab.
- void createNewTab (QTabWidget *tabWidget)

  Creates a new browser tab with input fields for URL entry and an HTML fetch button.
- int main (int argc, char *argv[])

  The main function of the DOM Browser application.

Variables

- QMap< QString, std::pair< DOMNode *, std::string > > pageCache

  Cache to store previously fetched and parsed pages.

6.4.1 Detailed Description

Entry point for the DOM Browser application using pthread for concurrency.

6.4.2 Function Documentation

6.4.2.1 createNewTab()

```
void createNewTab (
            QTabWidget * tabWidget)
```

Creates a new browser tab with input fields for URL entry and an HTML fetch button.

Adds a new tab to the `QTabWidget` and connects the fetch button to the `renderContent` function.

Parameters

tabWidget	The `QTabWidget` where the new tab will be added.

6.4.2.2 fetchHtmlThread()

```
void * fetchHtmlThread (
            void * arg)
```

Thread function for fetching HTML content.

This function fetches HTML content from the specified URL.

Parameters

arg	Pointer to `TaskData` containing the URL and results.

Returns

nullptr

6.4.2.3 main()

```
int main (
            int argc,
            char * argv[])
```

The main function of the DOM Browser application.

Initializes the Qt application, sets up the main window with a tab widget, and allows users to open, fetch, and render HTML pages.

Parameters

argc	The number of command-line arguments.
argv	The command-line arguments.

Returns

The exit code of the application.

6.4.2.4 parseHtmlThread()

```
void * parseHtmlThread (
            void * arg)
```

Thread function for parsing HTML content.

This function parses HTML content into a DOM tree.

Parameters

arg	Pointer to `TaskData` containing HTML content and results.

Returns

nullptr

6.4.2.5 renderContent()

```
void renderContent (
            const QString & url,
            QTabWidget * tabWidget,
            int tabIndex)
```

Renders HTML content in a browser tab.

Fetches and parses the content using pthread and renders it in the specified tab.

Parameters

url	The URL of the page to render.
tabWidget	The `QTabWidget` containing browser tabs.
tabIndex	The index of the tab to render the content in.

6.4.3 Variable Documentation

6.4.3.1 pageCache

```
QMap<QString, std::pair<DOMNode*, std::string> > pageCache
```

Cache to store previously fetched and parsed pages.

The cache maps URLs to a pair of parsed DOM nodes and their titles.

6.5 parser.hpp

```
00001 /* A Bison parser, made by GNU Bison 3.8.2.  */
00002
00003 /* Bison interface for Yacc-like parsers in C
00004
00005    Copyright (C) 1984, 1989-1990, 2000-2015, 2018-2021 Free Software Foundation,
00006    Inc.
00007
00008    This program is free software: you can redistribute it and/or modify
00009    it under the terms of the GNU General Public License as published by
00010    the Free Software Foundation, either version 3 of the License, or
00011    (at your option) any later version.
00012
00013    This program is distributed in the hope that it will be useful,
00014    but WITHOUT ANY WARRANTY; without even the implied warranty of
00015    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00016    GNU General Public License for more details.
00017
00018    You should have received a copy of the GNU General Public License
00019    along with this program.  If not, see <https://www.gnu.org/licenses/>.  */
00020
00021 /* As a special exception, you may create a larger work that contains
00022    part or all of the Bison parser skeleton and distribute that work
00023    under terms of your choice, so long as that work isn't itself a
00024    parser generator using the skeleton or a modified version thereof
00025    as a parser skeleton.  Alternatively, if you modify or redistribute
00026    the parser skeleton itself, you may (at your option) remove this
00027    special exception, which will cause the skeleton and the resulting
00028    Bison output files to be licensed under the GNU General Public
00029    License without this special exception.
00030
00031    This special exception was added by the Free Software Foundation in
00032    version 2.2 of Bison.  */
00033
00034 /* DO NOT RELY ON FEATURES THAT ARE NOT DOCUMENTED in the manual,
00035    especially those whose name start with YY_ or yy_.  They are
00036    private implementation details that can be changed or removed.  */
00037
00038 #ifndef YY_YY_USERS_DIVYANSHUDWIVEDI2018GMAIL_COM_DESKTOP_SSL_UNTITLED_PARSER_HPP_INCLUDED
00039 # define YY_YY_USERS_DIVYANSHUDWIVEDI2018GMAIL_COM_DESKTOP_SSL_UNTITLED_PARSER_HPP_INCLUDED
00040 /* Debug traces.  */
00041 #ifndef YYDEBUG
00042 # define YYDEBUG 0
00043 #endif
00044 #if YYDEBUG
00045 extern int yydebug;
00046 #endif
00047
00048 /* Token kinds.  */
00049 #ifndef YYTOKENTYPE
00050 # define YYTOKENTYPE
00051   enum yytokentype
00052   {
```

```
00053     YYEMPTY = -2,
00054     YYEOF = 0,                     /* "end of file"  */
00055     YYerror = 256,                 /* error  */
00056     YYUNDEF = 257,                 /* "invalid token"  */
00057     TEXT = 258,                    /* TEXT  */
00058     DOCTYPE = 259,                 /* DOCTYPE  */
00059     HTML_OPEN = 260,               /* HTML_OPEN  */
00060     HTML_CLOSE = 261,              /* HTML_CLOSE  */
00061     HEAD_OPEN = 262,               /* HEAD_OPEN  */
00062     HEAD_CLOSE = 263,              /* HEAD_CLOSE  */
00063     TITLE_OPEN = 264,              /* TITLE_OPEN  */
00064     TITLE_CLOSE = 265,             /* TITLE_CLOSE  */
00065     BODY_OPEN = 266,               /* BODY_OPEN  */
00066     BODY_CLOSE = 267,              /* BODY_CLOSE  */
00067     DIV_OPEN = 268,                /* DIV_OPEN  */
00068     DIV_CLOSE = 269,               /* DIV_CLOSE  */
00069     P_OPEN = 270,                  /* P_OPEN  */
00070     P_CLOSE = 271,                 /* P_CLOSE  */
00071     H1_OPEN = 272,                 /* H1_OPEN  */
00072     H1_CLOSE = 273,                /* H1_CLOSE  */
00073     H2_OPEN = 274,                 /* H2_OPEN  */
00074     H2_CLOSE = 275,                /* H2_CLOSE  */
00075     H3_OPEN = 276,                 /* H3_OPEN  */
00076     H3_CLOSE = 277,                /* H3_CLOSE  */
00077     H4_OPEN = 278,                 /* H4_OPEN  */
00078     H4_CLOSE = 279,                /* H4_CLOSE  */
00079     H5_OPEN = 280,                 /* H5_OPEN  */
00080     H5_CLOSE = 281,                /* H5_CLOSE  */
00081     NAV_OPEN = 282,                /* NAV_OPEN  */
00082     NAV_CLOSE = 283,               /* NAV_CLOSE  */
00083     UL_OPEN = 284,                 /* UL_OPEN  */
00084     UL_CLOSE = 285,                /* UL_CLOSE  */
00085     LI_OPEN = 286,                 /* LI_OPEN  */
00086     LI_CLOSE = 287,                /* LI_CLOSE  */
00087     HEADER_OPEN = 288,             /* HEADER_OPEN  */
00088     HEADER_CLOSE = 289,            /* HEADER_CLOSE  */
00089     FOOTER_OPEN = 290,             /* FOOTER_OPEN  */
00090     FOOTER_CLOSE = 291,            /* FOOTER_CLOSE  */
00091     SECTION_OPEN = 292,            /* SECTION_OPEN  */
00092     SECTION_CLOSE = 293,           /* SECTION_CLOSE  */
00093     ARTICLE_OPEN = 294,            /* ARTICLE_OPEN  */
00094     ARTICLE_CLOSE = 295,           /* ARTICLE_CLOSE  */
00095     ASIDE_OPEN = 296,              /* ASIDE_OPEN  */
00096     ASIDE_CLOSE = 297,             /* ASIDE_CLOSE  */
00097     OL_OPEN = 298,                 /* OL_OPEN  */
00098     OL_CLOSE = 299,                /* OL_CLOSE  */
00099     A_OPEN = 300,                  /* A_OPEN  */
00100     A_CLOSE = 301,                 /* A_CLOSE  */
00101     STRONG_OPEN = 302,             /* STRONG_OPEN  */
00102     STRONG_CLOSE = 303,            /* STRONG_CLOSE  */
00103     EM_OPEN = 304,                 /* EM_OPEN  */
00104     EM_CLOSE = 305,                /* EM_CLOSE  */
00105     U_OPEN = 306,                  /* U_OPEN  */
00106     U_CLOSE = 307,                 /* U_CLOSE  */
00107     SMALL_OPEN = 308,              /* SMALL_OPEN  */
00108     SMALL_CLOSE = 309,             /* SMALL_CLOSE  */
00109     PRE_OPEN = 310,                /* PRE_OPEN  */
00110     PRE_CLOSE = 311,               /* PRE_CLOSE  */
00111     BLOCKQUOTE_OPEN = 312,         /* BLOCKQUOTE_OPEN  */
00112     BLOCKQUOTE_CLOSE = 313,        /* BLOCKQUOTE_CLOSE  */
00113     CODE_OPEN = 314,               /* CODE_OPEN  */
00114     CODE_CLOSE = 315,              /* CODE_CLOSE  */
00115     IMG_TAG = 316                  /* IMG_TAG  */
00116   };
00117   typedef enum yytokentype yytoken_kind_t;
00118 #endif
00119
00120 /* Value type.  */
00121 #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
00122 union YYSTYPE
00123 {
00124 #line 18 "/Users/divyanshudwivedi2018gmail.com/Desktop/SSL/untitled/parser.y"
00125
00126     DOMNode* domNode;
00127     DOMNodeList* domNodeList;
00128     char* text;
00129
00130 #line 131 "/Users/divyanshudwivedi2018gmail.com/Desktop/SSL/untitled/parser.hpp"
00131
00132 };
00133 typedef union YYSTYPE YYSTYPE;
00134 # define YYSTYPE_IS_TRIVIAL 1
00135 # define YYSTYPE_IS_DECLARED 1
00136 #endif
00137
00138
00139 extern YYSTYPE yylval;
```

```
00140
00141
00142 int yyparse (void);
00143
00144
00145 #endif /* !YY_YY_USERS_DIVYANSHUDWIVEDI2018GMAIL_COM_DESKTOP_SSL_UNTITLED_PARSER_HPP_INCLUDED  */
```

6.6 widget.h

```
00001 #ifndef WIDGET_H
00002 #define WIDGET_H
00003
00004 #include <QWidget>
00005
00006 QT_BEGIN_NAMESPACE
00007 namespace Ui {
00008 class Widget;
00009 }
00010 QT_END_NAMESPACE
00011
00012 class Widget : public QWidget
00013 {
00014     Q_OBJECT
00015
00016 public:
00017     Widget(QWidget *parent = nullptr);
00018     ~Widget();
00019
00020 private:
00021     Ui::Widget *ui;
00022 };
00023 #endif // WIDGET_H
```

Index