Contents

introduction:

This XML editor serves as a comprehensive tool for users to seamlessly load, edit, and save XML files while offering a plethora of additional features. It not only detects errors in XML files but also provides users with insightful information, such as the number and locations of errors. Furthermore, the editor facilitates error correction, ensuring the resulting XML file maintains consistency.

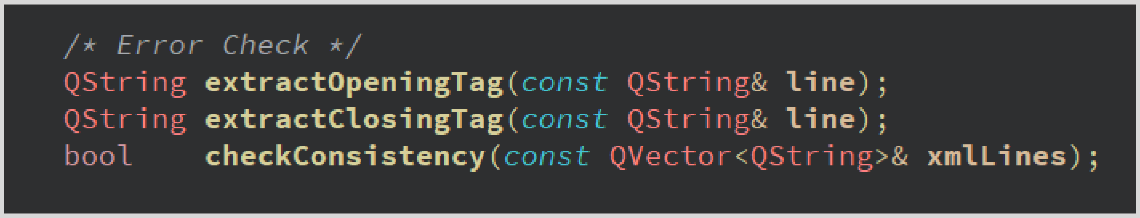
Beyond error handling, users can leverage various functionalities, including formatting and minifying XML files. The editor extends its capabilities to convert XML to JSON format, compress files to reduce size, and decompress them when needed. Additionally, users can harness the tool to generate graph visualizations for social networks represented by XML files. Furthermore, the editor offers essential analytical insights derived from the XML data, enhancing its utility for users engaged in diverse XML-related tasks.

Features:

- Browsing XML Files
- Checking the XML consistency
- Detecting & Correcting Error in consistency in XML File
- Format XML Files
- Minifying (Reducing file size by removing extra spaces).
- Compressing XML File
- Decompressing XML File
- Converting XML File to Json
- Undo & Redo
- Graph representation
- Social Network Analysis
- Save the XML file

XML Consistency:

Checking the XML consistency



```
/* Error Check */
QString extractOpeningTag(const QString& line);
QString extractClosingTag(const QString& line);
bool    checkConsistency(const QVector<QString>& xmlLines);
```

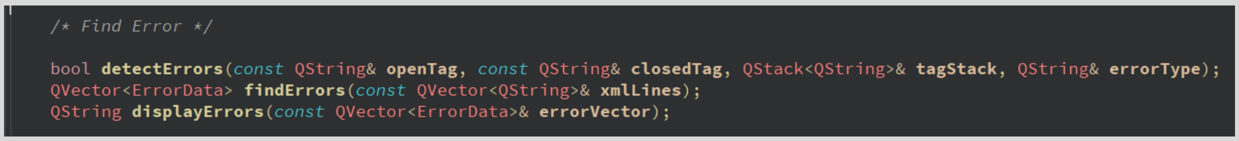
1. **extractOpeningTag() Function:**
   - Takes a line of text and extracts the opening XML tag.
   - Finds the starting and ending indices of the opening tag in the line.
   - Returns the extracted opening tag, or "INVALID" if it's not found or if it's a closing tag.

2. **extractClosingTag() Function:**
   - Takes a line of text and extracts the closing XML tag.
   - Finds the indices of '/' and '>' to identify the closing tag in the line.
   - Returns the extracted closing tag, or "INVALID" if it's not found.

3. **checkConsistency() Function:**
   - Takes a vector of strings (**xmlLines**) containing XML content.
   - Uses a stack to track opening tags encountered while iterating through the lines.
   - Calls **extractOpeningTag()** and **extractClosingTag()** to identify and handle opening and closing tags in the XML lines.
   - Checks consistency by ensuring opening and closing tags match appropriately:
     - If both opening and closing tags are present, checks if they match. If not, returns **false**.
     - Handles cases where only opening or only closing tags are encountered.
     - If the tags are consistent, returns **true**. If not, returns **false**.
   - Ultimately, checks if the stack is empty at the end of the iteration, ensuring all opening tags have their corresponding closing tags.

This code helps ensure the structural consistency of XML content by examining opening and closing tags, confirming their proper alignment and hierarchy, and returns **true** if the XML structure is consistent or **false** if inconsistencies are found.

Detecting Errors in consistency in XML File:



```
/* Find Error */

bool detectErrors(const QString& openTag, const QString& closedTag, QStack<QString>& tagStack, QString& errorType);
QVector<ErrorData> findErrors(const QVector<QString>& xmlLines);
QString displayErrors(const QVector<ErrorData>& errorVector);
```

detectErrors():

- **Purpose:** Checks for inconsistencies in XML tag pairs (openTag and closedTag) and manages a tagStack to track encountered tags.

- **Implementation:**
    - Compares opening and closing tags, reporting a mismatch if they are different.
    - Manages the tagStack to ensure proper matching of tags and handles missing opening or closing tags.
    - Returns false upon detecting inconsistencies and populates errorType with descriptive error messages.

findErrors():

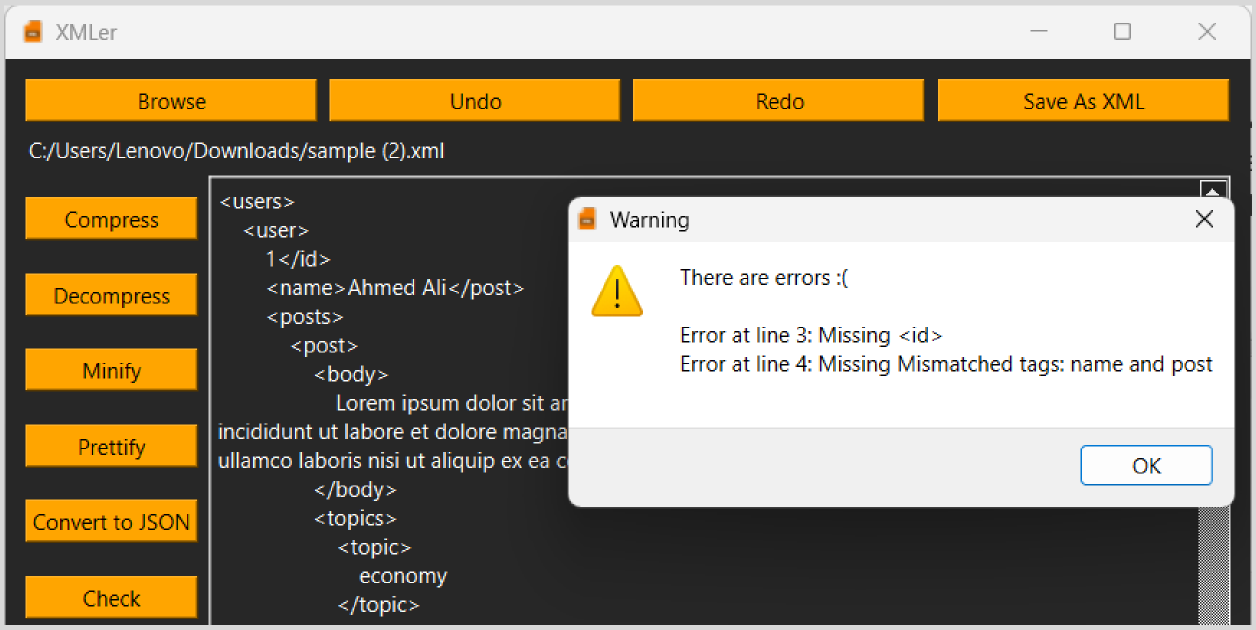
- **Purpose:** Scans a vector of XML lines (xmlLines) and utilizes detectErrors() to identify errors in tag pairs.

- **Implementation:**
    - Parses each line to extract opening and closing tags using extractOpeningTag() and extractClosingTag().
    - Utilizes detectErrors() to identify inconsistencies and records error details in the errorVector.
    - Handles incomplete XML content by identifying missing closing tags and adds corresponding error entries.

displayErrors():

- **Purpose:** Generates an error message based on the collected error data (errorVector).

- **Implementation:**
    - Formats the error messages with details about error locations and types.
    - Distinguishes between different error types (mismatched tags or missing tags) and constructs a comprehensive error message summarizing all identified issues. Together, these functions provide a robust mechanism to scan XML content, detect inconsistencies in tag pairs, and produce a detailed error report, aiding in debugging and correcting issues within XML files.



C:/Users/Lenovo/Downloads/sample (2).xml

```
<users>
  <user>
    1</id>
    <name>Ahmed Ali</post>
    <posts>
      <post>
        <body>
          Lorem ipsum dolor sit ar
incididunt ut labore et dolore magna
ullamco laboris nisi ut aliquip ex ea c
        </body>
        <topics>
          <topic>
            economy
          </topic>
```

There are errors :(

Error at line 3: Missing <id>
Error at line 4: Missing Mismatched tags: name and post

Correcting Error in consistency in XML File:

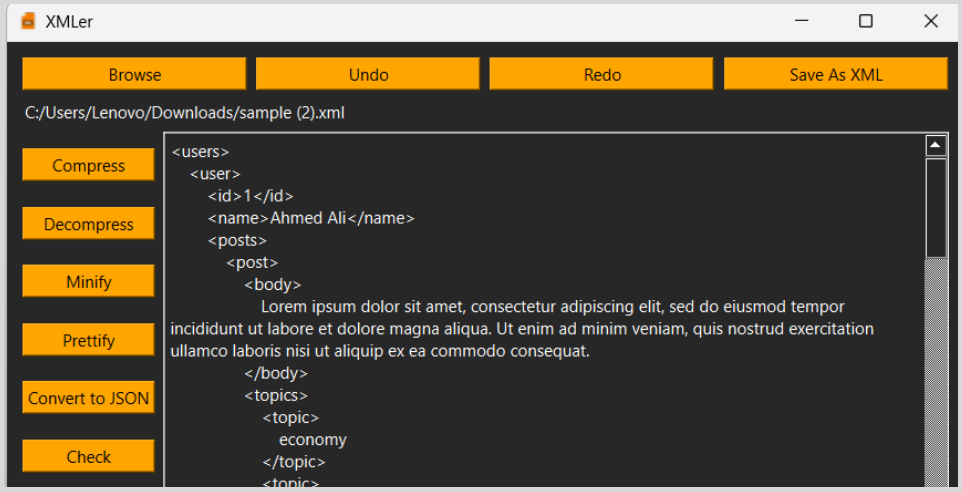
```
/*  correct */
QVector<QString> error_corrector(const QVector<QString>& xml_vector, const QVector<ErrorData>& error_vector);
```

Implementation Overview:

- **Purpose:**
  - Receives a list of XML lines (xmlLines) and a vector of error data (errorVector) describing detected errors.
  - Creates a new list of corrected lines (correctedLines) by addressing identified errors.

- **Error Correction Logic:**
  - Iterates through each line in the original XML content.
  - Matches error locations indicated in the errorVector with their corresponding lines in xmlLines.
  - Modifies lines based on the type of detected error:
    - For mismatched tags errors (length > 25), replaces the closing tag with the opening tag to rectify the issue.
    - For other types of errors, inserts the error type and the original line with formatting.

- **Handling Remaining Errors:**
  - If there are any unaddressed errors at the end of the XML content, adds them to the corrected lines to ensure all errors are captured in the output.



C:/Users/Lenovo/Downloads/sample (2).xml

```
<users>
  <user>
    <id>1</id>
    <name>Ahmed Ali</name>
    <posts>
      <post>
        <body>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
        </body>
        <topics>
          <topic>
            economy
          </topic>
          <topic>
```

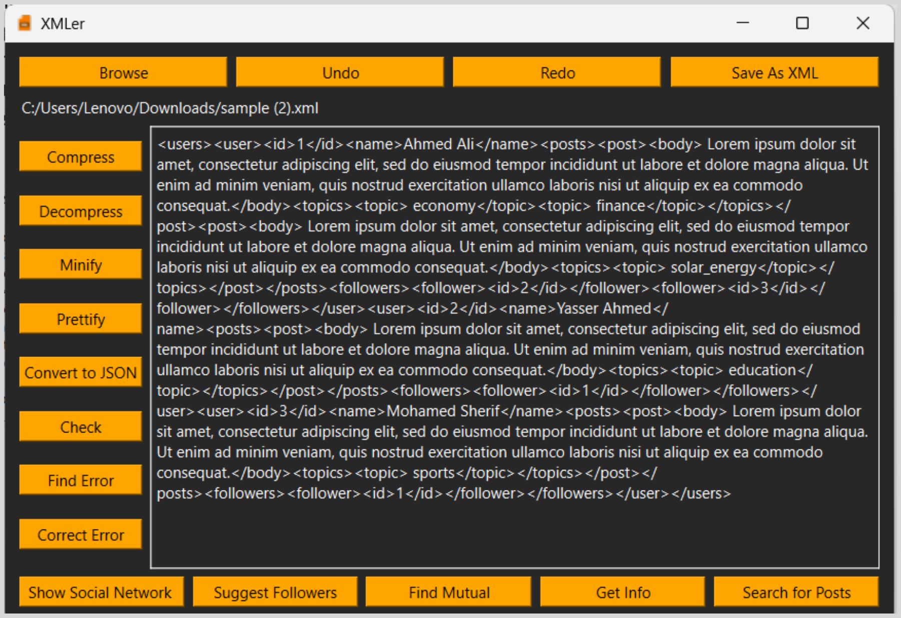
	Time complexity	**Space complexity**
Check consistency	O(N)	O(N)
Detect errors	O(N)	O(N)
Correct errors	O(N)	O(N)

Minifying:

```
        void minifyXml();
```

removing unnecessary whitespace while ensuring the integrity of the XML structure. It parses the text character by character, skipping and condensing whitespace within tags and consolidating consecutive whitespace characters. This process results in a more compact representation of the XML data while maintaining the document's structural integrity. If no file path is provided, the function prompts the user to select an XML file.

this is an example:

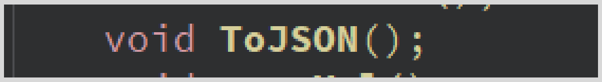


C:/Users/Lenovo/Downloads/sample (2).xml

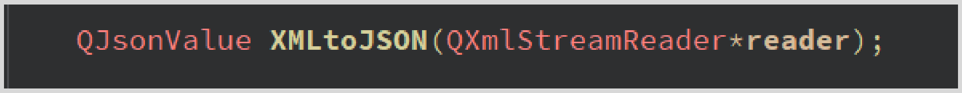
```
<users><user><id>1</id><name>Ahmed Ali</name><posts><post><body> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</body><topics><topic> economy</topic><topic> finance</topic></topics></post><post><body> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</body><topics><topic> solar_energy</topic></topics></post></posts><followers><follower><id>2</id></follower><follower><id>3</id></follower></followers></user><user><id>2</id><name>Yasser Ahmed</name><posts><post><body> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</body><topics><topic> education</topic></topics></post></posts><followers><follower><id>1</id></follower></followers></user><user><id>3</id><name>Mohamed Sherif</name><posts><post><body> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</body><topics><topic> sports</topic></topics></post></posts><followers><follower><id>1</id></follower></followers></user></users>
```

	Time complexity	**Space complexity**
Minify	O(N)	O(N)

convert to Json:

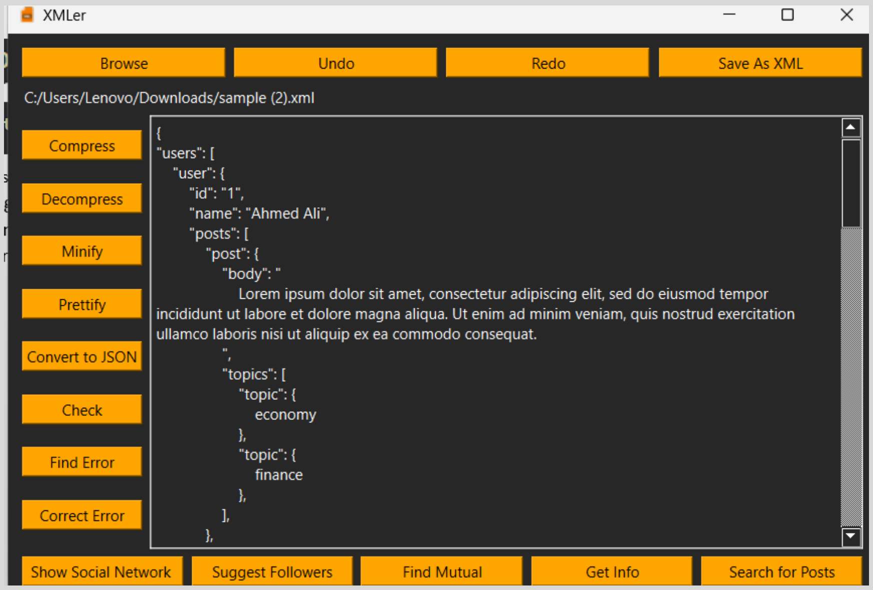


```
void ToJSON();
```



```
QJsonValue XMLtoJSON(QXmlStreamReader*reader);
```

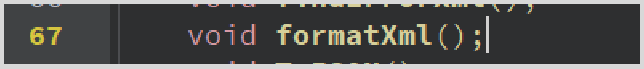
Conversion to Json process is done using these two functions:
These functions work together to read XML content, interpret its structure, and convert it into a JSON representation using Qt's classes for XML and JSON handling. The **XMLtoJSON()** function is crucial here, as it performs the actual parsing and conversion of the XML content into its JSON equivalent.



```
{
"users": [
   "user": {
      "id": "1",
      "name": "Ahmed Ali",
      "posts": [
         "post": {
            "body": "
               Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.
            ",
            "topics": [
               "topic": {
                  economy
               },
               "topic": {
                  finance
               },
            ],
         },
```

	Time complexity	**Space complexity**
Json	O(N)	O(1)

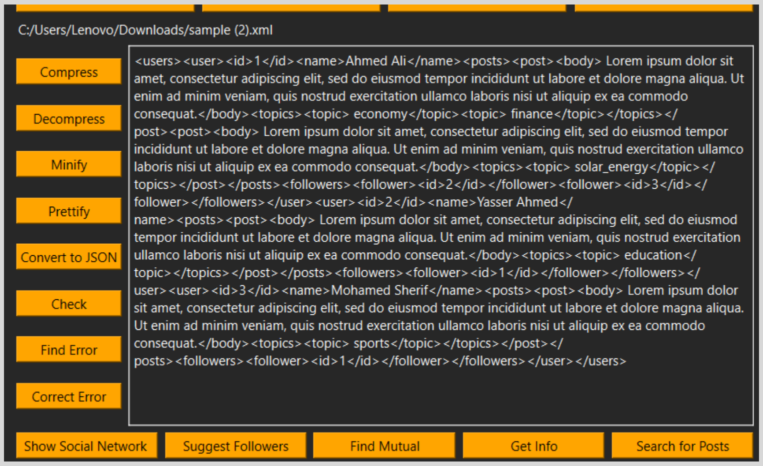
Formatting:



```
void formatXml();
```

This function aims to enhance the readability of XML content by adjusting indentation and removing unnecessary leading spaces. It reads XML content from a specified file path, processes the data by adjusting indentation based on opening and closing tags, and generates a formatted output. This formatted XML content, with consistent and appropriate indentation, is then set into the designated text field (**resultTextEdit**). Additionally, if no file path is provided, the function prompts the user to select an XML file. Ultimately, this function serves to improve the readability and maintainability of XML documents by applying consistent formatting to the content.

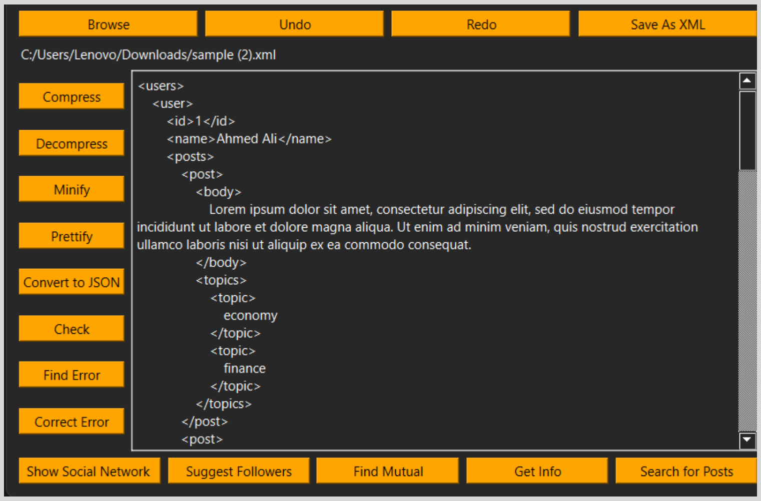
input



C:/Users/Lenovo/Downloads/sample (2).xml

```
<users><user><id>1</id><name>Ahmed Ali</name><posts><post><body> Lorem ipsum dolor sit
amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat.</body><topics><topic> economy</topic><topic> finance</topic></topics></
post><post><body> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat.</body><topics><topic> solar_energy</topic></
topics></post></posts><followers><follower><id>2</id></follower><follower><id>3</id></
follower></followers></user><user><id>2</id><name>Yasser Ahmed</
name><posts><post><body> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.</body><topics><topic> education</
topic></topics></post></posts><followers><follower><id>1</id></follower></followers></
user><user><id>3</id><name>Mohamed Sherif</name><posts><post><body> Lorem ipsum dolor
sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat.</body><topics><topic> sports</topic></topics></post></
posts><followers><follower><id>1</id></follower></followers></user></users>
```

output

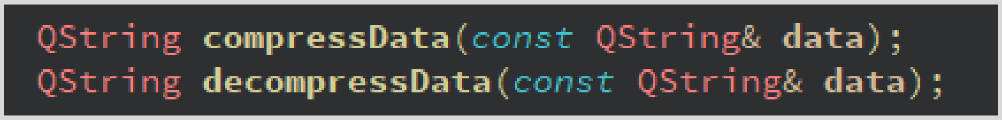


C:/Users/Lenovo/Downloads/sample (2).xml

```
<users>
    <user>
        <id>1</id>
        <name>Ahmed Ali</name>
        <posts>
            <post>
                <body>
                    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.
                </body>
                <topics>
                    <topic>
                        economy
                    </topic>
                    <topic>
                        finance
                    </topic>
                </topics>
            </post>
            <post>
```

	Time complexity	Space complexity
Format	O(N)	O(N)

compression & decompression of Xml files:



```
QString compressData(const QString& data);
QString decompressData(const QString& data);
```

compressData:

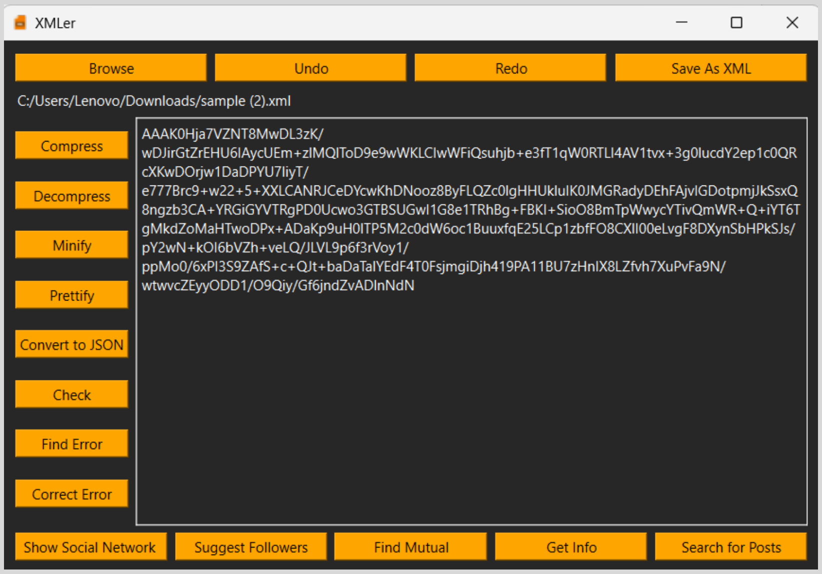
Purpose: This function is designed to compress XML data using the zlib compression algorithm. It takes a QString parameter representing the input XML data, converts it to UTF-8 encoding, applies compression using the **qCompress** function with a specified compression level of 9, and finally converts the compressed data to base64 encoding for easy storage or transmission.

Implementation Logic:
Convert the input QString data to a QByteArray using UTF-8 encoding.
Apply compression to the QByteArray using the **qCompress** function with a compression level of 9.
Convert the compressed QByteArray to a base64-encoded QString.
Return the base64-encoded compressed data.

decompressData:

Purpose: This function is intended to decompress previously compressed XML data. It takes a QString parameter representing the compressed data in base64 encoding, decodes it to a QByteArray, performs decompression using the zlib decompression algorithm (**qUncompress**), and finally converts the decompressed data back to a QString.
Implementation Logic:
Decode the base64-encoded input QString to a QByteArray using UTF-8 encoding.
Perform decompression on the QByteArray using the **qUncompress** function.
Convert the decompressed QByteArray to a QString using UTF-8 encoding.
Return the QString containing the decompressed XML data.

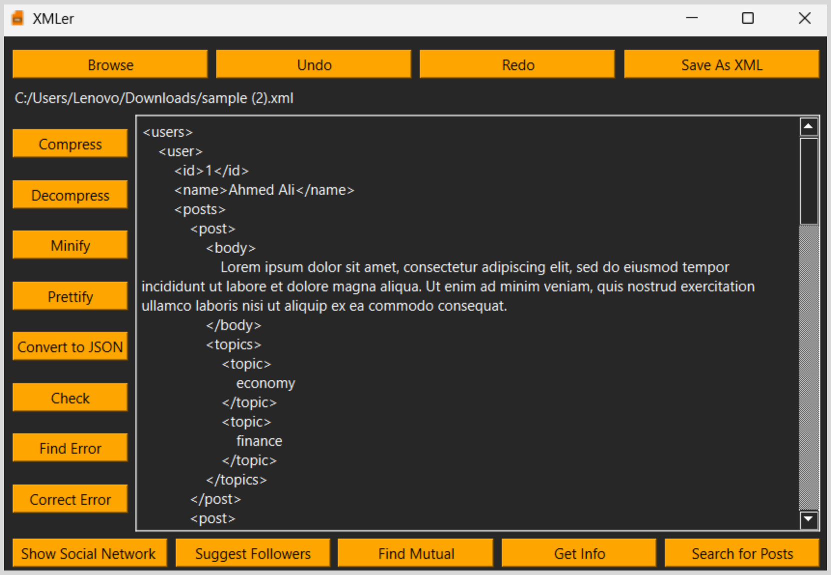
Compress:



C:/Users/Lenovo/Downloads/sample (2).xml

AAAK0Hja7VZNT8MwDL3zK/
wDJirGtZrEHU6lAycUEm+zlMQlToD9e9wWKLClwWFiQsuhjb+e3fT1qW0RTLI4AV1tvx+3g0lucdY2ep1c0QR
cXKwDOrjw1DaDPYU7liyT/
e777Brc9+w22+5+XXLCANRJCeDYcwKhDNooz8ByFLQZc0lgHHUkluIK0JMGRadyDEhFAjvlGDotpmjJkSsxQ
8ngzb3CA+YRGiGYVTRgPD0Ucwo3GTBSUGwl1G8e1TRhBg+FBKl+SioO8BmTpWwycYTivQmWR+Q+iYT6T
gMkdZoMaHTwoDPx+ADaKp9uH0ITP5M2c0dW6oc1BuuxfqE25LCp1zbfFO8CXII00eLvgF8DXynSbHPkSJs/
pY2wN+kOI6bVZh+veLQ/JLVL9p6f3rVoy1/
ppMo0/6xPI3S9ZAfS+c+QJt+baDaTalYEdF4T0FsjmgiDjh419PA11BU7zHnlX8LZfvh7XuPvFa9N/
wtwvcZEyyODD1/O9Qiy/Gf6jndZvADlnNdN

Decompress:



C:/Users/Lenovo/Downloads/sample (2).xml

```
<users>
    <user>
        <id>1</id>
        <name>Ahmed Ali</name>
        <posts>
            <post>
                <body>
                    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.
                </body>
                <topics>
                    <topic>
                        economy
                    </topic>
                    <topic>
                        finance
                    </topic>
                </topics>
            </post>
            <post>
```

	Time complexity	**Space complexity**
Compress	O(N), N is size of input string	O(N)
Decompress	O(N)	O(N)

Undo & redo operations:

Undo:

- Checking if the undo stack (stack_undo) isn't empty.
- If not empty, it saves the current text state to the redo stack (stack_redo).
- Sets the text content to the top (latest) state in the undo stack
- Removes this state from the undo stack, enabling the undo operation by reverting to the previous text state.

We push in the stack the old content in each function before modifying it

Redo:

- Checking if the redo stack (stack_redo) isn't empty.
- If not empty, it saves the current text state to the undo stack (stack_undo).
- Sets the text content to the top (latest) state in the redo stack.
- Removes this state from the redo stack, allowing the redo operation to proceed by applying the next text state.

We push in the stack the old content in Undo function

	Time complexity	**Space complexity**
Undo	O(1)	O(n)
Redo	O(1)	O(n)

Graph Representation

```
void XMLer::drawGraph()
```

The **drawGraph** function is a multi-step process:

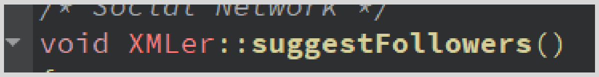
1. **Input Handling:**
    - Checks if there's content in the text area (resultTextEdit). If there is, it proceeds with the XML parsing and visualization.
2. **XML Parsing:**
    - Reads and processes the XML content, splitting it into lines.
    - Utilizes the xmlParse function to interpret the XML structure and extract user-related details.
    - The parsing logic identifies specific XML tags like "user," "name," "id," "body," and "topic" to build a structured representation of users, their IDs, followers, posts, and topics.
3. **Debug Information:**
    - Outputs debugging messages like the count of parsed vertices and details of the first vertex's user information if parsing is successful.
4. **Graph Visualization:**
    - Invokes createGraphVisualization (currently commented out) to potentially generate a visual representation of a social network graph based on the parsed data (parsed).
5. **Error Management:**
    - Handles potential errors during parsing, such as exceeding the vector size or encountering invalid indices when populating the user data structure (userVector).
    - Provides a user-friendly message if no XML file is selected.

Overall, the function orchestrates the process of extracting relevant user information from XML content, potentially visualizing a social network graph, and ensuring error-free operation by validating inputs and handling potential parsing issues.

	Time complexity	Space complexity
drawGraph	O(n)	O(n)

social network analysis

suggest Followers:



```
/* Social Network */
void XMLer::suggestFollowers()
```

- **Concept:** This function analyzes the parsed XML data representing a social network. It suggests followers for each user based on their followers' connections.
- **Implementation Overview:**
    - Parses the XML content to create a user network structure (parsed).
    - Iterates through each user's followers, finds their followers, and suggests potential connections for the original user.
    - Constructs a string (temp) detailing suggested followers for each user.
    - Generates a dialog box to display the suggestions to the user.

Post Search:

```
void XMLer::postSearch()
```

- **Concept:** Allows users to search for posts based on a specified topic within the parsed XML data.
- **Implementation Overview:**
    - Parses the XML content to create a user network structure (parsed).
    - Asks the user for a topic to search.
    - Searches for posts related to the entered topic in the parsed data.
    - Constructs a dialog box to display the found posts.

find Mutual:

```
void XMLer::findMutual()
```

- **Concept**: Finds mutual followers between two specified users based on their IDs**.**
- **Implementation Overview:**
  - Parses the XML content to create a user network structure (parsed).
  - Takes input of user IDs to find mutual followers.
  - Searches for mutual followers between the provided users.
  - Displays the mutual followers in a dialog box.

Grap Info:

```
void XMLer::grapInfo()
```

- **Concept:** Identifies the most influential user (most followers) and the most active user (connected to many users) within the parsed network data.
- **Implementation Overview:**
  - Parses the XML content to create a user network structure (**parsed**).
  - Identifies the user with the most followers and the most connected user.
  - Generates a dialog box to display this information.

	Time complexity	**Space complexity**
suggestFollowers	O(n^3)	O(n)
postSearch	O(n^2)	O(n)
findMutual	O(n^2)	O(n)
grapInfo	O(n^3)	O(n)