TinxyXML2

SAMPLE XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<students>
  <student>
    <name>ONE</name>
    <age>25.64</age>
  </student>
  <student>
    <name>TWO</name>
    <age>25.64</age>
  </student>
</students>
```

CODE TO READ

```cpp
#include "tinyxml2.h"
#include <iostream>

int main() {
    tinyxml2::XMLDocument doc;
    tinyxml2::XMLError result = doc.LoadFile("students.xml");

    if (result != tinyxml2::XML_SUCCESS) {
        std::cerr << "Failed to load file!" << std::endl;
        return 1;
    }

    tinyxml2::XMLElement* root = doc.FirstChildElement("students");
    if (root == nullptr) {
        std::cerr << "Failed to find 'students' element!" << std::endl;
        return 1;
    }

    for (tinyxml2::XMLElement* student = root->FirstChildElement("student"); student != nullptr; student = student->NextSiblingElement("student")) {
        const char* name = student->FirstChildElement("name")->GetText();
        double age = 0;
        student->FirstChildElement("age")->QueryDoubleText(&age);

        std::cout << "Student Name: " << (name ? name : "Unknown") << ", Age: " << age << std::endl;
```

```
    }

    return 0;
}
```

Explanation

The provided code snippet demonstrates how to use the TinyXML-2 library for parsing and processing an XML file in C++. Here's a detailed explanation of each part:

1. **Loading the XML Document**:
   - `tinyxml2::XMLDocument doc;` initializes an XML document object using the TinyXML-2 library.
   - `tinyxml2::XMLError result = doc.LoadFile("students.xml");` attempts to load an XML file named "students.xml" into the `doc` object. The result of the loading operation is stored in `result`, which will be checked to ensure the file loaded successfully.
2. **Error Checking**:
   - The `if` statement checks whether the XML file was loaded successfully by comparing `result` to `tinyxml2::XML_SUCCESS`. If the file didn't load correctly, an error message is printed, and the program returns 1, indicating failure.
3. **Accessing the Root Element**:
   - `tinyxml2::XMLElement* root = doc.FirstChildElement("students");` gets the first child element named "students" from the document. This is typically the root element in this context. If "students" isn't found, `root` will be `nullptr`.
4. **Error Checking for Root Element**:
   - Another `if` statement checks if the root element was found. If `root` is `nullptr`, it indicates the "students" element was not found, and an error message is printed before exiting the program with status 1.
5. **Iterating Over Child Elements**:
   - The `for` loop initializes a pointer `student` to iterate through each "student" element under the "students" root element. It uses `FirstChildElement("student")` to find the first "student" element, and `NextSiblingElement("student")` to move to the next "student" element in the loop.
6. **Processing Each Student Element**:
   - Inside the loop, `const char* name = student->FirstChildElement("name")->GetText();` retrieves the text content of the "name" child element of each "student" element and stores it in `name`.
   - `double age = 0;` initializes a variable to store the age.
   - `student->FirstChildElement("age")->QueryDoubleText(&age);` attempts to retrieve the "age" element's text as a double and store it in the `age` variable.
7. **Outputting Student Information**:
   - The final line inside the loop, `std::cout << "Student Name: " << (name ? name : "Unknown") << ", Age: " << age << std::endl;`, prints out each student's name and age to the console. If the name is not found, it prints "Unknown".

This code snippet effectively demonstrates XML parsing with TinyXML-2, including loading a document, navigating its structure, extracting information from elements, and handling potential errors gracefully.

More Intutive

```
#include "tinyxml2.h"
#include <iostream>
#include <string>

// Function prototypes
bool loadXMLDocument(const std::string& filename, tinyxml2::XMLDocument& doc);
void processStudents(const tinyxml2::XMLDocument& doc);
void processStudent(const tinyxml2::XMLElement* student);
std::string getElementText(const tinyxml2::XMLElement* element, const char* childName,
const std::string& defaultValue = "Unknown");

int main() {
    tinyxml2::XMLDocument doc;
    if (!loadXMLDocument("students.xml", doc)) {
        return 1;
    }

    processStudents(doc);
    return 0;
}

bool loadXMLDocument(const std::string& filename, tinyxml2::XMLDocument& doc) {
    tinyxml2::XMLError result = doc.LoadFile(filename.c_str());
    if (result != tinyxml2::XML_SUCCESS) {
        std::cerr << "Failed to load file '" << filename << "'!" << std::endl;
        return false;
    }
    return true;
}

void processStudents(const tinyxml2::XMLDocument& doc) {
    const tinyxml2::XMLElement* root = doc.FirstChildElement("students");
    if (!root) {
        std::cerr << "Failed to find 'students' element!" << std::endl;
        return;
    }

    for (const tinyxml2::XMLElement* student = root->FirstChildElement("student");
student != nullptr; student = student->NextSiblingElement("student")) {
        processStudent(student);
    }
}

void processStudent(const tinyxml2::XMLElement* student) {
    std::string name = getElementText(student, "name");
    std::string ageText = getElementText(student, "age", "0");
    double age = std::stod(ageText);
```

```
    std::cout << "Student Name: " << name << ", Age: " << age << std::endl;
}

std::string getElementText(const tinyxml2::XMLElement* element, const char* childName,
const std::string& defaultValue) {
    const tinyxml2::XMLElement* childElement = element->FirstChildElement(childName);
    if (childElement && childElement->GetText()) {
        return childElement->GetText();
    }
    return defaultValue;
}
```

MODULAR

```
#include "tinyxml2.h"
#include <iostream>
#include <string>

// Function prototypes
bool loadXMLDocument(const std::string& filename, tinyxml2::XMLDocument& doc);
void processStudents(const tinyxml2::XMLDocument& doc);
void processStudent(const tinyxml2::XMLElement* student);
std::string getElementText(const tinyxml2::XMLElement* element, const char* childName,
const std::string& defaultValue = "Unknown");

int main() {
    tinyxml2::XMLDocument doc;
    if (!loadXMLDocument("students.xml", doc)) {
        return 1;
    }

    processStudents(doc);
    return 0;
}

bool loadXMLDocument(const std::string& filename, tinyxml2::XMLDocument& doc) {
    tinyxml2::XMLError result = doc.LoadFile(filename.c_str());
    if (result != tinyxml2::XML_SUCCESS) {
        std::cerr << "Failed to load file '" << filename << "'!" << std::endl;
        return false;
    }
    return true;
}

void processStudents(const tinyxml2::XMLDocument& doc) {
    const tinyxml2::XMLElement* root = doc.FirstChildElement("students");
    if (!root) {
        std::cerr << "Failed to find 'students' element!" << std::endl;
        return;
    }
```

```
    for (const tinyxml2::XMLElement* student = root->FirstChildElement("student");
student != nullptr; student = student->NextSiblingElement("student")) {
        processStudent(student);
    }
}

void processStudent(const tinyxml2::XMLElement* student) {
    std::string name = getElementText(student, "name");
    std::string ageText = getElementText(student, "age", "0");
    double age = std::stod(ageText);

    std::cout << "Student Name: " << name << ", Age: " << age << std::endl;
}

std::string getElementText(const tinyxml2::XMLElement* element, const char* childName,
const std::string& defaultValue) {
    const tinyxml2::XMLElement* childElement = element->FirstChildElement(childName);
    if (childElement && childElement->GetText()) {
        return childElement->GetText();
    }
    return defaultValue;
}
```

SAMPLE

```
<person>
    <job>
        <title>SOFTWARE ENGINEER</title>
        <office>GOOGLE</office>
    </job>
    <marriage>
        <wife_name>tobi</wife_name>
        <duration>19.0</duration>
    </marriage>
    <personal>
        <name>GOBI</name>
        <age>26</age>
    </personal>
</person>
```

C++ Code

```
// PersonInfo.h
#ifndef PERSON_INFO_H
#define PERSON_INFO_H
```

```
struct JobInfo {
    char title[255];
    char office[255];
};

struct MarriageInfo {
    char wife_name[255];
    double duration;
};

struct PersonalInfo {
    char name[255];
    double age;
};

struct Person {
    JobInfo job;
    MarriageInfo marriage;
    PersonalInfo personal;
};

#endif
```

C CODE

```
#ifdef __cplusplus
extern "C" {
#endif

// Declare your C-compatible interfaces here
void parseXMLFile(const char* filename);

#ifdef __cplusplus
}
#endif
```

```
#include "xml_wrapper.h"
#include "tinyxml2.h"
#include <iostream>

using namespace tinyxml2;

extern "C" {

void parseXMLFile(const char* filename) {
    XMLDocument doc;
    if (doc.LoadFile(filename) == XML_SUCCESS) {
        XMLElement* root = doc.FirstChildElement("person");
```

```
        if (root) {
            XMLElement* personal = root->FirstChildElement("personal");
            if (personal) {
                XMLElement* nameElement = personal->FirstChildElement("name");
                XMLElement* ageElement = personal->FirstChildElement("age");

                if (nameElement) {
                    std::cout << "Name: " << nameElement->GetText() << std::endl;
                }
                if (ageElement) {
                    std::cout << "Age: " << ageElement->GetText() << std::endl;
                }
            }
        }
    } else {
        std::cerr << "Failed to load file: " << filename << std::endl;
    }
}

} // extern "C"
```

```
#include "xml_wrapper.h"

int main() {
    parseXMLFile("path_to_your_xml_file.xml");
    return 0;
}
```

PURE C CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libxml/parser.h>
#include <libxml/tree.h>
#include "PersonInfo.h"

void populatePersonal(xmlNode *node, struct PersonalInfo *personal) {
    for (xmlNode *cur_node = node; cur_node; cur_node = cur_node->next) {
        if (cur_node->type == XML_ELEMENT_NODE) {
            if (strcmp((const char *)cur_node->name, "name") == 0) {
                xmlChar *val = xmlNodeGetContent(cur_node);
                strncpy(personal->name, (const char *)val, 254);
                xmlFree(val);
            } else if (strcmp((const char *)cur_node->name, "age") == 0) {
                xmlChar *val = xmlNodeGetContent(cur_node);
                personal->age = atof((const char *)val);
                xmlFree(val);
            }
        }
```

```
    }
}

// Similar functions can be defined for populateJob and populateMarriage

void parsePerson(xmlNode *node, struct Person *person) {
    for (xmlNode *cur_node = node; cur_node; cur_node = cur_node->next) {
        if (cur_node->type == XML_ELEMENT_NODE) {
            if (strcmp((const char *)cur_node->name, "personal") == 0) {
                populatePersonal(cur_node->children, &person->personal);
            }
            // Add conditions for job and marriage
        }
    }
}

int main(void) {
    xmlDoc *doc = NULL;
    xmlNode *root_element = NULL;

    LIBXML_TEST_VERSION

    // Parse the file and get the DOM
    doc = xmlReadMemory("<person><job><title>SOFTWARE ENGINEER</title>
<office>GOOGLE</office></job><marriage><wife_name>tobi</wife_name>
<duration>19.0</duration></marriage><personal><name>GOBI</name><age>26</age></personal>
</person>", strlen("<person><job><title>SOFTWARE ENGINEER</title><office>GOOGLE</office>
</job><marriage><wife_name>tobi</wife_name><duration>19.0</duration></marriage>
<personal><name>GOBI</name><age>26</age></personal></person>"), "noname.xml", NULL, 0);

    if (doc == NULL) {
        fprintf(stderr, "Failed to parse XML\n");
        return 1;
    }

    root_element = xmlDocGetRootElement(doc);

    struct Person person;
    parsePerson(root_element, &person);

    printf("Name: %s, Age: %f\n", person.personal.name, person.personal.age);

    // Free the document
    xmlFreeDoc(doc);

    // Free the global variables that may have been allocated by the parser
    xmlCleanupParser();

    return 0;
}
```