

Projeto Programação Orientada a Objetos

Generated by Doxygen 1.15.0

Chapter 1

Directory Hierarchy

1.1 Directories

app	??
include	??
app.hpp	??
date.hpp	??
element.hpp	??
file.hpp	??
filename.hpp	??
fileSystem.hpp	??
folder.hpp	??
input.hpp	??
menu.hpp	??
systemConfig.hpp	??
tinyxml2.h	??
utils.hpp	??
src	??
app.cpp	??
date.cpp	??
element.cpp	??
file.cpp	??
filename.cpp	??
fileSystem.cpp	??
folder.cpp	??
input.cpp	??
main.cpp	??
menu.cpp	??
tinyxml2.cpp	??
include	??
app.hpp	??
date.hpp	??
element.hpp	??
file.hpp	??
filename.hpp	??
fileSystem.hpp	??
folder.hpp	??
input.hpp	??
menu.hpp	??
systemConfig.hpp	??
tinyxml2.h	??

utils.hpp	??
src	??
app.cpp	??
date.cpp	??
element.cpp	??
file.cpp	??
filename.cpp	??
fileSystem.cpp	??
folder.cpp	??
input.cpp	??
main.cpp	??
menu.cpp	??
tinysql2.cpp	??

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

tinyxml2	??
--------------------------	-------	----

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

App	??
Date	??
tinyxml2::DynArray< T, INITIAL_SIZE >	??
Element	??
File	??
Folder	??
tinyxml2::Entity	??
Filename	??
FileSystem	??
Input	??
tinyxml2::MemPool	??
tinyxml2::MemPoolT< sizeof(tinyxml2::XMLElement) >	??
tinyxml2::MemPoolT< sizeof(tinyxml2::XMLAttribute) >	??
tinyxml2::MemPoolT< sizeof(tinyxml2::XMLText) >	??
tinyxml2::MemPoolT< sizeof(tinyxml2::XMLComment) >	??
tinyxml2::MemPoolT< ITEM_SIZE >	??
Menu	??
tinyxml2::StrPair	??
SystemConfig	??
Utils	??
tinyxml2::XMLAttribute	??
tinyxml2::XMLConstHandle	??
tinyxml2::XMLHandle	??
tinyxml2::XMLNode	??
tinyxml2::XMLComment	??
tinyxml2::XMLDeclaration	??
tinyxml2::XMLDocument	??
tinyxml2::XMLElement	??
tinyxml2::XMLText	??
tinyxml2::XMLUnknown	??
tinyxml2::XMLUtil	??
tinyxml2::XMLVisitor	??
tinyxml2::XMLPrinter	??

Chapter 4

Class Index

4.1 Class List

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

App	Main application logic	??
Date	Handle date operations and storage	??
tinyxml2::DynArray< T, INITIAL_SIZE >	??
Element	Base class for a filesystem element	??
tinyxml2::Entity	??
File	Handle all file related operations	??
Filename	Handle a file/folder name	??
FileSystem	Handle a filesystem and its operations	??
Folder	Handle all folder related operations	??
Input	Handle input from the user	??
tinyxml2::MemPool	??
tinyxml2::MemPoolT< ITEM_SIZE >	??
Menu	Handle menu output and option chosen	??
tinyxml2::StrPair	??
SystemConfig	Configure system output	??
Utils	Functions with complementary use	??
tinyxml2::XMLAttribute	??
tinyxml2::XMLComment	??
tinyxml2::XMLConstHandle	??
tinyxml2::XMLDeclaration	??
tinyxml2::XMLDocument	??
tinyxml2::XMLElement	??
tinyxml2::XMLHandle	??
tinyxml2::XMLNode	??

tinyxml2::XMLPrinter	??
tinyxml2::XMLText	??
tinyxml2::XMLUnknown	??
tinyxml2::XMLUtil	??
tinyxml2::XMLVisitor	??

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

app/include/app.hpp	??
app/include/date.hpp	??
app/include/element.hpp	??
app/include/file.hpp	??
app/include/filename.hpp	??
app/include/fileSystem.hpp	??
app/include/folder.hpp	??
app/include/input.hpp	??
app/include/menu.hpp	??
app/include/systemConfig.hpp	??
app/include/tinyxml2.h	??
app/include/utils.hpp	??
app/src/app.cpp	??
app/src/date.cpp	??
app/src/element.cpp	??
app/src/file.cpp	??
app/src/filename.cpp	??
app/src/fileSystem.cpp	??
app/src/folder.cpp	??
app/src/input.cpp	??
app/src/main.cpp	??
app/src/menu.cpp	??
app/src/tinyxml2.cpp	??

Chapter 6

Directory Documentation

6.1 app Directory Reference

Directory dependency graph for app:

Directories

- directory [include](#)
- directory [src](#)

6.2 app/include Directory Reference

Directory dependency graph for include:

Files

- file [app.hpp](#)
- file [date.hpp](#)
- file [element.hpp](#)
- file [file.hpp](#)
- file [filename.hpp](#)
- file [fileSystem.hpp](#)
- file [folder.hpp](#)
- file [input.hpp](#)
- file [menu.hpp](#)
- file [systemConfig.hpp](#)
- file [tinyxml2.h](#)
- file [utils.hpp](#)

6.3 app/src Directory Reference

Directory dependency graph for src:

Files

- file [app.cpp](#)
- file [date.cpp](#)
- file [element.cpp](#)
- file [file.cpp](#)
- file [filename.cpp](#)
- file [fileSystem.cpp](#)
- file [folder.cpp](#)
- file [input.cpp](#)
- file [main.cpp](#)
- file [menu.cpp](#)
- file [tinyxml2.cpp](#)

Chapter 7

Namespace Documentation

7.1 tinyxml2 Namespace Reference

Classes

- struct [Entity](#)
- class [StrPair](#)
- class [DynArray](#)
- class [MemPool](#)
- class [MemPoolT](#)
- class [XMLVisitor](#)
- class [XMLUtil](#)
- class [XMLNode](#)
- class [XMLText](#)
- class [XMLComment](#)
- class [XMLDeclaration](#)
- class [XMLUnknown](#)
- class [XMLAttribute](#)
- class [XMLElement](#)
- class [XMLDocument](#)
- class [XMLHandle](#)
- class [XMLConstHandle](#)
- class [XMLPrinter](#)

Enumerations

- enum [XMLError](#) {
 [XML_SUCCESS](#) = 0 , [XML_NO_ATTRIBUTE](#) , [XML_WRONG_ATTRIBUTE_TYPE](#) , [XML_ERROR_FILE_NOT_FOUND](#)
 ,
 [XML_ERROR_FILE_COULD_NOT_BE_OPENED](#) , [XML_ERROR_FILE_READ_ERROR](#) , [XML_ERROR_PARSING_ELEMENT](#)
 , [XML_ERROR_PARSING_ATTRIBUTE](#) ,
 [XML_ERROR_PARSING_TEXT](#) , [XML_ERROR_PARSING_CDATA](#) , [XML_ERROR_PARSING_COMMENT](#)
 , [XML_ERROR_PARSING_DECLARATION](#) ,
 [XML_ERROR_PARSING_UNKNOWN](#) , [XML_ERROR_EMPTY_DOCUMENT](#) , [XML_ERROR_MISMATCHED_ELEMENT](#)
 , [XML_ERROR_PARSING](#) ,
 [XML_CAN_NOT_CONVERT_TEXT](#) , [XML_NO_TEXT_NODE](#) , [XML_ELEMENT_DEPTH_EXCEEDED](#) ,
 [XML_ERROR_COUNT](#) }
- enum [Whitespace](#) { [PRESERVE_WHITESPACE](#) , [COLLAPSE_WHITESPACE](#) , [PEDANTIC_WHITESPACE](#) }

7.1.1 Enumeration Type Documentation

7.1.1.1 Whitespace

enum `tinyxml2::Whitespace`

Enumerator

PRESERVE_WHITESPACE	
COLLAPSE_WHITESPACE	
PEDANTIC_WHITESPACE	

7.1.1.2 XMLError

enum `tinyxml2::XMLError`

Enumerator

XML_SUCCESS	
XML_NO_ATTRIBUTE	
XML_WRONG_ATTRIBUTE_TYPE	
XML_ERROR_FILE_NOT_FOUND	
XML_ERROR_FILE_COULD_NOT_BE_OPENED	
XML_ERROR_FILE_READ_ERROR	
XML_ERROR_PARSING_ELEMENT	
XML_ERROR_PARSING_ATTRIBUTE	
XML_ERROR_PARSING_TEXT	
XML_ERROR_PARSING_CDATA	
XML_ERROR_PARSING_COMMENT	
XML_ERROR_PARSING_DECLARATION	
XML_ERROR_PARSING_UNKNOWN	
XML_ERROR_EMPTY_DOCUMENT	
XML_ERROR_MISMATCHED_ELEMENT	
XML_ERROR_PARSING	
XML_CAN_NOT_CONVERT_TEXT	
XML_NO_TEXT_NODE	
XML_ELEMENT_DEPTH_EXCEEDED	
XML_ERROR_COUNT	

Chapter 8

Class Documentation

8.1 App Class Reference

Main application logic.

```
#include <app.hpp>
```

Public Member Functions

- [App](#) ()
Construct a new [App::App](#) object.
- void [run](#) ()
Shows the main menu and calls submenus accordingly.

8.1.1 Detailed Description

Main application logic.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 App()

```
App::App ()
```

Construct a new [App::App](#) object.

8.1.3 Member Function Documentation

8.1.3.1 run()

```
void App::run ()
```

Shows the main menu and calls submenus accordingly.

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

- [app/include/app.hpp](#)
- [app/src/app.cpp](#)

8.2 Date Class Reference

Handle date operations and storage.

```
#include <date.hpp>
```

Public Member Functions

- [Date](#) ()
Construct a new [Date](#):: [Date](#) object.
- [Date](#) (std::uint16_t day, std::uint16_t month, std::uint16_t year)
- [Date](#) (const std::string &date)
- std::string [getFormattedDate](#) () const
Get the date formatted as a string with "/" separating.
- std::uint16_t [getDay](#) () const
Get the day.
- std::uint16_t [getMonth](#) () const
Get the month.
- std::uint16_t [getYear](#) () const
Get the year.

Static Public Member Functions

- static [Date](#) [convertFileTime](#) (const std::filesystem::file_time_type &ftime)
Converts a filesystem::file_time_type into a [Date](#) scrutured in day, month and year.
- static [Date](#) [now](#) ()
Return current date.

8.2.1 Detailed Description

Handle date operations and storage.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 [Date](#)() [1/3]

```
Date::Date () [default]
```

Construct a new [Date](#):: [Date](#) object.

8.2.2.2 [Date](#)() [2/3]

```
Date::Date (
    std::uint16_t day,
    std::uint16_t month,
    std::uint16_t year)
```

8.2.2.3 Date() [3/3]

```
Date::Date (  
    const std::string & date)
```

8.2.3 Member Function Documentation

8.2.3.1 convertFileTime()

```
Date Date::convertFileTime (  
    const std::filesystem::file_time_type & ftime) [static]
```

Converts a `filesystem::file_time_type` into a [Date](#) structured in day, month and year.

Parameters

<i>ftime</i>	filesystem object to convert
--------------	------------------------------

Returns

[Date](#) [Date](#) converted

8.2.3.2 getDay()

```
uint16_t Date::getDay () const
```

Get the day.

Returns

`uint16_t` day

8.2.3.3 getFormattedDate()

```
string Date::getFormattedDate () const
```

Get the date formatted as a string with "/" separating.

Returns

string [Date](#) formatted

8.2.3.4 getMonth()

```
uint16_t Date::getMonth () const
```

Get the month.

Returns

uint16_t month

8.2.3.5 getYear()

```
uint16_t Date::getYear () const
```

Get the year.

Returns

uint16_t year

8.2.3.6 now()

```
Date Date::now () [static]
```

Return current date.

Returns

[Date](#) Today's date

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

- [app/include/date.hpp](#)
- [app/src/date.cpp](#)

8.3 tinycl2::DynArray< T, INITIAL_SIZE > Class Template Reference

```
#include <tinycl2.h>
```

Public Member Functions

- [DynArray](#) ()
- [~DynArray](#) ()
- void [Clear](#) ()
- void [Push](#) (T t)
- T * [PushArr](#) (size_t count)
- T [Pop](#) ()
- void [PopArr](#) (size_t count)
- bool [Empty](#) () const
- T & [operator\[\]](#) (size_t i)
- const T & [operator\[\]](#) (size_t i) const
- const T & [PeekTop](#) () const
- size_t [Size](#) () const
- size_t [Capacity](#) () const
- void [SwapRemove](#) (size_t i)
- const T * [Mem](#) () const
- T * [Mem](#) ()

8.3.1 Constructor & Destructor Documentation

8.3.1.1 DynArray()

```
template<class T, size_t INITIAL_SIZE>
tinyxml2::DynArray< T, INITIAL_SIZE >::DynArray () [inline]
```

8.3.1.2 ~DynArray()

```
template<class T, size_t INITIAL_SIZE>
tinyxml2::DynArray< T, INITIAL_SIZE >::~~DynArray () [inline]
```

8.3.2 Member Function Documentation

8.3.2.1 Capacity()

```
template<class T, size_t INITIAL_SIZE>
size_t tinyxml2::DynArray< T, INITIAL_SIZE >::Capacity () const [inline]
```

8.3.2.2 Clear()

```
template<class T, size_t INITIAL_SIZE>
void tinyxml2::DynArray< T, INITIAL_SIZE >::Clear () [inline]
```

8.3.2.3 Empty()

```
template<class T, size_t INITIAL_SIZE>
bool tinyxml2::DynArray< T, INITIAL_SIZE >::Empty () const [inline]
```

8.3.2.4 Mem() [1/2]

```
template<class T, size_t INITIAL_SIZE>
T * tinyxml2::DynArray< T, INITIAL_SIZE >::Mem () [inline]
```

8.3.2.5 Mem() [2/2]

```
template<class T, size_t INITIAL_SIZE>
const T * tinyxml2::DynArray< T, INITIAL_SIZE >::Mem () const [inline]
```

8.3.2.6 operator[]() [1/2]

```
template<class T, size_t INITIAL_SIZE>
T & tinyxml2::DynArray< T, INITIAL_SIZE >::operator[] (
    size_t i) [inline]
```

8.3.2.7 operator[]() [2/2]

```
template<class T, size_t INITIAL_SIZE>
const T & tinyxml2::DynArray< T, INITIAL_SIZE >::operator[] (
    size_t i) const [inline]
```

8.3.2.8 PeekTop()

```
template<class T, size_t INITIAL_SIZE>
const T & tinyxml2::DynArray< T, INITIAL_SIZE >::PeekTop () const [inline]
```

8.3.2.9 Pop()

```
template<class T, size_t INITIAL_SIZE>
T tinyxml2::DynArray< T, INITIAL_SIZE >::Pop () [inline]
```

8.3.2.10 PopArr()

```
template<class T, size_t INITIAL_SIZE>
void tinyxml2::DynArray< T, INITIAL_SIZE >::PopArr (
    size_t count) [inline]
```

8.3.2.11 Push()

```
template<class T, size_t INITIAL_SIZE>
void tinyxml2::DynArray< T, INITIAL_SIZE >::Push (
    T t) [inline]
```

8.3.2.12 PushArr()

```
template<class T, size_t INITIAL_SIZE>
T * tinyxml2::DynArray< T, INITIAL_SIZE >::PushArr (
    size_t count) [inline]
```

8.3.2.13 Size()

```
template<class T, size_t INITIAL_SIZE>
size_t tinyxml2::DynArray< T, INITIAL_SIZE >::Size () const [inline]
```

8.3.2.14 SwapRemove()

```
template<class T, size_t INITIAL_SIZE>
void tinyxml2::DynArray< T, INITIAL_SIZE >::SwapRemove (
    size_t i) [inline]
```

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

- [app/include/tinyxml2.h](#)

8.4 Element Class Reference

Base class for a filesystem element.

```
#include <element.hpp>
```

Inheritance diagram for Element:

Collaboration diagram for Element:

Public Member Functions

- [Element](#) (const std::string &[name](#))
Construct a new [Element](#):: [Element](#) object.
- virtual [~Element](#) ()=default
- virtual bool [isFile](#) () const =0
- virtual bool [isFolder](#) () const =0
- const [Filename](#) [getName](#) () const
Get the fullname of the file.
- [Filename](#) & [getName](#) ()
Get filename and allow changes.
- void [setName](#) (const std::string &[name](#))
Change the name of the element.

Protected Attributes

- [Filename](#) [name](#)

8.4.1 Detailed Description

Base class for a filesystem element.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 Element()

```
Element::Element (  
    const std::string & name)
```

Construct a new [Element::Element](#) object.

Parameters

<i>name</i>	
-------------	--

8.4.2.2 ~Element()

```
virtual Element::~Element () [virtual], [default]
```

8.4.3 Member Function Documentation

8.4.3.1 getName() [1/2]

```
Filename & Element::getName ()
```

Get filename and allow changes.

Returns

[Filename&Filename](#)

8.4.3.2 getName() [2/2]

```
const Filename Element::getName () const
```

Get the fullname of the file.

Returns

const string Name.extension

8.4.3.3 isFile()

```
virtual bool Element::isFile () const [pure virtual]
```

Implemented in [File](#), and [Folder](#).

8.4.3.4 isFolder()

```
virtual bool Element::isFolder () const [pure virtual]
```

Implemented in [File](#), and [Folder](#).

8.4.3.5 setName()

```
void Element::setName (  
    const std::string & name)
```

Change the name of the element.

Parameters

<i>newName</i>	New name to attribute
----------------	-----------------------

8.4.4 Member Data Documentation

8.4.4.1 name

```
Filename Element::name [protected]
```

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

- app/include/[element.hpp](#)
- app/src/[element.cpp](#)

8.5 tinyxml2::Entity Struct Reference

Public Attributes

- const char * [pattern](#)
- int [length](#)
- char [value](#)

8.5.1 Member Data Documentation

8.5.1.1 length

```
int tinyxml2::Entity::length
```

8.5.1.2 pattern

```
const char* tinyxml2::Entity::pattern
```

8.5.1.3 value

```
char tinyxml2::Entity::value
```

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

- [app/src/tinyxml2.cpp](#)

8.6 File Class Reference

Handle all file related operations.

```
#include <file.hpp>
```

Inheritance diagram for File:

Collaboration diagram for File:

Public Member Functions

- [File](#) (const std::string &filename)
- [File](#) (const std::string &filename, [Date](#) date, const std::uintmax_t size)
- [File](#) (const std::string &filename, const std::string &date, const std::uintmax_t size)
- void [setDate](#) (const [Date](#) &newDate)
Change the date of the file.
- std::uintmax_t [getSize](#) () const
Get the size occupied by the file.
- const [Date](#) [getDate](#) () const
Get the name formatted as a string.
- bool [isFile](#) () const override
- bool [isFolder](#) () const override

Public Member Functions inherited from [Element](#)

- [Element](#) (const std::string &name)
Construct a new [Element::Element](#) object.
- virtual [~Element](#) ()=default
- const [Filename](#) [getName](#) () const
Get the fullname of the file.
- [Filename](#) & [getName](#) ()
Get filename and allow changes.
- void [setName](#) (const std::string &name)
Change the name of the element.

Additional Inherited Members

Protected Attributes inherited from [Element](#)

- [Filename](#) name

8.6.1 Detailed Description

Handle all file related operations.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 [File\(\)](#) [1/3]

```
File::File (  
    const std::string & filename)
```

8.6.2.2 [File\(\)](#) [2/3]

```
File::File (  
    const std::string & filename,  
    Date date,  
    const std::uintmax_t size)
```

8.6.2.3 [File\(\)](#) [3/3]

```
File::File (  
    const std::string & filename,  
    const std::string & date,  
    const std::uintmax_t size)
```

8.6.3 Member Function Documentation

8.6.3.1 getDate()

```
const Date File::getDate () const
```

Get the name formatted as a string.

Returns

const string [Date](#)

8.6.3.2 getSize()

```
uintmax_t File::getSize () const
```

Get the size occupied by the file.

Returns

uintmax_t Size

8.6.3.3 isFile()

```
bool File::isFile () const [inline], [override], [virtual]
```

Implements [Element](#).

8.6.3.4 isFolder()

```
bool File::isFolder () const [inline], [override], [virtual]
```

Implements [Element](#).

8.6.3.5 setDate()

```
void File::setDate (  
    const Date & newDate)
```

Change the date of the file.

Parameters

<i>newDate</i>	New date
----------------	----------

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

- [app/include/file.hpp](#)
- [app/src/file.cpp](#)

8.7 Filename Class Reference

Handle a file/folder name.

```
#include <filename.hpp>
```

Public Member Functions

- [Filename](#) (const std::string &fullname)
- [Filename](#) (const std::string &name, const std::string &extention)
- void [generateSequentialName](#) (std::uint16_t counter)
Change the name of the file to deal with duplicate names.
- void [setExtension](#) (const std::string &newExtension)
Change the extension of the file.
- void [setName](#) (const std::string &newName)
Update file's name.
- std::string [getFullname](#) () const
Get the fullname of the file.
- std::string [getName](#) () const
Get only the name of the file (no extension).
- std::string [getExtension](#) () const
Get only the extension of the file ('.' not included).

8.7.1 Detailed Description

Handle a file/folder name.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 Filename() [1/2]

```
Filename::Filename (
    const std::string & fullname)
```

8.7.2.2 Filename() [2/2]

```
Filename::Filename (
    const std::string & name,
    const std::string & extention)
```

8.7.3 Member Function Documentation

8.7.3.1 generateSequentialName()

```
void Filename::generateSequentialName (
    std::uint16_t counter)
```

Change the name of the file to deal with duplicate names.

Parameters

<i>counter</i>	Number of the copy
----------------	--------------------

8.7.3.2 getExtension()

```
string Filename::getExtension () const
```

Get only the extension of the file ('.' not included).

Returns

string Extension

8.7.3.3 getFullName()

```
string Filename::getFullname () const
```

Get the fullname of the file.

Returns

string Name.extension

8.7.3.4 getName()

```
string Filename::getName () const
```

Get only the name of the file (no extension).

Returns

string Name

8.7.3.5 setExtension()

```
void Filename::setExtension (  
    const std::string & newExtension)
```

Change the extension of the file.

Parameters

<i>newExtension</i>	New extension (without '.')
---------------------	-----------------------------

8.7.3.6 setName()

```
void Filename::setName (  
    const std::string & newName)
```

Update file's name.

Parameters

<code>newName</code>	New name to give to the file
----------------------	------------------------------

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

- [app/include/filename.hpp](#)
- [app/src/filename.cpp](#)

8.8 FileSystem Class Reference

Handle a filesystem and its operations.

```
#include <fileSystem.hpp>
```

Public Member Functions

- [FileSystem](#) ()
Construct a new [File](#) System:: [File](#) System object.
- [FileSystem](#) (const std::string &rootPath)
- bool [load](#) ()
Loads the folders and files to memory from the absolute path stored.
- bool [load](#) (const std::string &rootPath)
- void [clear](#) ()
Clear/Reset the filesystem.
- std::uint32_t [countFiles](#) () const
Gets the number of files.
- std::uint32_t [countFolders](#) () const
Gets the number of folders.
- std::uintmax_t [memory](#) () const
Gets the memory occupied by all files, folders and the current program execution.
- std::string * [mostElementsFolder](#) () const
Finds the folder with the most elements inside (out of all folders).
- std::string * [leastElementsFolder](#) () const
Finds the folder with the least elements inside (out of all folders).
- std::string * [largestFile](#) () const
Finds the largest file.
- std::string * [largestFolder](#) () const
Finds the largest folder.
- void [saveToXML](#) (const std::string &s) const
Save all the filesystem folders and files to XML format.
- bool [readFromXML](#) (const std::string &s)
Load all the filesystem folders and files from XML format to memory.
- bool [removeAll](#) (const std::string &name, [ElementType](#) type)
Remove all occurrences of a file or folder, determined by type.
- bool [moveFile](#) (const std::string &file, const std::string &newDir)
Move a file to a new folder.
- bool [moveFolder](#) (const std::string &oldDir, const std::string &newDir)
Move a folder into another folder.

- `std::string * getFileDate (const std::string &file)`
Get a file's date as a string.
- `void renameAllFiles (const std::string ¤tName, const std::string &newName)`
Rename all files to "newName".
- `bool copyBatch (const std::string &pattern, const std::string &originDir, const std::string &destinDir)`
Copy.
- `std::optional< std::string > search (const std::string &name, ElementType type)`
Search by the name of a folder/file.
- `void searchAllFolders (std::list< std::string > &li, const std::string &folder) const`
Search all folders with name 'folder' and place them in 'li'.
- `void searchAllFiles (std::list< std::string > &li, const std::string &file) const`
Search all folders for files with name 'file' and place them in 'li'.
- `bool checkDupFiles ()`
Check if there are any files with the same name on the system.
- `void tree (std::ostream &out, std::ostream *mirror=nullptr)`
Output Windows like tree command.
- `void setPath (const std::string &path)`
Set the absolute path to the root directory.
- `const std::string & getPath () const`
Get the current absolute path being used.

8.8.1 Detailed Description

Handle a filesystem and its operations.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 FileSystem() [1/2]

```
FileSystem::FileSystem ()
```

Construct a new File System:: File System object.

8.8.2.2 FileSystem() [2/2]

```
FileSystem::FileSystem (
    const std::string & rootPath)
```

8.8.3 Member Function Documentation

8.8.3.1 checkDupFiles()

```
bool FileSystem::checkDupFiles ()
```

Check if there are any files with the same name on the system.

Returns

true There's duplicate files
false There's no duplicate files

8.8.3.2 clear()

```
void FileSystem::clear ()
```

Clear/Reset the filesystem.

8.8.3.3 copyBatch()

```
bool FileSystem::copyBatch (
    const std::string & pattern,
    const std::string & originDir,
    const std::string & destinDir)
```

Copy.

Parameters

<i>pattern</i>	Pattern to find in
<i>originDir</i>	
<i>destinDir</i>	

Returns

true
false

8.8.3.4 countFiles()

```
uint32_t FileSystem::countFiles () const
```

Gets the number of files.

Returns

uint32_t Number of files

8.8.3.5 countFolders()

```
uint32_t FileSystem::countFolders () const
```

Gets the number of folders.

Returns

uint32_t Number of folders

8.8.3.6 getFileDate()

```
string * FileSystem::getFileDate (
    const std::string & file)
```

Get a file's date as a string.

Note

Caller must delete return value

Parameters

<i>name</i>	Name of the file to search for
-------------	--------------------------------

Returns

string* [Date](#) formatted if found, else nullptr

8.8.3.7 getPath()

```
const string & FileSystem::getPath () const
```

Get the current absolute path being used.

Returns

const string& Path

8.8.3.8 largestFile()

```
string * FileSystem::largestFile () const
```

Finds the largest file.

Returns

std::string* Name of the file, nullptr if error

8.8.3.9 largestFolder()

```
string * FileSystem::largestFolder () const
```

Finds the largest folder.

Returns

std::string* Name of the folder, nullptr if error

8.8.3.10 leastElementsFolder()

```
string * FileSystem::leastElementsFolder () const
```

Finds the folder with the least elements inside (out of all folders).

Note

return value must be deleted after use

Returns

string* Name of the folder, nullptr if error

8.8.3.11 load() [1/2]

```
bool FileSystem::load ()
```

Loads the folders and files to memory from the absolute path stored.

Returns

true Loading succeeded
false Loading failed

8.8.3.12 load() [2/2]

```
bool FileSystem::load (  
    const std::string & rootPath)
```

8.8.3.13 memory()

```
uintmax_t FileSystem::memory () const
```

Gets the memory occupied by all files, folders and the current program execution.

Returns

uintmax_t memory in bytes, 0 if error

8.8.3.14 mostElementsFolder()

```
string * FileSystem::mostElementsFolder () const
```

Finds the folder with the most elements inside (out of all folders).

Note

return value must be deleted after use

Returns

string* Name of the folder, nullptr if error

8.8.3.15 moveFile()

```
bool FileSystem::moveFile (  
    const std::string & file,  
    const std::string & newDir)
```

Move a file to a new folder.

Parameters

<i>file</i>	File to be moved
<i>newFolder</i>	New folder where the file will be moved into

Returns

true Success moving the file
false Failed to move the file

8.8.3.16 moveFolder()

```
bool FileSystem::moveFolder (  
    const std::string & oldDir,  
    const std::string & newDir)
```

Move a folder into another folder.

Parameters

<i>oldDir</i>	Folder to me moved
<i>newDir</i>	Folder to move oldDir into

Returns

true Success
false Failure

8.8.3.17 readFromXML()

```
bool FileSystem::readFromXML (  
    const std::string & s)
```

Load all the filesystem folders and files from XML format to memory.

Parameters

<i>filename</i>	Filename with extension
-----------------	---

Returns

true Success
false Failure

8.8.3.18 removeAll()

```
bool FileSystem::removeAll (  
    const std::string & name,  
    ElementType type)
```

Remove all occurrences of a file or folder, determined by type.

Parameters

<i>name</i>	Name of the folder/file to remove
<i>type</i>	What to remove (folder/file)

Returns

true Success (element deleted successfully)
false Failure (either the element didn't exist or failed to be deleted)

8.8.3.19 renameAllFiles()

```
void FileSystem::renameAllFiles (
    const std::string & currentName,
    const std::string & newName)
```

Rename all files to "newName".

Parameters

<i>currentName</i>	Current name to be changed
<i>newName</i>	New name WITHOUT EXTENSION

8.8.3.20 saveToXML()

```
void FileSystem::saveToXML (
    const std::string & s) const
```

Save all the filesystem folders and files to XML format.

Parameters

<i>filename</i>	Name of the file (with or without extension)
-----------------	--

8.8.3.21 search()

```
optional< string > FileSystem::search (
    const std::string & name,
    ElementType type)
```

Search by the name of a folder/file.

Parameters

<i>name</i>	Name to search for
-------------	--------------------

<i>type</i>	Type Folder/File
-------------	------------------

Returns

nullopt if there are no search results
optional<string> Absolute path to the type element

8.8.3.22 searchAllFiles()

```
void FileSystem::searchAllFiles (
    std::list< std::string > & li,
    const std::string & file) const
```

Search all folders for files with name 'file' and place them in 'li'.

Parameters

<i>li</i>	List where results will be placed
<i>folder</i>	Name of the file to search for

8.8.3.23 searchAllFolders()

```
void FileSystem::searchAllFolders (
    std::list< std::string > & li,
    const std::string & folder) const
```

Search all folders with name 'folder' and place them in 'li'.

Parameters

<i>li</i>	List where results will be placed
<i>folder</i>	Name of the folder to search for

8.8.3.24 setPath()

```
void FileSystem::setPath (
    const std::string & path)
```

Set the absolute path to the root directory.

Parameters

<i>newPath</i>	New path to be set
----------------	--------------------

8.8.3.25 tree()

```
void FileSystem::tree (  
    std::ostream & out,  
    std::ostream * mirror = nullptr)
```

Output Windows like tree command.

Parameters

<i>out</i>	Where to show the tree
<i>mirror</i>	Use to show to multiple interfaces concurrently

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

- [app/include/fileSystem.hpp](#)
- [app/src/fileSystem.cpp](#)

8.9 Folder Class Reference

Handle all folder related operations.

```
#include <folder.hpp>
```

Inheritance diagram for Folder:

Collaboration diagram for Folder:

Public Member Functions

- [Folder](#) (std::string [name](#), [Folder](#) *father)
Construct a new [Folder](#)::[Folder](#) object.
- bool [load](#) (const fs::path &path)
Load all files and folders to memory on this folder.
- void [add](#) (std::unique_ptr< [Element](#) > element)
Add an element to this folder.
- std::unique_ptr< [Element](#) > [remove](#) (const std::string &name, [ElementType](#) type)
Remove an element and return its ownership.
- bool [copyBatch](#) (const std::string &pattern, [Folder](#) *destin)
Copy a batch of files to another folder.
- std::uint32_t [countFiles](#) () const
Get number of files.
- std::uint32_t [countFolders](#) () const
Get number of folders.
- std::uintmax_t [memory](#) () const
Get memory utilization + size of the folder.
- const [Folder](#) * [mostElementsFolder](#) () const
Finds the folder or subfolder with the most number of elements.
- const [Folder](#) * [leastElementsFolder](#) () const
Finds the folder or subfolder with the least number of elements.
- const [File](#) * [largestFile](#) () const
Get the largest file in size.
- const [Folder](#) * [largestFolder](#) (bool isRoot) const
Get the largest folder in size.
- void [saveToXML](#) ([xml::XMLDocument](#) &doc, [xml::XMLElement](#) *parentElem) const
Save a folder to XML document 'doc'.
- void [readFromXML](#) ([xml::XMLElement](#) *dirElem)
Load from an XML file to memory.

- `std::string searchFolder` (const `std::string &name`) const
Get the path to the folder whose name is 'name'.
- void `searchAllFolders` (std::list< `std::string` > &li, const `std::string &name`, const `std::string &path`) const
Search all folders whose name is 'name' and store the path in 'li'.
- `std::string searchFile` (const `std::string &name`) const
Get the path to the file whose name is 'name'.
- void `searchAllFiles` (std::list< `std::string` > &li, const `std::string &name`, const `std::string &path`) const
Search all files whose name is 'name' and store the path in 'li'.
- bool `checkDupFiles` (std::unordered_set< `std::string` > &names)
Check for duplicate files in this folder.
- void `tree` (const `std::string &prefix`, bool isLast, `std::ostream &out`, `std::ostream *mirror`) const
Output Windows like tree command for the current folder.
- bool `removeAll` (const `std::string &name`, `ElementType` type)
Remove type element recursively.
- void `renameAllFiles` (const `std::string ¤tName`, const `std::string &newName`)
Rename all files recursively.
- bool `hasFile` (const `std::string &name`) const
Check if there a file in this folder (does not check subfolders).
- void `setParent` (`Folder *parent`)
Set the folder's parent folder.
- `Folder * getFolderByName` (const `std::string &name`) const
Get a pointer to the folder by name.
- `File * getFileByName` (const `std::string &name`) const
Get a pointer to a file by name search.
- `Folder * getFolderByFileName` (const `std::string &name`) const
Get the parent folder of a file by its name.
- `Folder * getParent` () const
Get the parent folder of the current folder.
- const `std::string getName` () const
Get the folder's name.
- bool `isFile` () const override
- bool `isFolder` () const override

Public Member Functions inherited from `Element`

- `Element` (const `std::string &name`)
Construct a new `Element::Element` object.
- virtual `~Element` ()=default
- const `Filename getName` () const
Get the fullname of the file.
- `Filename & getName` ()
Get filename and allow changes.
- void `setName` (const `std::string &name`)
Change the name of the element.

Additional Inherited Members

Protected Attributes inherited from `Element`

- `Filename name`

8.9.1 Detailed Description

Handle all folder related operations.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 Folder()

```
Folder::Folder (
    std::string name,
    Folder * father)
```

Construct a new `Folder::Folder` object.

Parameters

<i>name</i>	Name of the folder
<i>father</i>	<code>Folder</code> 's parent folder

8.9.3 Member Function Documentation

8.9.3.1 add()

```
void Folder::add (
    std::unique_ptr< Element > element)
```

Add an element to this folder.

Parameters

<i>element</i>	<code>Element</code> to be added
----------------	----------------------------------

8.9.3.2 checkDupFiles()

```
bool Folder::checkDupFiles (
    std::unordered_set< std::string > & names)
```

Check for duplicate files in this folder.

Parameters

<i>names</i>	Names found so far
--------------	--------------------

Returns

true There's duplicates

false There's no duplicates

8.9.3.3 copyBatch()

```
bool Folder::copyBatch (
    const std::string & pattern,
    Folder * destin)
```

Copy a batch of files to another folder.

Parameters

<i>pattern</i>	Pattern to find in the name of the file
<i>destin</i>	Destination folder

Returns

true Copy was successfull

false No file matching the pattern was found or the copy of the files found was not successful

8.9.3.4 countFiles()

```
uint32_t Folder::countFiles () const
```

Get number of files.

Returns

uint32_t Number of files

8.9.3.5 countFolders()

```
uint32_t Folder::countFolders () const
```

Get number of folders.

Returns

uint32_t Number of folders

8.9.3.6 getFileByName()

```
File * Folder::getFileByName (
    const std::string & name) const
```

Get a pointer to a file by name search.

Parameters

<i>name</i>	Name to search for
-------------	--------------------

Returns

File* [File](#) if found, else nullptr

8.9.3.7 getFolderByFileName()

```
Folder * Folder::getFolderByFileName (  
    const std::string & name) const
```

Get the parent folder of a file by its name.

Parameters

<i>name</i>	Name of the file to search
-------------	----------------------------

Returns

Folder* [Folder](#) if found, else nullptr

8.9.3.8 getFolderByName()

```
Folder * Folder::getFolderByName (  
    const std::string & name) const
```

Get a pointer to the folder by name.

Parameters

<i>name</i>	Name to search for
-------------	--------------------

Returns

Folder* [Folder](#) if found, else nullptr

8.9.3.9 getName()

```
const string Folder::getName () const
```

Get the folder's name.

Returns

const string& Name

8.9.3.10 getParent()

```
Folder * Folder::getParent () const
```

Get the parent folder of the current folder.

Returns

Folder* [Folder](#)

8.9.3.11 hasFile()

```
bool Folder::hasFile (
    const std::string & name) const
```

Check if there a file in this folder (does not check subfolders).

Parameters

<i>name</i>	Name to search for
-------------	--------------------

Returns

true [File](#) exists

false [File](#) does not exist

8.9.3.12 isFile()

```
bool Folder::isFile () const [inline], [override], [virtual]
```

Implements [Element](#).

8.9.3.13 isFolder()

```
bool Folder::isFolder () const [inline], [override], [virtual]
```

Implements [Element](#).

8.9.3.14 largestFile()

```
const File * Folder::largestFile () const
```

Get the largest file in size.

Returns

const File* Largest file

8.9.3.15 largestFolder()

```
const Folder * Folder::largestFolder (
    bool isRoot = false) const
```

Get the largest folder in size.

Returns

const Folder* Largest folder

8.9.3.16 leastElementsFolder()

```
const Folder * Folder::leastElementsFolder () const
```

Finds the folder or subfolder with the least number of elements.

Returns

const Folder* Folder found

8.9.3.17 load()

```
bool Folder::load (
    const fs::path & path)
```

Load all files and folders to memory on this folder.

Parameters

<i>path</i>	Path of the current folder to load
-------------	------------------------------------

Returns

true Folder and all it's content loaded successfully
false Path does not exist or it isn't a folder

8.9.3.18 memory()

```
uintmax_t Folder::memory () const
```

Get memory utilization + size of the folder.

Returns

uintmax_t Memory

8.9.3.19 mostElementsFolder()

```
const Folder * Folder::mostElementsFolder () const
```

Finds the folder or subfolder with the most number of elements.

Returns

const Folder* [Folder](#) found

8.9.3.20 readFromXML()

```
void Folder::readFromXML (
    xml::XMLElement * dirElem)
```

Load from an XML file to memory.

Parameters

<i>dirElem</i>	Current folder to load in tinyxml2 format
----------------	---

8.9.3.21 remove()

```
std::unique_ptr< Element > Folder::remove (
    const std::string & name,
    ElementType type)
```

Remove an element and return its ownership.

Parameters

<i>name</i>	Name of the element to be removed
<i>type</i>	Type of the element to be removed

Returns

std::unique_ptr<Element> Ownership or nullptr if failure

8.9.3.22 removeAll()

```
bool Folder::removeAll (
    const std::string & name,
    ElementType type)
```

Remove type element recursively.

Parameters

<i>name</i>	Name to remove
<i>type</i>	Type of the element

Returns

true Removed with success

false Failed to remove or name and type don't exist

8.9.3.23 renameAllFiles()

```
void Folder::renameAllFiles (
    const std::string & currentName,
    const std::string & newName)
```

Rename all files recursively.

Parameters

<i>currentName</i>	Current name to change
<i>newName</i>	New name to change to

8.9.3.24 saveToXML()

```
void Folder::saveToXML (
    xml::XMLDocument & doc,
    xml::XMLElement * parentElem) const
```

Save a folder to XML document 'doc'.

Parameters

<i>doc</i>	Document in tinyxml2 format
<i>parentElem</i>	Parent folder in tinyxml2 format

8.9.3.25 searchAllFiles()

```
void Folder::searchAllFiles (
    std::list< std::string > & li,
    const std::string & name,
    const std::string & path) const
```

Search all files whose name is 'name' and store the path in 'li'.

Parameters

<i>li</i>	List where to store the paths
<i>name</i>	Name to search
<i>path</i>	Initial path, "" if calling on root

8.9.3.26 searchAllFolders()

```
void Folder::searchAllFolders (
    std::list< std::string > & li,
    const std::string & name,
    const std::string & path) const
```

Search all folders whose name is 'name' and store the path in 'li'.

Parameters

<i>li</i>	List where to store the paths
<i>name</i>	Name to search
<i>path</i>	Initial path, "" if calling on root

8.9.3.27 searchFile()

```
string Folder::searchFile (
    const std::string & name) const
```

Get the path to the file whose name is 'name'.

Note

Searches in this [Folder](#) first

Parameters

<i>name</i>	Name of the file to search
-------------	----------------------------

Returns

string Path of the file found or "" if not found

8.9.3.28 searchFolder()

```
string Folder::searchFolder (
    const std::string & name) const
```

Get the path to the folder whose name is 'name'.

Parameters

<i>name</i>	Name of the folder to search
-------------	------------------------------

Returns

string Path of the folder found or "" if not found

8.9.3.29 setParent()

```
void Folder::setParent (
    Folder * parent)
```

Set the folder's parent folder.

Parameters

<i>parent</i>	Pointer to the father
---------------	-----------------------

8.9.3.30 tree()

```
void Folder::tree (
    const std::string & prefix,
    bool isLast,
    std::ostream & out,
    std::ostream * mirror) const
```

Output Windows like tree command for the current folder.

Parameters

<i>prefix</i>	Prefix to the output string
<i>isLast</i>	Whether it is the last file/folder of the current folder
<i>out</i>	Output file
<i>mirror</i>	Output file mirror, if needed

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

- [app/include/folder.hpp](#)
- [app/src/folder.cpp](#)

8.10 Input Class Reference

Handle input from the user.

```
#include <input.hpp>
```

Static Public Member Functions

- static std::string [getString](#) (const std::string &prompt, bool allowEmpty=false)
Get a string as user input.
- static void [wait](#) ()
Wait for user confirmation.

8.10.1 Detailed Description

Handle input from the user.

8.10.2 Member Function Documentation

8.10.2.1 getString()

```
std::string Input::getString (  
    const std::string & prompt,  
    bool allowEmpty = false) [static]
```

Get a string as user input.

Parameters

<i>prompt</i>	Prompt to show
<i>allowEmpty</i>	true to allow "", false by default

Returns

std::string

8.10.2.2 wait()

```
void Input::wait () [static]
```

Wait for user confirmation.

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

- [app/include/input.hpp](#)
- [app/src/input.cpp](#)

8.11 tinycl2::MemPool Class Reference

```
#include <tinycl2.h>
```

Inheritance diagram for tinycl2::MemPool:

Public Member Functions

- [MemPool](#) ()
- virtual [~MemPool](#) ()
- virtual [size_t ItemSize](#) () const =0
- virtual void * [Alloc](#) ()=0
- virtual void [Free](#) (void *)=0
- virtual void [SetTracked](#) ()=0

8.11.1 Constructor & Destructor Documentation

8.11.1.1 MemPool()

```
tinyxml2::MemPool::MemPool () [inline]
```

8.11.1.2 ~MemPool()

```
virtual tinyxml2::MemPool::~~MemPool () [inline], [virtual]
```

8.11.2 Member Function Documentation

8.11.2.1 Alloc()

```
virtual void * tinyxml2::MemPool::Alloc () [pure virtual]
```

Implemented in [tinyxml2::MemPoolT< ITEM_SIZE >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLAttribute\) >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLComment\) >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLElement\) >](#), and [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLText\) >](#).

8.11.2.2 Free()

```
virtual void tinyxml2::MemPool::Free (
    void * ) [pure virtual]
```

Implemented in [tinyxml2::MemPoolT< ITEM_SIZE >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLAttribute\) >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLComment\) >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLElement\) >](#), and [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLText\) >](#).

8.11.2.3 ItemSize()

```
virtual size_t tinyxml2::MemPool::ItemSize () const [pure virtual]
```

Implemented in [tinyxml2::MemPoolT< ITEM_SIZE >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLAttribute\) >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLComment\) >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLElement\) >](#), and [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLText\) >](#).

8.11.2.4 SetTracked()

```
virtual void tinyxml2::MemPool::SetTracked () [pure virtual]
```

Implemented in [tinyxml2::MemPoolT< ITEM_SIZE >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLAttribute\) >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLComment\) >](#), [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLElement\) >](#), and [tinyxml2::MemPoolT< sizeof\(tinyxml2::XMLText\) >](#).

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

- [app/include/tinyxml2.h](#)

8.12 tinyxml2::MemPoolT< ITEM_SIZE > Class Template Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::MemPoolT< ITEM_SIZE >:

Collaboration diagram for tinyxml2::MemPoolT< ITEM_SIZE >:

Public Types

- enum { [ITEMS_PER_BLOCK](#) = (4 * 1024) / ITEM_SIZE }

Public Member Functions

- [MemPoolT](#) ()
- [~MemPoolT](#) ()
- void [Clear](#) ()
- virtual size_t [ItemSize](#) () const override
- size_t [CurrentAllocs](#) () const
- virtual void * [Alloc](#) () override
- virtual void [Free](#) (void *mem) override
- void [Trace](#) (const char *name)
- void [SetTracked](#) () override
- size_t [Untracked](#) () const

Public Member Functions inherited from [tinyxml2::MemPool](#)

- [MemPool](#) ()
- virtual [~MemPool](#) ()

8.12.1 Member Enumeration Documentation

8.12.1.1 anonymous enum

```
template<size_t ITEM_SIZE>
anonymous enum
```

Enumerator

ITEMS_PER_BLOCK	
-----------------	--

8.12.2 Constructor & Destructor Documentation

8.12.2.1 MemPoolT()

```
template<size_t ITEM_SIZE>
tinyxml2::MemPoolT< ITEM_SIZE >::MemPoolT () [inline]
```

8.12.2.2 ~MemPoolT()

```
template<size_t ITEM_SIZE>
tinyxml2::MemPoolT< ITEM_SIZE >::~~MemPoolT () [inline]
```

8.12.3 Member Function Documentation

8.12.3.1 Alloc()

```
template<size_t ITEM_SIZE>
virtual void * tinyxml2::MemPoolT< ITEM_SIZE >::Alloc () [inline], [override], [virtual]
```

Implements [tinyxml2::MemPool](#).

8.12.3.2 Clear()

```
template<size_t ITEM_SIZE>
void tinyxml2::MemPoolT< ITEM_SIZE >::Clear () [inline]
```

8.12.3.3 CurrentAllocs()

```
template<size_t ITEM_SIZE>
size_t tinyxml2::MemPoolT< ITEM_SIZE >::CurrentAllocs () const [inline]
```

8.12.3.4 Free()

```
template<size_t ITEM_SIZE>
virtual void tinyxml2::MemPoolT< ITEM_SIZE >::Free (
    void * mem) [inline], [override], [virtual]
```

Implements [tinyxml2::MemPool](#).

8.12.3.5 ItemSize()

```
template<size_t ITEM_SIZE>
virtual size_t tinyxml2::MemPoolT< ITEM_SIZE >::ItemSize () const [inline], [override], [virtual]
```

Implements [tinyxml2::MemPool](#).

8.12.3.6 SetTracked()

```
template<size_t ITEM_SIZE>
void tinyxml2::MemPoolT< ITEM_SIZE >::SetTracked () [inline], [override], [virtual]
```

Implements [tinyxml2::MemPool](#).

8.12.3.7 Trace()

```
template<size_t ITEM_SIZE>
void tinyxml2::MemPoolT< ITEM_SIZE >::Trace (
    const char * name) [inline]
```

8.12.3.8 Untracked()

```
template<size_t ITEM_SIZE>
size_t tinyxml2::MemPoolT< ITEM_SIZE >::Untracked () const [inline]
```

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

- [app/include/tinyxml2.h](#)

8.13 Menu Class Reference

Handle menu output and option chosen.

```
#include <menu.hpp>
```

Public Member Functions

- [Menu](#) (const std::string &title, const std::vector< std::string > &options)
Construct a new [Menu::Menu](#) object.
- int [show](#) (bool clearTerminal=true)
Show the menu and wait for input.

Static Public Member Functions

- static bool [askYesNo](#) (const std::string &question, bool clearTerminal=false)
Gets user input in Yes/No form.

8.13.1 Detailed Description

Handle menu output and option chosen.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 Menu()

```
Menu::Menu (  
    const std::string & title,  
    const std::vector< std::string > & options)
```

Construct a new [Menu:: Menu](#) object.

Parameters

<i>title</i>	Title of the menu
<i>options</i>	All options of the menu (must be ordered)

8.13.3 Member Function Documentation

8.13.3.1 askYesNo()

```
bool Menu::askYesNo (  
    const std::string & question,  
    bool clearTerminal = false) [static]
```

Gets user input in Yes/No form.

Parameters

<i>question</i>	Question to answer
-----------------	--------------------

Returns

true If user chooses Yes
false If user chooses No

8.13.3.2 show()

```
int Menu::show (  
    bool clearTerminal = true)
```

Show the menu and wait for input.

Returns

int Button chosen, starting on 0 and up to the number of options - 1

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

- [app/include/menu.hpp](#)
- [app/src/menu.cpp](#)

8.14 tinyxml2::StrPair Class Reference

```
#include <tinyxml2.h>
```

Public Types

- enum [Mode](#) {
[NEEDS_ENTITY_PROCESSING](#) = 0x01 , [NEEDS_NEWLINE_NORMALIZATION](#) = 0x02 , [NEEDS_WHITESPACE_COLLAPSING](#) = 0x04 , [TEXT_ELEMENT](#) = [NEEDS_ENTITY_PROCESSING](#) | [NEEDS_NEWLINE_NORMALIZATION](#) ,
[TEXT_ELEMENT_LEAVE_ENTITIES](#) = [NEEDS_NEWLINE_NORMALIZATION](#) , [ATTRIBUTE_NAME](#) = 0 ,
[ATTRIBUTE_VALUE](#) = [NEEDS_ENTITY_PROCESSING](#) | [NEEDS_NEWLINE_NORMALIZATION](#) ,
[ATTRIBUTE_VALUE_LEAVE_ENTITIES](#) = [NEEDS_NEWLINE_NORMALIZATION](#) ,
[COMMENT](#) = [NEEDS_NEWLINE_NORMALIZATION](#) }

Public Member Functions

- [StrPair](#) ()
- [~StrPair](#) ()
- void [Set](#) (char *start, char *end, int flags)
- const char * [GetStr](#) ()
- bool [Empty](#) () const
- void [SetInternedStr](#) (const char *str)
- void [SetStr](#) (const char *str, int flags=0)
- char * [ParseText](#) (char *in, const char *endTag, int strFlags, int *curLineNumPtr)
- char * [ParseName](#) (char *in)
- void [TransferTo](#) ([StrPair](#) *other)
- void [Reset](#) ()

8.14.1 Member Enumeration Documentation

8.14.1.1 Mode

```
enum tinyxml2::StrPair::Mode
```

Enumerator

NEEDS_ENTITY_PROCESSING	
NEEDS_NEWLINE_NORMALIZATION	
NEEDS_WHITESPACE_COLLAPSING	
TEXT_ELEMENT	
TEXT_ELEMENT_LEAVE_ENTITIES	
ATTRIBUTE_NAME	
ATTRIBUTE_VALUE	
ATTRIBUTE_VALUE_LEAVE_ENTITIES	
COMMENT	

8.14.2 Constructor & Destructor Documentation

8.14.2.1 StrPair()

```
tinycl2::StrPair::StrPair () [inline]
```

8.14.2.2 ~StrPair()

```
tinycl2::StrPair::~StrPair ()
```

8.14.3 Member Function Documentation

8.14.3.1 Empty()

```
bool tinycl2::StrPair::Empty () const [inline]
```

8.14.3.2 GetStr()

```
const char * tinycl2::StrPair::GetStr ()
```

8.14.3.3 ParseName()

```
char * tinycl2::StrPair::ParseName (  
    char * in)
```

8.14.3.4 ParseText()

```
char * tinycl2::StrPair::ParseText (  
    char * in,  
    const char * endTag,  
    int strFlags,  
    int * curLineNumPtr)
```

8.14.3.5 Reset()

```
void tinycl2::StrPair::Reset ()
```

8.14.3.6 Set()

```
void tinycl2::StrPair::Set (  
    char * start,  
    char * end,  
    int flags) [inline]
```

8.14.3.7 SetInternedStr()

```
void tinyxml2::StrPair::SetInternedStr (  
    const char * str) [inline]
```

8.14.3.8 SetStr()

```
void tinyxml2::StrPair::SetStr (  
    const char * str,  
    int flags = 0)
```

8.14.3.9 TransferTo()

```
void tinyxml2::StrPair::TransferTo (  
    StrPair * other)
```

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

- [app/include/tinyxml2.h](#)
- [app/src/tinyxml2.cpp](#)

8.15 SystemConfig Class Reference

Configure system output.

```
#include <systemConfig.hpp>
```

Static Public Member Functions

- static void [setUTF8](#) ()
Set terminal with UTF8 charset.

8.15.1 Detailed Description

Configure system output.

8.15.2 Member Function Documentation

8.15.2.1 setUTF8()

```
void SystemConfig::setUTF8 () [inline], [static]
```

Set terminal with UTF8 charset.

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

- [app/include/systemConfig.hpp](#)

8.16 Utils Struct Reference

Functions with complementary use.

```
#include <utils.hpp>
```

Static Public Member Functions

- static std::string [zeroPadding](#) (uint16_t value, std::size_t width)
Places 0's before value.
- static std::string [extractAttribute](#) (const std::string &line, const std::string &attr)
Extract something from a line.
- static bool [hasPattern](#) (const std::string &str, const std::string &pattern)
Check if a string has a certain pattern.
- static void [clear](#) ()
Clear terminal.

8.16.1 Detailed Description

Functions with complementary use.

8.16.2 Member Function Documentation

8.16.2.1 [clear\(\)](#)

```
void Utils::clear () [inline], [static]
```

Clear terminal.

8.16.2.2 [extractAttribute\(\)](#)

```
std::string Utils::extractAttribute (
    const std::string & line,
    const std::string & attr) [inline], [static]
```

Extract something from a line.

Parameters

<i>line</i>	Line
<i>attr</i>	What to extract

Returns

std::string Extracted piece

8.16.2.3 [hasPattern\(\)](#)

```
bool Utils::hasPattern (
    const std::string & str,
    const std::string & pattern) [inline], [static]
```

Check if a string has a certain pattern.

Parameters

<i>str</i>	string
<i>pattern</i>	pattern

Returns

true Has pattern
false Doesn't have pattern

8.16.2.4 zeroPadding()

```
std::string Utils::zeroPadding (
    uint16_t value,
    std::size_t width) [inline], [static]
```

Places 0's before value.

Parameters

<i>value</i>	Value to pad with 0's
<i>width</i>	Width expected

Returns

std::string String padded

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

- app/include/[utils.hpp](#)

8.17 tinyxml2::XMLAttribute Class Reference

```
#include <tinyxml2.h>
```

Public Member Functions

- const char * [Name](#) () const
The name of the attribute.
- const char * [Value](#) () const
The value of the attribute.
- int [GetLineNum](#) () const
Gets the line number the attribute is in, if the document was parsed from a file.
- const XMLAttribute * [Next](#) () const
The next attribute in the list.
- int [IntValue](#) () const
- int64_t [Int64Value](#) () const

- uint64_t [Unsigned64Value](#) () const
- unsigned [UnsignedValue](#) () const
Query as an unsigned integer. See [IntValue\(\)](#).
- bool [BoolValue](#) () const
Query as a boolean. See [IntValue\(\)](#).
- double [DoubleValue](#) () const
Query as a double. See [IntValue\(\)](#).
- float [FloatValue](#) () const
Query as a float. See [IntValue\(\)](#).
- [XMLError QueryIntValue](#) (int *value) const
- [XMLError QueryUnsignedValue](#) (unsigned int *value) const
See [QueryIntValue](#).
- [XMLError QueryInt64Value](#) (int64_t *value) const
See [QueryIntValue](#).
- [XMLError QueryUnsigned64Value](#) (uint64_t *value) const
See [QueryIntValue](#).
- [XMLError QueryBoolValue](#) (bool *value) const
See [QueryIntValue](#).
- [XMLError QueryDoubleValue](#) (double *value) const
See [QueryIntValue](#).
- [XMLError QueryFloatValue](#) (float *value) const
See [QueryIntValue](#).
- void [SetAttribute](#) (const char *value)
Set the attribute to a string value.
- void [SetAttribute](#) (int value)
Set the attribute to value.
- void [SetAttribute](#) (unsigned value)
Set the attribute to value.
- void [SetAttribute](#) (int64_t value)
Set the attribute to value.
- void [SetAttribute](#) (uint64_t value)
Set the attribute to value.
- void [SetAttribute](#) (bool value)
Set the attribute to value.
- void [SetAttribute](#) (double value)
Set the attribute to value.
- void [SetAttribute](#) (float value)
Set the attribute to value.

Friends

- class [XMLElement](#)

8.17.1 Detailed Description

An attribute is a name-value pair. Elements have an arbitrary number of attributes, each with a unique name.

Note

The attributes are not XMLNodes. You may only query the [Next\(\)](#) attribute in a list.

8.17.2 Member Function Documentation

8.17.2.1 BoolValue()

```
bool tinyxml2::XMLAttribute::BoolValue () const [inline]
```

Query as a boolean. See [IntValue\(\)](#).

8.17.2.2 DoubleValue()

```
double tinyxml2::XMLAttribute::DoubleValue () const [inline]
```

Query as a double. See [IntValue\(\)](#).

8.17.2.3 FloatValue()

```
float tinyxml2::XMLAttribute::FloatValue () const [inline]
```

Query as a float. See [IntValue\(\)](#).

8.17.2.4 GetLineNum()

```
int tinyxml2::XMLAttribute::GetLineNum () const [inline]
```

Gets the line number the attribute is in, if the document was parsed from a file.

8.17.2.5 Int64Value()

```
int64_t tinyxml2::XMLAttribute::Int64Value () const [inline]
```

8.17.2.6 IntValue()

```
int tinyxml2::XMLAttribute::IntValue () const [inline]
```

IntValue interprets the attribute as an integer, and returns the value. If the value isn't an integer, 0 will be returned. There is no error checking; use [QueryIntValue\(\)](#) if you need error checking.

8.17.2.7 Name()

```
const char * tinyxml2::XMLAttribute::Name () const
```

The name of the attribute.

8.17.2.8 Next()

```
const XMLAttribute * tinyxml2::XMLAttribute::Next () const [inline]
```

The next attribute in the list.

8.17.2.9 QueryBoolValue()

```
XML_Error tinyxml2::XMLAttribute::QueryBoolValue (
    bool * value) const
```

See QueryIntValue.

8.17.2.10 QueryDoubleValue()

```
XML_Error tinyxml2::XMLAttribute::QueryDoubleValue (
    double * value) const
```

See QueryIntValue.

8.17.2.11 QueryFloatValue()

```
XML_Error tinyxml2::XMLAttribute::QueryFloatValue (
    float * value) const
```

See QueryIntValue.

8.17.2.12 QueryInt64Value()

```
XML_Error tinyxml2::XMLAttribute::QueryInt64Value (
    int64_t * value) const
```

See QueryIntValue.

8.17.2.13 QueryIntValue()

```
XML_Error tinyxml2::XMLAttribute::QueryIntValue (
    int * value) const
```

QueryIntValue interprets the attribute as an integer, and returns the value in the provided parameter. The function will return XML_SUCCESS on success, and XML_WRONG_ATTRIBUTE_TYPE if the conversion is not successful.

8.17.2.14 QueryUnsigned64Value()

```
XML_Error tinyxml2::XMLAttribute::QueryUnsigned64Value (
    uint64_t * value) const
```

See QueryIntValue.

8.17.2.15 QueryUnsignedValue()

```
XML_Error tinyxml2::XMLAttribute::QueryUnsignedValue (
    unsigned int * value) const
```

See QueryIntValue.

8.17.2.16 SetAttribute() [1/8]

```
void tinyxml2::XMLAttribute::SetAttribute (
    bool value)
```

Set the attribute to value.

8.17.2.17 SetAttribute() [2/8]

```
void tinyxml2::XMLAttribute::SetAttribute (
    const char * value)
```

Set the attribute to a string value.

8.17.2.18 SetAttribute() [3/8]

```
void tinyxml2::XMLAttribute::SetAttribute (
    double value)
```

Set the attribute to value.

8.17.2.19 SetAttribute() [4/8]

```
void tinyxml2::XMLAttribute::SetAttribute (
    float value)
```

Set the attribute to value.

8.17.2.20 SetAttribute() [5/8]

```
void tinyxml2::XMLAttribute::SetAttribute (
    int value)
```

Set the attribute to value.

8.17.2.21 SetAttribute() [6/8]

```
void tinyxml2::XMLAttribute::SetAttribute (
    int64_t value)
```

Set the attribute to value.

8.17.2.22 SetAttribute() [7/8]

```
void tinyxml2::XMLAttribute::SetAttribute (
    uint64_t value)
```

Set the attribute to value.

8.17.2.23 SetAttribute() [8/8]

```
void tinyxml2::XMLAttribute::SetAttribute (
    unsigned value)
```

Set the attribute to value.

8.17.2.24 Unsigned64Value()

```
uint64_t tinyxml2::XMLAttribute::Unsigned64Value () const [inline]
```

8.17.2.25 UnsignedValue()

```
unsigned tinyxml2::XMLAttribute::UnsignedValue () const [inline]
```

Query as an unsigned integer. See [IntValue\(\)](#).

8.17.2.26 Value()

```
const char * tinyxml2::XMLAttribute::Value () const
```

The value of the attribute.

8.17.3 Friends And Related Symbol Documentation**8.17.3.1 XMLElement**

```
friend class XMLElement [friend]
```

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

- [app/include/tinyxml2.h](#)
- [app/src/tinyxml2.cpp](#)

8.18 tinyxml2::XMLComment Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLComment:

Collaboration diagram for tinyxml2::XMLComment:

Public Member Functions

- virtual [XMLComment](#) * [ToComment](#) () override
Safely cast to a [Comment](#), or null.
- virtual const [XMLComment](#) * [ToComment](#) () const override
- virtual bool [Accept](#) ([XMLVisitor](#) *visitor) const override
- virtual [XMLNode](#) * [ShallowClone](#) ([XMLDocument](#) *document) const override
- virtual bool [ShallowEqual](#) (const [XMLNode](#) *compare) const override

Public Member Functions inherited from [tinyxml2::XMLNode](#)

- const [XMLDocument](#) * [GetDocument](#) () const
Get the [XMLDocument](#) that owns this [XMLNode](#).
- [XMLDocument](#) * [GetDocument](#) ()
Get the [XMLDocument](#) that owns this [XMLNode](#).
- virtual [XMLElement](#) * [ToElement](#) ()
Safely cast to an [Element](#), or null.
- virtual [XMLText](#) * [ToText](#) ()
Safely cast to [Text](#), or null.
- virtual [XMLDocument](#) * [ToDocument](#) ()
Safely cast to a [Document](#), or null.
- virtual [XMLDeclaration](#) * [ToDeclaration](#) ()
Safely cast to a [Declaration](#), or null.
- virtual [XMLUnknown](#) * [ToUnknown](#) ()
Safely cast to an [Unknown](#), or null.
- virtual const [XMLElement](#) * [ToElement](#) () const
- virtual const [XMLText](#) * [ToText](#) () const
- virtual const [XMLDocument](#) * [ToDocument](#) () const
- virtual const [XMLDeclaration](#) * [ToDeclaration](#) () const
- virtual const [XMLUnknown](#) * [ToUnknown](#) () const
- int [ChildElementCount](#) (const char *value) const
- int [ChildElementCount](#) () const
- const char * [Value](#) () const
- void [SetValue](#) (const char *val, bool staticMem=false)
- int [GetLineNum](#) () const
Gets the line number the node is in, if the document was parsed from a file.
- const [XMLNode](#) * [Parent](#) () const
Get the parent of this node on the DOM.
- [XMLNode](#) * [Parent](#) ()
- bool [NoChildren](#) () const
Returns true if this node has no children.
- const [XMLNode](#) * [FirstChild](#) () const
Get the first child node, or null if none exists.
- [XMLNode](#) * [FirstChild](#) ()
- const [XMLElement](#) * [FirstChildElement](#) (const char *name=0) const
- [XMLElement](#) * [FirstChildElement](#) (const char *name=0)
- const [XMLNode](#) * [LastChild](#) () const
Get the last child node, or null if none exists.
- [XMLNode](#) * [LastChild](#) ()
- const [XMLElement](#) * [LastChildElement](#) (const char *name=0) const
- [XMLElement](#) * [LastChildElement](#) (const char *name=0)
- const [XMLNode](#) * [PreviousSibling](#) () const

Get the previous (left) sibling node of this node.

- [XMLNode](#) * [PreviousSibling](#) ()
- const [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0) const

Get the previous (left) sibling element of this node, with an optionally supplied name.

- [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0)
- const [XMLNode](#) * [NextSibling](#) () const

Get the next (right) sibling node of this node.

- [XMLNode](#) * [NextSibling](#) ()
- const [XMLElement](#) * [NextSiblingElement](#) (const char *name=0) const

Get the next (right) sibling element of this node, with an optionally supplied name.

- [XMLElement](#) * [NextSiblingElement](#) (const char *name=0)
- [XMLNode](#) * [InsertEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [LinkEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertFirstChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertAfterChild](#) ([XMLNode](#) *afterThis, [XMLNode](#) *addThis)
- void [DeleteChildren](#) ()
- void [DeleteChild](#) ([XMLNode](#) *node)
- [XMLNode](#) * [DeepClone](#) ([XMLDocument](#) *target) const
- void [SetUserData](#) (void *userData)
- void * [GetUserData](#) () const

Protected Member Functions

- [XMLComment](#) ([XMLDocument](#) *doc)
- virtual [~XMLComment](#) ()
- char * [ParseDeep](#) (char *p, [StrPair](#) *parentEndTag, int *curLineNumPtr) override

Protected Member Functions inherited from [tinyxml2::XMLNode](#)

- [XMLNode](#) ([XMLDocument](#) *)
- virtual [~XMLNode](#) ()

Friends

- class [XMLDocument](#)

Additional Inherited Members

Protected Attributes inherited from [tinyxml2::XMLNode](#)

- [XMLDocument](#) * [_document](#)
- [XMLNode](#) * [_parent](#)
- [StrPair](#) [_value](#)
- int [_parseLineNum](#)
- [XMLNode](#) * [_firstChild](#)
- [XMLNode](#) * [_lastChild](#)
- [XMLNode](#) * [_prev](#)
- [XMLNode](#) * [_next](#)
- void * [_userData](#)

8.18.1 Detailed Description

An XML Comment.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 XMLComment()

```
tinyxml2::XMLComment::XMLComment (
    XMLDocument * doc) [explicit], [protected]
```

8.18.2.2 ~XMLComment()

```
tinyxml2::XMLComment::~~XMLComment () [protected], [virtual]
```

8.18.3 Member Function Documentation

8.18.3.1 Accept()

```
bool tinyxml2::XMLComment::Accept (
    XMLVisitor * visitor) const [override], [virtual]
```

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the [XMLVisitor](#) interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinyxml2::XMLNode](#).

8.18.3.2 ParseDeep()

```
char * tinyxml2::XMLComment::ParseDeep (  
    char * p,  
    StrPair * parentEndTag,  
    int * curLineNumPtr) [override], [protected], [virtual]
```

Reimplemented from [tinyxml2::XMLNode](#).

8.18.3.3 ShallowClone()

```
XMLNode * tinyxml2::XMLComment::ShallowClone (  
    XMLDocument * document) const [override], [virtual]
```

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (this->[GetDocument\(\)](#))

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

8.18.3.4 ShallowEqual()

```
bool tinyxml2::XMLComment::ShallowEqual (  
    const XMLNode * compare) const [override], [virtual]
```

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

8.18.3.5 ToComment() [1/2]

```
virtual const XMLComment * tinyxml2::XMLComment::ToComment () const [inline], [override],  
[virtual]
```

Reimplemented from [tinyxml2::XMLNode](#).

8.18.3.6 ToComment() [2/2]

```
virtual XMLComment * tinyxml2::XMLComment::ToComment () [inline], [override], [virtual]
```

Safely cast to a Comment, or null.

Reimplemented from [tinyxml2::XMLNode](#).

8.18.4 Friends And Related Symbol Documentation

8.18.4.1 XMLDocument

```
friend class XMLDocument [friend]
```

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

- [app/include/tinyxml2.h](#)
- [app/src/tinyxml2.cpp](#)

8.19 tinyxml2::XMLConstHandle Class Reference

```
#include <tinyxml2.h>
```

Public Member Functions

- [XMLConstHandle](#) (const [XMLNode](#) *node)
- [XMLConstHandle](#) (const [XMLNode](#) &node)
- [XMLConstHandle](#) (const XMLConstHandle &ref)
- [XMLConstHandle](#) & [operator=](#) (const [XMLConstHandle](#) &ref)
- const [XMLConstHandle](#) [FirstChild](#) () const
- const [XMLConstHandle](#) [FirstChildElement](#) (const char *name=0) const
- const [XMLConstHandle](#) [LastChild](#) () const
- const [XMLConstHandle](#) [LastChildElement](#) (const char *name=0) const
- const [XMLConstHandle](#) [PreviousSibling](#) () const
- const [XMLConstHandle](#) [PreviousSiblingElement](#) (const char *name=0) const
- const [XMLConstHandle](#) [NextSibling](#) () const
- const [XMLConstHandle](#) [NextSiblingElement](#) (const char *name=0) const
- const [XMLNode](#) * [ToNode](#) () const
- const [XMLElement](#) * [ToElement](#) () const
- const [XMLText](#) * [ToText](#) () const
- const [XMLUnknown](#) * [ToUnknown](#) () const
- const [XMLDeclaration](#) * [ToDeclaration](#) () const

8.19.1 Detailed Description

A variant of the [XMLHandle](#) class for working with const XMLNodes and Documents. It is the same in all regards, except for the 'const' qualifiers. See [XMLHandle](#) for API.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 XMLConstHandle() [1/3]

```
tinyxml2::XMLConstHandle::XMLConstHandle (
    const XMLNode * node) [inline], [explicit]
```

8.19.2.2 XMLConstHandle() [2/3]

```
tinyxml2::XMLConstHandle::XMLConstHandle (  
    const XMLNode & node) [inline], [explicit]
```

8.19.2.3 XMLConstHandle() [3/3]

```
tinyxml2::XMLConstHandle::XMLConstHandle (  
    const XMLConstHandle & ref) [inline]
```

8.19.3 Member Function Documentation

8.19.3.1 FirstChild()

```
const XMLConstHandle tinyxml2::XMLConstHandle::FirstChild () const [inline]
```

8.19.3.2 FirstChildElement()

```
const XMLConstHandle tinyxml2::XMLConstHandle::FirstChildElement (  
    const char * name = 0) const [inline]
```

8.19.3.3 LastChild()

```
const XMLConstHandle tinyxml2::XMLConstHandle::LastChild () const [inline]
```

8.19.3.4 LastChildElement()

```
const XMLConstHandle tinyxml2::XMLConstHandle::LastChildElement (  
    const char * name = 0) const [inline]
```

8.19.3.5 NextSibling()

```
const XMLConstHandle tinyxml2::XMLConstHandle::NextSibling () const [inline]
```

8.19.3.6 NextSiblingElement()

```
const XMLConstHandle tinyxml2::XMLConstHandle::NextSiblingElement (  
    const char * name = 0) const [inline]
```

8.19.3.7 operator=()

```
XMLConstHandle & tinyxml2::XMLConstHandle::operator= (  
    const XMLConstHandle & ref) [inline]
```

8.19.3.8 PreviousSibling()

```
const XMLConstHandle tinyxml2::XMLConstHandle::PreviousSibling () const [inline]
```

8.19.3.9 PreviousSiblingElement()

```
const XMLConstHandle tinyxml2::XMLConstHandle::PreviousSiblingElement (
    const char * name = 0) const [inline]
```

8.19.3.10 ToDeclaration()

```
const XMLDeclaration * tinyxml2::XMLConstHandle::ToDeclaration () const [inline]
```

8.19.3.11 ToElement()

```
const XMLElement * tinyxml2::XMLConstHandle::ToElement () const [inline]
```

8.19.3.12 ToNode()

```
const XMLNode * tinyxml2::XMLConstHandle::ToNode () const [inline]
```

8.19.3.13 ToText()

```
const XMLText * tinyxml2::XMLConstHandle::ToText () const [inline]
```

8.19.3.14 ToUnknown()

```
const XMLUnknown * tinyxml2::XMLConstHandle::ToUnknown () const [inline]
```

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

- [app/include/tinyxml2.h](#)

8.20 tinyxml2::XMLDeclaration Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLDeclaration:

Collaboration diagram for tinyxml2::XMLDeclaration:

Public Member Functions

- virtual [XMLDeclaration](#) * [ToDeclaration](#) () override
Safely cast to a Declaration, or null.
- virtual const [XMLDeclaration](#) * [ToDeclaration](#) () const override
- virtual bool [Accept](#) ([XMLVisitor](#) *visitor) const override
- virtual [XMLNode](#) * [ShallowClone](#) ([XMLDocument](#) *document) const override
- virtual bool [ShallowEqual](#) (const [XMLNode](#) *compare) const override

Public Member Functions inherited from [tinyxml2::XMLNode](#)

- const [XMLDocument](#) * [GetDocument](#) () const
Get the XMLDocument that owns this XMLNode.
- [XMLDocument](#) * [GetDocument](#) ()
Get the XMLDocument that owns this XMLNode.
- virtual [XMLElement](#) * [ToElement](#) ()
Safely cast to an Element, or null.
- virtual [XMLText](#) * [ToText](#) ()
Safely cast to Text, or null.
- virtual [XMLComment](#) * [ToComment](#) ()
Safely cast to a Comment, or null.
- virtual [XMLDocument](#) * [ToDocument](#) ()
Safely cast to a Document, or null.
- virtual [XMLUnknown](#) * [ToUnknown](#) ()
Safely cast to an Unknown, or null.
- virtual const [XMLElement](#) * [ToElement](#) () const
- virtual const [XMLText](#) * [ToText](#) () const
- virtual const [XMLComment](#) * [ToComment](#) () const
- virtual const [XMLDocument](#) * [ToDocument](#) () const
- virtual const [XMLUnknown](#) * [ToUnknown](#) () const
- int [ChildElementCount](#) (const char *value) const
- int [ChildElementCount](#) () const
- const char * [Value](#) () const
- void [SetValue](#) (const char *val, bool staticMem=false)
- int [GetLineNum](#) () const
Gets the line number the node is in, if the document was parsed from a file.
- const [XMLNode](#) * [Parent](#) () const
Get the parent of this node on the DOM.
- [XMLNode](#) * [Parent](#) ()
- bool [NoChildren](#) () const
Returns true if this node has no children.
- const [XMLNode](#) * [FirstChild](#) () const
Get the first child node, or null if none exists.
- [XMLNode](#) * [FirstChild](#) ()
- const [XMLElement](#) * [FirstChildElement](#) (const char *name=0) const
- [XMLElement](#) * [FirstChildElement](#) (const char *name=0)
- const [XMLNode](#) * [LastChild](#) () const
Get the last child node, or null if none exists.
- [XMLNode](#) * [LastChild](#) ()
- const [XMLElement](#) * [LastChildElement](#) (const char *name=0) const
- [XMLElement](#) * [LastChildElement](#) (const char *name=0)
- const [XMLNode](#) * [PreviousSibling](#) () const

Get the previous (left) sibling node of this node.

- [XMLNode](#) * [PreviousSibling](#) ()
- const [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0) const

Get the previous (left) sibling element of this node, with an optionally supplied name.

- [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0)
- const [XMLNode](#) * [NextSibling](#) () const

Get the next (right) sibling node of this node.

- [XMLNode](#) * [NextSibling](#) ()
- const [XMLElement](#) * [NextSiblingElement](#) (const char *name=0) const

Get the next (right) sibling element of this node, with an optionally supplied name.

- [XMLElement](#) * [NextSiblingElement](#) (const char *name=0)
- [XMLNode](#) * [InsertEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [LinkEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertFirstChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertAfterChild](#) ([XMLNode](#) *afterThis, [XMLNode](#) *addThis)
- void [DeleteChildren](#) ()
- void [DeleteChild](#) ([XMLNode](#) *node)
- [XMLNode](#) * [DeepClone](#) ([XMLDocument](#) *target) const
- void [SetUserData](#) (void *userData)
- void * [GetUserData](#) () const

Protected Member Functions

- [XMLDeclaration](#) ([XMLDocument](#) *doc)
- virtual [~XMLDeclaration](#) ()
- char * [ParseDeep](#) (char *p, [StrPair](#) *parentEndTag, int *curLineNumPtr) override

Protected Member Functions inherited from [tinyxml2::XMLNode](#)

- [XMLNode](#) ([XMLDocument](#) *)
- virtual [~XMLNode](#) ()

Friends

- class [XMLDocument](#)

Additional Inherited Members

Protected Attributes inherited from [tinyxml2::XMLNode](#)

- [XMLDocument](#) * [_document](#)
- [XMLNode](#) * [_parent](#)
- [StrPair](#) [_value](#)
- int [_parseLineNum](#)
- [XMLNode](#) * [_firstChild](#)
- [XMLNode](#) * [_lastChild](#)
- [XMLNode](#) * [_prev](#)
- [XMLNode](#) * [_next](#)
- void * [_userData](#)

8.20.1 Detailed Description

In correct XML the declaration is the first entry in the file.

```
<?xml version="1.0" standalone="yes"?>
```

TinyXML-2 will happily read or write files without a declaration, however.

The text of the declaration isn't interpreted. It is parsed and written as a string.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 XMLDeclaration()

```
tinycl2::XMLDeclaration::XMLDeclaration (
    XMLDocument * doc) [explicit], [protected]
```

8.20.2.2 ~XMLDeclaration()

```
tinycl2::XMLDeclaration::~~XMLDeclaration () [protected], [virtual]
```

8.20.3 Member Function Documentation

8.20.3.1 Accept()

```
bool tinycl2::XMLDeclaration::Accept (
    XMLVisitor * visitor) const [override], [virtual]
```

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the [XMLVisitor](#) interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinycl2Doc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinycl2::XMLNode](#).

8.20.3.2 ParseDeep()

```
char * tinyxml2::XMLDeclaration::ParseDeep (  
    char * p,  
    StrPair * parentEndTag,  
    int * curLineNumPtr) [override], [protected], [virtual]
```

Reimplemented from [tinyxml2::XMLNode](#).

8.20.3.3 ShallowClone()

```
XMLNode * tinyxml2::XMLDeclaration::ShallowClone (  
    XMLDocument * document) const [override], [virtual]
```

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (this->[GetDocument\(\)](#))

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

8.20.3.4 ShallowEqual()

```
bool tinyxml2::XMLDeclaration::ShallowEqual (  
    const XMLNode * compare) const [override], [virtual]
```

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

8.20.3.5 ToDeclaration() [1/2]

```
virtual const XMLDeclaration * tinyxml2::XMLDeclaration::ToDeclaration () const [inline],  
[override], [virtual]
```

Reimplemented from [tinyxml2::XMLNode](#).

8.20.3.6 ToDeclaration() [2/2]

```
virtual XMLDeclaration * tinyxml2::XMLDeclaration::ToDeclaration () [inline], [override],  
[virtual]
```

Safely cast to a Declaration, or null.

Reimplemented from [tinyxml2::XMLNode](#).

8.20.4 Friends And Related Symbol Documentation

8.20.4.1 XMLDocument

```
friend class XMLDocument [friend]
```

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

- [app/include/tinyxml2.h](#)
- [app/src/tinyxml2.cpp](#)

8.21 tinyxml2::XMLDocument Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLDocument:

Collaboration diagram for tinyxml2::XMLDocument:

Public Member Functions

- [XMLDocument](#) (bool processEntities=true, [Whitespace](#) whitespaceMode=[PRESERVE_WHITESPACE](#))
constructor
- [~XMLDocument](#) ()
- virtual [XMLDocument](#) * [ToDocument](#) () override
Safely cast to a Document, or null.
- virtual const [XMLDocument](#) * [ToDocument](#) () const override
- [XMLError](#) [Parse](#) (const char *xml, size_t nBytes=static_cast< size_t >(-1))
- [XMLError](#) [LoadFile](#) (const char *filename)
- [XMLError](#) [LoadFile](#) (FILE *)
- [XMLError](#) [SaveFile](#) (const char *filename, bool compact=false)
- [XMLError](#) [SaveFile](#) (FILE *fp, bool compact=false)
- bool [ProcessEntities](#) () const
- [Whitespace](#) [WhitespaceMode](#) () const
- bool [HasBOM](#) () const
- void [SetBOM](#) (bool useBOM)
- [XMLElement](#) * [RootElement](#) ()
- const [XMLElement](#) * [RootElement](#) () const
- void [Print](#) ([XMLPrinter](#) *streamer=0) const
- virtual bool [Accept](#) ([XMLVisitor](#) *visitor) const override
- [XMLElement](#) * [NewElement](#) (const char *name)
- [XMLComment](#) * [NewComment](#) (const char *comment)
- [XMLText](#) * [NewText](#) (const char *text)
- [XMLDeclaration](#) * [NewDeclaration](#) (const char *text=0)
- [XMLUnknown](#) * [NewUnknown](#) (const char *text)
- void [DeleteNode](#) ([XMLNode](#) *node)
- void [ClearError](#) ()
Clears the error flags.
- bool [Error](#) () const
Return true if there was an error parsing the document.

- [XMLError](#) [ErrorID](#) () const
Return the errorID.
- const char * [ErrorName](#) () const
- const char * [ErrorStr](#) () const
- void [PrintError](#) () const
A (trivial) utility function that prints the [ErrorStr\(\)](#) to stdout.
- int [ErrorLineNum](#) () const
Return the line where the error occurred, or zero if unknown.
- void [Clear](#) ()
Clear the document, resetting it to the initial state.
- void [DeepCopy](#) ([XMLDocument](#) *target) const
- char * [Identify](#) (char *p, [XMLNode](#) **node, bool first)
- void [MarkInUse](#) (const [XMLNode](#) *const)
- virtual [XMLNode](#) * [ShallowClone](#) ([XMLDocument](#) *) const override
- virtual bool [ShallowEqual](#) (const [XMLNode](#) *) const override

Public Member Functions inherited from [tinyxml2::XMLNode](#)

- const [XMLDocument](#) * [GetDocument](#) () const
Get the [XMLDocument](#) that owns this [XMLNode](#).
- [XMLDocument](#) * [GetDocument](#) ()
Get the [XMLDocument](#) that owns this [XMLNode](#).
- virtual [XMLElement](#) * [ToElement](#) ()
Safely cast to an [Element](#), or null.
- virtual [XMLText](#) * [ToText](#) ()
Safely cast to [Text](#), or null.
- virtual [XMLComment](#) * [ToComment](#) ()
Safely cast to a [Comment](#), or null.
- virtual [XMLDeclaration](#) * [ToDeclaration](#) ()
Safely cast to a [Declaration](#), or null.
- virtual [XMLUnknown](#) * [ToUnknown](#) ()
Safely cast to an [Unknown](#), or null.
- virtual const [XMLElement](#) * [ToElement](#) () const
- virtual const [XMLText](#) * [ToText](#) () const
- virtual const [XMLComment](#) * [ToComment](#) () const
- virtual const [XMLDeclaration](#) * [ToDeclaration](#) () const
- virtual const [XMLUnknown](#) * [ToUnknown](#) () const
- int [ChildElementCount](#) (const char *value) const
- int [ChildElementCount](#) () const
- const char * [Value](#) () const
- void [SetValue](#) (const char *val, bool staticMem=false)
- int [GetLineNum](#) () const
Gets the line number the node is in, if the document was parsed from a file.
- const [XMLNode](#) * [Parent](#) () const
Get the parent of this node on the DOM.
- [XMLNode](#) * [Parent](#) ()
- bool [NoChildren](#) () const
Returns true if this node has no children.
- const [XMLNode](#) * [FirstChild](#) () const
Get the first child node, or null if none exists.
- [XMLNode](#) * [FirstChild](#) ()

- const [XMLElement](#) * [FirstChildElement](#) (const char *name=0) const
- [XMLElement](#) * [FirstChildElement](#) (const char *name=0)
- const [XMLNode](#) * [LastChild](#) () const
Get the last child node, or null if none exists.
- [XMLNode](#) * [LastChild](#) ()
- const [XMLElement](#) * [LastChildElement](#) (const char *name=0) const
- [XMLElement](#) * [LastChildElement](#) (const char *name=0)
- const [XMLNode](#) * [PreviousSibling](#) () const
Get the previous (left) sibling node of this node.
- [XMLNode](#) * [PreviousSibling](#) ()
- const [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0) const
Get the previous (left) sibling element of this node, with an optionally supplied name.
- [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0)
- const [XMLNode](#) * [NextSibling](#) () const
Get the next (right) sibling node of this node.
- [XMLNode](#) * [NextSibling](#) ()
- const [XMLElement](#) * [NextSiblingElement](#) (const char *name=0) const
Get the next (right) sibling element of this node, with an optionally supplied name.
- [XMLElement](#) * [NextSiblingElement](#) (const char *name=0)
- [XMLNode](#) * [InsertEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [LinkEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertFirstChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertAfterChild](#) ([XMLNode](#) *afterThis, [XMLNode](#) *addThis)
- void [DeleteChildren](#) ()
- void [DeleteChild](#) ([XMLNode](#) *node)
- [XMLNode](#) * [DeepClone](#) ([XMLDocument](#) *target) const
- void [SetUserData](#) (void *userData)
- void * [GetUserData](#) () const

Static Public Member Functions

- static const char * [ErrorIDToName](#) ([XMLError](#) errorID)

Friends

- class [XMLElement](#)
- class [XMLNode](#)
- class [XMLText](#)
- class [XMLComment](#)
- class [XMLDeclaration](#)
- class [XMLUnknown](#)

Additional Inherited Members

Protected Member Functions inherited from [tinyxml2::XMLNode](#)

- [XMLNode](#) ([XMLDocument](#) *)
- virtual [~XMLNode](#) ()
- virtual char * [ParseDeep](#) (char *p, [StrPair](#) *parentEndTag, int *curLineNumPtr)

Protected Attributes inherited from [tinyxml2::XMLNode](#)

- [XMLDocument](#) * [_document](#)
- [XMLNode](#) * [_parent](#)
- [StrPair](#) [_value](#)
- int [_parseLineNum](#)
- [XMLNode](#) * [_firstChild](#)
- [XMLNode](#) * [_lastChild](#)
- [XMLNode](#) * [_prev](#)
- [XMLNode](#) * [_next](#)
- void * [_userData](#)

8.21.1 Detailed Description

A Document binds together all the functionality. It can be saved, loaded, and printed to the screen. All Nodes are connected and allocated to a Document. If the Document is deleted, all its Nodes are also deleted.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 XMLDocument()

```
tinyxml2::XMLDocument::XMLDocument (
    bool processEntities = true,
    Whitespace whitespaceMode = PRESERVE_WHITESPACE)
```

constructor

8.21.2.2 ~XMLDocument()

```
tinyxml2::XMLDocument::~~XMLDocument ()
```

8.21.3 Member Function Documentation

8.21.3.1 Accept()

```
bool tinyxml2::XMLDocument::Accept (
    XMLVisitor * visitor) const [override], [virtual]
```

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the [XMLVisitor](#) interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinyxml2::XMLNode](#).

8.21.3.2 Clear()

```
void tinyxml2::XMLDocument::Clear ()
```

Clear the document, resetting it to the initial state.

8.21.3.3 ClearError()

```
void tinyxml2::XMLDocument::ClearError ()
```

Clears the error flags.

8.21.3.4 DeepCopy()

```
void tinyxml2::XMLDocument::DeepCopy (
    XMLDocument * target) const
```

Copies this document to a target document. The target will be completely cleared before the copy. If you want to copy a sub-tree, see [XMLNode::DeepClone\(\)](#).

NOTE: that the 'target' must be non-null.

8.21.3.5 DeleteNode()

```
void tinyxml2::XMLDocument::DeleteNode (
    XMLNode * node)
```

Delete a node associated with this document. It will be unlinked from the DOM.

8.21.3.6 Error()

```
bool tinyxml2::XMLDocument::Error () const [inline]
```

Return true if there was an error parsing the document.

8.21.3.7 ErrorID()

```
XMLError tinyxml2::XMLDocument::ErrorID () const [inline]
```

Return the errorID.

8.21.3.8 ErrorIDToName()

```
const char * tinyxml2::XMLDocument::ErrorIDToName (
    XMLError errorID) [static]
```

8.21.3.9 ErrorLineNum()

```
int tinyxml2::XMLDocument::ErrorLineNum () const [inline]
```

Return the line where the error occurred, or zero if unknown.

8.21.3.10 ErrorName()

```
const char * tinyxml2::XMLDocument::ErrorName () const
```

8.21.3.11 ErrorStr()

```
const char * tinyxml2::XMLDocument::ErrorStr () const
```

Returns a "long form" error description. A hopefully helpful diagnostic with location, line number, and/or additional info.

8.21.3.12 HasBOM()

```
bool tinyxml2::XMLDocument::HasBOM () const [inline]
```

Returns true if this document has a leading Byte Order Mark of UTF8.

8.21.3.13 Identify()

```
char * tinyxml2::XMLDocument::Identify (  
    char * p,  
    XMLNode ** node,  
    bool first)
```

8.21.3.14 LoadFile() [1/2]

```
XMLError tinyxml2::XMLDocument::LoadFile (  
    const char * filename)
```

Load an XML file from disk. Returns XML_SUCCESS (0) on success, or an errorID.

8.21.3.15 LoadFile() [2/2]

```
XMLError tinyxml2::XMLDocument::LoadFile (  
    FILE * fp)
```

Load an XML file from disk. You are responsible for providing and closing the FILE*.

NOTE: The file should be opened as binary ("rb") not text in order for TinyXML-2 to correctly do newline normalization.

Returns XML_SUCCESS (0) on success, or an errorID.

8.21.3.16 MarkInUse()

```
void tinyxml2::XMLDocument::MarkInUse (
    const XMLNode * const node)
```

8.21.3.17 NewComment()

```
XMLComment * tinyxml2::XMLDocument::NewComment (
    const char * comment)
```

Create a new Comment associated with this Document. The memory for the Comment is managed by the Document.

8.21.3.18 NewDeclaration()

```
XMLDeclaration * tinyxml2::XMLDocument::NewDeclaration (
    const char * text = 0)
```

Create a new Declaration associated with this Document. The memory for the object is managed by the Document.

If the 'text' param is null, the standard declaration is used.:

```
<?xml version="1.0" encoding="UTF-8"?>
```

8.21.3.19 NewElement()

```
XMLElement * tinyxml2::XMLDocument::NewElement (
    const char * name)
```

Create a new Element associated with this Document. The memory for the Element is managed by the Document.

8.21.3.20 NewText()

```
XMLText * tinyxml2::XMLDocument::NewText (
    const char * text)
```

Create a new Text associated with this Document. The memory for the Text is managed by the Document.

8.21.3.21 NewUnknown()

```
XMLUnknown * tinyxml2::XMLDocument::NewUnknown (
    const char * text)
```

Create a new Unknown associated with this Document. The memory for the object is managed by the Document.

8.21.3.22 Parse()

```
XMLError tinyxml2::XMLDocument::Parse (
    const char * xml,
    size_t nBytes = static_cast<size_t>(-1))
```

Parse an XML file from a character string. Returns XML_SUCCESS (0) on success, or an errorID.

You may optionally pass in the 'nBytes', which is the number of bytes which will be parsed. If not specified, TinyXML-2 will assume 'xml' points to a null terminated string.

8.21.3.23 Print()

```
void tinyxml2::XMLDocument::Print (
    XMLPrinter * streamer = 0) const
```

Print the Document. If the Printer is not provided, it will print to stdout. If you provide Printer, this can print to a file:

```
XMLPrinter printer( fp );
doc.Print( &printer );
```

Or you can use a printer to print to memory:

```
XMLPrinter printer;
doc.Print( &printer );
// printer.CStr() has a const char* to the XML
```

8.21.3.24 PrintError()

```
void tinyxml2::XMLDocument::PrintError () const
```

A (trivial) utility function that prints the [ErrorStr\(\)](#) to stdout.

8.21.3.25 ProcessEntities()

```
bool tinyxml2::XMLDocument::ProcessEntities () const [inline]
```

8.21.3.26 RootElement() [1/2]

```
XMLElement * tinyxml2::XMLDocument::RootElement () [inline]
```

Return the root element of DOM. Equivalent to [FirstChildElement\(\)](#). To get the first node, use [FirstChild\(\)](#).

8.21.3.27 RootElement() [2/2]

```
const XMLElement * tinyxml2::XMLDocument::RootElement () const [inline]
```

8.21.3.28 SaveFile() [1/2]

```
XML_Error tinyxml2::XMLDocument::SaveFile (
    const char * filename,
    bool compact = false)
```

Save the XML file to disk. Returns XML_SUCCESS (0) on success, or an errorID.

8.21.3.29 SaveFile() [2/2]

```
XML_Error tinyxml2::XMLDocument::SaveFile (
    FILE * fp,
    bool compact = false)
```

Save the XML file to disk. You are responsible for providing and closing the FILE*.

Returns XML_SUCCESS (0) on success, or an errorID.

8.21.3.30 SetBOM()

```
void tinyxml2::XMLDocument::SetBOM (
    bool useBOM) [inline]
```

Sets whether to write the BOM when writing the file.

8.21.3.31 ShallowClone()

```
virtual XMLNode * tinyxml2::XMLDocument::ShallowClone (
    XMLDocument * document) const [inline], [override], [virtual]
```

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (this->GetDocument())

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

8.21.3.32 ShallowEqual()

```
virtual bool tinyxml2::XMLDocument::ShallowEqual (
    const XMLNode * compare) const [inline], [override], [virtual]
```

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

8.21.3.33 ToDocument() [1/2]

```
virtual const XMLDocument * tinyxml2::XMLDocument::ToDocument () const [inline], [override],  
[virtual]
```

Reimplemented from [tinyxml2::XMLNode](#).

8.21.3.34 ToDocument() [2/2]

```
virtual XMLDocument * tinyxml2::XMLDocument::ToDocument () [inline], [override], [virtual]
```

Safely cast to a Document, or null.

Reimplemented from [tinyxml2::XMLNode](#).

8.21.3.35 WhitespaceMode()

```
Whitespace tinyxml2::XMLDocument::WhitespaceMode () const [inline]
```

8.21.4 Friends And Related Symbol Documentation

8.21.4.1 XMLComment

```
friend class XMLComment [friend]
```

8.21.4.2 XMLDeclaration

```
friend class XMLDeclaration [friend]
```

8.21.4.3 XMLElement

```
friend class XMLElement [friend]
```

8.21.4.4 XMLNode

```
friend class XMLNode [friend]
```

8.21.4.5 XMLText

```
friend class XMLText [friend]
```

8.21.4.6 XMLUnknown

```
friend class XMLUnknown [friend]
```

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

- [app/include/tinyxml2.h](#)
- [app/src/tinyxml2.cpp](#)

8.22 tinyxml2::XMLElement Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLElement:

Collaboration diagram for tinyxml2::XMLElement:

Public Types

- enum [ElementClosingType](#) { [OPEN](#) , [CLOSED](#) , [CLOSING](#) }

Public Member Functions

- const char * [Name](#) () const
Get the name of an element (which is the [Value\(\)](#) of the node.).
- void [SetName](#) (const char *str, bool staticMem=false)
Set the name of the element.
- virtual XMLElement * [ToElement](#) () override
Safely cast to an [Element](#), or null.
- virtual const XMLElement * [ToElement](#) () const override
- virtual bool [Accept](#) (XMLVisitor *visitor) const override
- const char * [Attribute](#) (const char *name, const char *value=0) const
- int [IntAttribute](#) (const char *name, int defaultValue=0) const
- unsigned [UnsignedAttribute](#) (const char *name, unsigned defaultValue=0) const
See [IntAttribute\(\)](#).
- int64_t [Int64Attribute](#) (const char *name, int64_t defaultValue=0) const
See [IntAttribute\(\)](#).
- uint64_t [Unsigned64Attribute](#) (const char *name, uint64_t defaultValue=0) const
See [IntAttribute\(\)](#).
- bool [BoolAttribute](#) (const char *name, bool defaultValue=false) const
See [IntAttribute\(\)](#).
- double [DoubleAttribute](#) (const char *name, double defaultValue=0) const
See [IntAttribute\(\)](#).
- float [FloatAttribute](#) (const char *name, float defaultValue=0) const
See [IntAttribute\(\)](#).
- [XMLError](#) [QueryIntAttribute](#) (const char *name, int *value) const
- [XMLError](#) [QueryUnsignedAttribute](#) (const char *name, unsigned int *value) const
See [QueryIntAttribute\(\)](#).
- [XMLError](#) [QueryInt64Attribute](#) (const char *name, int64_t *value) const

- See [QueryIntAttribute\(\)](#).
- [XML_Error_QueryUnsigned64Attribute](#) (const char *name, uint64_t *value) const
 - See [QueryIntAttribute\(\)](#).
- [XML_Error_QueryBoolAttribute](#) (const char *name, bool *value) const
 - See [QueryIntAttribute\(\)](#).
- [XML_Error_QueryDoubleAttribute](#) (const char *name, double *value) const
 - See [QueryIntAttribute\(\)](#).
- [XML_Error_QueryFloatAttribute](#) (const char *name, float *value) const
 - See [QueryIntAttribute\(\)](#).
- [XML_Error_QueryStringAttribute](#) (const char *name, const char **value) const
 - See [QueryIntAttribute\(\)](#).
- [XML_Error_QueryAttribute](#) (const char *name, int *value) const
- [XML_Error_QueryAttribute](#) (const char *name, unsigned int *value) const
- [XML_Error_QueryAttribute](#) (const char *name, int64_t *value) const
- [XML_Error_QueryAttribute](#) (const char *name, uint64_t *value) const
- [XML_Error_QueryAttribute](#) (const char *name, bool *value) const
- [XML_Error_QueryAttribute](#) (const char *name, double *value) const
- [XML_Error_QueryAttribute](#) (const char *name, float *value) const
- [XML_Error_QueryAttribute](#) (const char *name, const char **value) const
- void [SetAttribute](#) (const char *name, const char *value)
 - Sets the named attribute to value.
- void [SetAttribute](#) (const char *name, int value)
 - Sets the named attribute to value.
- void [SetAttribute](#) (const char *name, unsigned value)
 - Sets the named attribute to value.
- void [SetAttribute](#) (const char *name, int64_t value)
 - Sets the named attribute to value.
- void [SetAttribute](#) (const char *name, uint64_t value)
 - Sets the named attribute to value.
- void [SetAttribute](#) (const char *name, bool value)
 - Sets the named attribute to value.
- void [SetAttribute](#) (const char *name, double value)
 - Sets the named attribute to value.
- void [SetAttribute](#) (const char *name, float value)
 - Sets the named attribute to value.
- void [DeleteAttribute](#) (const char *name)
- const [XMLAttribute](#) * [FirstAttribute](#) () const
 - Return the first attribute in the list.
- const [XMLAttribute](#) * [FindAttribute](#) (const char *name) const
 - Query a specific attribute in the list.
- const char * [GetText](#) () const
- void [SetText](#) (const char *inText)
- void [SetText](#) (int value)
 - Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.
- void [SetText](#) (unsigned value)
 - Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.
- void [SetText](#) (int64_t value)
 - Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.
- void [SetText](#) (uint64_t value)
 - Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.
- void [SetText](#) (bool value)

- Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.*

 - void [SetText](#) (double value)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

 - void [SetText](#) (float value)

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

 - [XMLError QueryIntText](#) (int *ival) const
 - [XMLError QueryUnsignedText](#) (unsigned *uval) const

See [QueryIntText\(\)](#).

 - [XMLError QueryInt64Text](#) (int64_t *uval) const

See [QueryIntText\(\)](#).

 - [XMLError QueryUnsigned64Text](#) (uint64_t *uval) const

See [QueryIntText\(\)](#).

 - [XMLError QueryBoolText](#) (bool *bval) const

See [QueryIntText\(\)](#).

 - [XMLError QueryDoubleText](#) (double *dval) const

See [QueryIntText\(\)](#).

 - [XMLError QueryFloatText](#) (float *fval) const

See [QueryIntText\(\)](#).

 - int [IntText](#) (int defaultValue=0) const
 - unsigned [UnsignedText](#) (unsigned defaultValue=0) const

See [QueryIntText\(\)](#).

 - int64_t [Int64Text](#) (int64_t defaultValue=0) const

See [QueryIntText\(\)](#).

 - uint64_t [Unsigned64Text](#) (uint64_t defaultValue=0) const

See [QueryIntText\(\)](#).

 - bool [BoolText](#) (bool defaultValue=false) const

See [QueryIntText\(\)](#).

 - double [DoubleText](#) (double defaultValue=0) const

See [QueryIntText\(\)](#).

 - float [FloatText](#) (float defaultValue=0) const

See [QueryIntText\(\)](#).

 - [XMLElement](#) * [InsertNewChildElement](#) (const char *name)
 - [XMLComment](#) * [InsertNewComment](#) (const char *comment)

See [InsertNewChildElement\(\)](#).

 - [XMLText](#) * [InsertNewText](#) (const char *text)

See [InsertNewChildElement\(\)](#).

 - [XMLDeclaration](#) * [InsertNewDeclaration](#) (const char *text)

See [InsertNewChildElement\(\)](#).

 - [XMLUnknown](#) * [InsertNewUnknown](#) (const char *text)

See [InsertNewChildElement\(\)](#).

 - [ElementClosingType](#) [ClosingType](#) () const
 - virtual [XMLNode](#) * [ShallowClone](#) ([XMLDocument](#) *document) const override
 - virtual bool [ShallowEqual](#) (const [XMLNode](#) *compare) const override

Public Member Functions inherited from [tinyxml2::XMLNode](#)

- const [XMLDocument](#) * [GetDocument](#) () const
Get the [XMLDocument](#) that owns this [XMLNode](#).
- [XMLDocument](#) * [GetDocument](#) ()
Get the [XMLDocument](#) that owns this [XMLNode](#).
- virtual [XMLText](#) * [ToText](#) ()
Safely cast to [Text](#), or null.
- virtual [XMLComment](#) * [ToComment](#) ()
Safely cast to a [Comment](#), or null.
- virtual [XMLDocument](#) * [ToDocument](#) ()
Safely cast to a [Document](#), or null.
- virtual [XMLDeclaration](#) * [ToDeclaration](#) ()
Safely cast to a [Declaration](#), or null.
- virtual [XMLUnknown](#) * [ToUnknown](#) ()
Safely cast to an [Unknown](#), or null.
- virtual const [XMLText](#) * [ToText](#) () const
- virtual const [XMLComment](#) * [ToComment](#) () const
- virtual const [XMLDocument](#) * [ToDocument](#) () const
- virtual const [XMLDeclaration](#) * [ToDeclaration](#) () const
- virtual const [XMLUnknown](#) * [ToUnknown](#) () const
- int [ChildElementCount](#) (const char *value) const
- int [ChildElementCount](#) () const
- const char * [Value](#) () const
- void [SetValue](#) (const char *val, bool staticMem=false)
- int [GetLineNum](#) () const
Gets the line number the node is in, if the document was parsed from a file.
- const [XMLNode](#) * [Parent](#) () const
Get the parent of this node on the DOM.
- [XMLNode](#) * [Parent](#) ()
- bool [NoChildren](#) () const
Returns true if this node has no children.
- const [XMLNode](#) * [FirstChild](#) () const
Get the first child node, or null if none exists.
- [XMLNode](#) * [FirstChild](#) ()
- const [XMLElement](#) * [FirstChildElement](#) (const char *name=0) const
- [XMLElement](#) * [FirstChildElement](#) (const char *name=0)
- const [XMLNode](#) * [LastChild](#) () const
Get the last child node, or null if none exists.
- [XMLNode](#) * [LastChild](#) ()
- const [XMLElement](#) * [LastChildElement](#) (const char *name=0) const
- [XMLElement](#) * [LastChildElement](#) (const char *name=0)
- const [XMLNode](#) * [PreviousSibling](#) () const
Get the previous (left) sibling node of this node.
- [XMLNode](#) * [PreviousSibling](#) ()
- const [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0) const
Get the previous (left) sibling element of this node, with an optionally supplied name.
- [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0)
- const [XMLNode](#) * [NextSibling](#) () const
Get the next (right) sibling node of this node.
- [XMLNode](#) * [NextSibling](#) ()
- const [XMLElement](#) * [NextSiblingElement](#) (const char *name=0) const

Get the next (right) sibling element of this node, with an optionally supplied name.

- [XMLElement](#) * [NextSiblingElement](#) (const char *name=0)
- [XMLNode](#) * [InsertEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [LinkEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertFirstChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertAfterChild](#) ([XMLNode](#) *afterThis, [XMLNode](#) *addThis)
- void [DeleteChildren](#) ()
- void [DeleteChild](#) ([XMLNode](#) *node)
- [XMLNode](#) * [DeepClone](#) ([XMLDocument](#) *target) const
- void [SetUserData](#) (void *userData)
- void * [GetUserData](#) () const

Protected Member Functions

- char * [ParseDeep](#) (char *p, [StrPair](#) *parentEndTag, int *curLineNumPtr) override

Protected Member Functions inherited from [tinyxml2::XMLNode](#)

- [XMLNode](#) ([XMLDocument](#) *)
- virtual [~XMLNode](#) ()

Friends

- class [XMLDocument](#)

Additional Inherited Members

Protected Attributes inherited from [tinyxml2::XMLNode](#)

- [XMLDocument](#) * [_document](#)
- [XMLNode](#) * [_parent](#)
- [StrPair](#) [_value](#)
- int [_parseLineNum](#)
- [XMLNode](#) * [_firstChild](#)
- [XMLNode](#) * [_lastChild](#)
- [XMLNode](#) * [_prev](#)
- [XMLNode](#) * [_next](#)
- void * [_userData](#)

8.22.1 Detailed Description

The element is a container class. It has a value, the element name, and can contain other elements, text, comments, and unknowns. Elements also contain an arbitrary number of attributes.

8.22.2 Member Enumeration Documentation

8.22.2.1 ElementClosingType

```
enum tinyxml2::XMLElement::ElementClosingType
```

Enumerator

OPEN	
CLOSED	
CLOSING	

8.22.3 Member Function Documentation

8.22.3.1 Accept()

```
bool tinyxml2::XMLElement::Accept (
    XMLVisitor * visitor) const [override], [virtual]
```

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the [XMLVisitor](#) interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinyxml2::XMLNode](#).

8.22.3.2 Attribute()

```
const char * tinyxml2::XMLElement::Attribute (
    const char * name,
    const char * value = 0) const
```

Given an attribute name, [Attribute\(\)](#) returns the value for the attribute of that name, or null if none exists. For example:

```
const char* value = ele->Attribute( "foo" );
```

The 'value' parameter is normally null. However, if specified, the attribute will only be returned if the 'name' and 'value' match. This allow you to write code:

```
if ( ele->Attribute( "foo", "bar" ) ) callFooIsBar();
```

rather than:

```
if ( ele->Attribute( "foo" ) ) {
    if ( strcmp( ele->Attribute( "foo" ), "bar" ) == 0 ) callFooIsBar();
}
```

8.22.3.3 BoolAttribute()

```
bool tinyxml2::XMLElement::BoolAttribute (
    const char * name,
    bool defaultValue = false) const
```

See [IntAttribute\(\)](#).

8.22.3.4 BoolText()

```
bool tinyxml2::XMLElement::BoolText (
    bool defaultValue = false) const
```

See [QueryIntText\(\)](#).

8.22.3.5 ClosingType()

```
ElementClosingType tinyxml2::XMLElement::ClosingType () const [inline]
```

8.22.3.6 DeleteAttribute()

```
void tinyxml2::XMLElement::DeleteAttribute (
    const char * name)
```

Delete an attribute.

8.22.3.7 DoubleAttribute()

```
double tinyxml2::XMLElement::DoubleAttribute (
    const char * name,
    double defaultValue = 0) const
```

See [IntAttribute\(\)](#).

8.22.3.8 DoubleText()

```
double tinyxml2::XMLElement::DoubleText (
    double defaultValue = 0) const
```

See [QueryIntText\(\)](#).

8.22.3.9 FindAttribute()

```
const XMLAttribute * tinyxml2::XMLElement::FindAttribute (
    const char * name) const
```

Query a specific attribute in the list.

8.22.3.10 FirstAttribute()

```
const XMLAttribute * tinyxml2::XMLElement::FirstAttribute () const [inline]
```

Return the first attribute in the list.

8.22.3.11 FloatAttribute()

```
float tinyxml2::XMLElement::FloatAttribute (  
    const char * name,  
    float defaultValue = 0) const
```

See [IntAttribute\(\)](#).

8.22.3.12 FloatText()

```
float tinyxml2::XMLElement::FloatText (  
    float defaultValue = 0) const
```

See [QueryIntText\(\)](#).

8.22.3.13 GetText()

```
const char * tinyxml2::XMLElement::GetText () const
```

Convenience function for easy access to the text inside an element. Although easy and concise, [GetText\(\)](#) is limited compared to getting the [XMLText](#) child and accessing it directly.

If the first child of 'this' is a [XMLText](#), the [GetText\(\)](#) returns the character string of the Text node, else null is returned.

This is a convenient method for getting the text of simple contained text:

```
<foo>This is text</foo>  
const char* str = fooElement->GetText();
```

'str' will be a pointer to "This is text".

Note that this function can be misleading. If the element foo was created from this XML:

```
<foo><b>This is text</b></foo>
```

then the value of str would be null. The first child node isn't a text node, it is another element. From this XML:

```
<foo>This is <b>text</b></foo>
```

[GetText\(\)](#) will return "This is ".

8.22.3.14 InsertNewChildElement()

```
XMLElement * tinyxml2::XMLElement::InsertNewChildElement (
    const char * name)
```

Convenience method to create a new [XMLElement](#) and add it as last (right) child of this node. Returns the created and inserted element.

8.22.3.15 InsertNewComment()

```
XMLComment * tinyxml2::XMLElement::InsertNewComment (
    const char * comment)
```

See [InsertNewChildElement\(\)](#).

8.22.3.16 InsertNewDeclaration()

```
XMLDeclaration * tinyxml2::XMLElement::InsertNewDeclaration (
    const char * text)
```

See [InsertNewChildElement\(\)](#).

8.22.3.17 InsertNewText()

```
XMLText * tinyxml2::XMLElement::InsertNewText (
    const char * text)
```

See [InsertNewChildElement\(\)](#).

8.22.3.18 InsertNewUnknown()

```
XMLUnknown * tinyxml2::XMLElement::InsertNewUnknown (
    const char * text)
```

See [InsertNewChildElement\(\)](#).

8.22.3.19 Int64Attribute()

```
int64_t tinyxml2::XMLElement::Int64Attribute (
    const char * name,
    int64_t defaultValue = 0) const
```

See [IntAttribute\(\)](#).

8.22.3.20 Int64Text()

```
int64_t tinyxml2::XMLElement::Int64Text (
    int64_t defaultValue = 0) const
```

See [QueryIntText\(\)](#).

8.22.3.21 IntAttribute()

```
int tinyxml2::XMLElement::IntAttribute (
    const char * name,
    int defaultValue = 0) const
```

Given an attribute name, [IntAttribute\(\)](#) returns the value of the attribute interpreted as an integer. The default value will be returned if the attribute isn't present, or if there is an error. (For a method with error checking, see [QueryIntAttribute\(\)](#)).

8.22.3.22 IntText()

```
int tinyxml2::XMLElement::IntText (
    int defaultValue = 0) const
```

8.22.3.23 Name()

```
const char * tinyxml2::XMLElement::Name () const [inline]
```

Get the name of an element (which is the [Value\(\)](#) of the node.).

8.22.3.24 ParseDeep()

```
char * tinyxml2::XMLElement::ParseDeep (
    char * p,
    StrPair * parentEndTag,
    int * curLineNumPtr) [override], [protected], [virtual]
```

Reimplemented from [tinyxml2::XMLNode](#).

8.22.3.25 QueryAttribute() [1/8]

```
XML_Error tinyxml2::XMLElement::QueryAttribute (
    const char * name,
    bool * value) const [inline]
```

8.22.3.26 QueryAttribute() [2/8]

```
XML_Error tinyxml2::XMLElement::QueryAttribute (
    const char * name,
    const char ** value) const [inline]
```

8.22.3.27 QueryAttribute() [3/8]

```
XML_Error tinyxml2::XMLElement::QueryAttribute (
    const char * name,
    double * value) const [inline]
```

8.22.3.28 QueryAttribute() [4/8]

```
XML_Error_tinyxml2::XMLElement::QueryAttribute (
    const char * name,
    float * value) const [inline]
```

8.22.3.29 QueryAttribute() [5/8]

```
XML_Error_tinyxml2::XMLElement::QueryAttribute (
    const char * name,
    int * value) const [inline]
```

Given an attribute name, [QueryAttribute\(\)](#) returns XML_SUCCESS, XML_WRONG_ATTRIBUTE_TYPE if the conversion can't be performed, or XML_NO_ATTRIBUTE if the attribute doesn't exist. It is overloaded for the primitive types, and is a generally more convenient replacement of [QueryIntAttribute\(\)](#) and related functions.

If successful, the result of the conversion will be written to 'value'. If not successful, nothing will be written to 'value'. This allows you to provide default value:

```
int value = 10;
QueryAttribute( "foo", &value );           // if "foo" isn't found, value will still be 10
```

8.22.3.30 QueryAttribute() [6/8]

```
XML_Error_tinyxml2::XMLElement::QueryAttribute (
    const char * name,
    int64_t * value) const [inline]
```

8.22.3.31 QueryAttribute() [7/8]

```
XML_Error_tinyxml2::XMLElement::QueryAttribute (
    const char * name,
    uint64_t * value) const [inline]
```

8.22.3.32 QueryAttribute() [8/8]

```
XML_Error_tinyxml2::XMLElement::QueryAttribute (
    const char * name,
    unsigned int * value) const [inline]
```

8.22.3.33 QueryBoolAttribute()

```
XML_Error_tinyxml2::XMLElement::QueryBoolAttribute (
    const char * name,
    bool * value) const [inline]
```

See [QueryIntAttribute\(\)](#).

8.22.3.34 QueryBoolText()

```
XML_Error tinyxml2::XMLElement::QueryBoolText (
    bool * bval) const
```

See [QueryIntText\(\)](#).

8.22.3.35 QueryDoubleAttribute()

```
XML_Error tinyxml2::XMLElement::QueryDoubleAttribute (
    const char * name,
    double * value) const [inline]
```

See [QueryIntAttribute\(\)](#).

8.22.3.36 QueryDoubleText()

```
XML_Error tinyxml2::XMLElement::QueryDoubleText (
    double * dval) const
```

See [QueryIntText\(\)](#).

8.22.3.37 QueryFloatAttribute()

```
XML_Error tinyxml2::XMLElement::QueryFloatAttribute (
    const char * name,
    float * value) const [inline]
```

See [QueryIntAttribute\(\)](#).

8.22.3.38 QueryFloatText()

```
XML_Error tinyxml2::XMLElement::QueryFloatText (
    float * fval) const
```

See [QueryIntText\(\)](#).

8.22.3.39 QueryInt64Attribute()

```
XML_Error tinyxml2::XMLElement::QueryInt64Attribute (
    const char * name,
    int64_t * value) const [inline]
```

See [QueryIntAttribute\(\)](#).

8.22.3.40 QueryInt64Text()

```
XML_Error tinyxml2::XMLElement::QueryInt64Text (
    int64_t * uval) const
```

See [QueryIntText\(\)](#).

8.22.3.41 QueryIntAttribute()

```
XML_Error tinyxml2::XMLElement::QueryIntAttribute (
    const char * name,
    int * value) const [inline]
```

Given an attribute name, [QueryIntAttribute\(\)](#) returns XML_SUCCESS, XML_WRONG_ATTRIBUTE_TYPE if the conversion can't be performed, or XML_NO_ATTRIBUTE if the attribute doesn't exist. If successful, the result of the conversion will be written to 'value'. If not successful, nothing will be written to 'value'. This allows you to provide default value:

```
int value = 10;
QueryIntAttribute( "foo", &value );    // if "foo" isn't found, value will still be 10
```

8.22.3.42 QueryIntText()

```
XML_Error tinyxml2::XMLElement::QueryIntText (
    int * ival) const
```

Convenience method to query the value of a child text node. This is probably best shown by example. Given you have a document is this form:

```
<point>
  <x>1</x>
  <y>1.4</y>
</point>
```

The [QueryIntText\(\)](#) and similar functions provide a safe and easier way to get to the "value" of x and y.

```
int x = 0;
float y = 0;    // types of x and y are contrived for example
const XMLElement* xElement = pointElement->FirstChildElement( "x" );
const XMLElement* yElement = pointElement->FirstChildElement( "y" );
xElement->QueryIntText( &x );
yElement->QueryFloatText( &y );
```

Returns

XML_SUCCESS (0) on success, XML_CAN_NOT_CONVERT_TEXT if the text cannot be converted to the requested type, and XML_NO_TEXT_NODE if there is no child text to query.

8.22.3.43 QueryStringAttribute()

```
XML_Error tinyxml2::XMLElement::QueryStringAttribute (
    const char * name,
    const char ** value) const [inline]
```

See [QueryIntAttribute\(\)](#).

8.22.3.44 QueryUnsigned64Attribute()

```
XML_Error tinyxml2::XMLElement::QueryUnsigned64Attribute (
    const char * name,
    uint64_t * value) const [inline]
```

See [QueryIntAttribute\(\)](#).

8.22.3.45 QueryUnsigned64Text()

```
XML_Error tinyxml2::XMLElement::QueryUnsigned64Text (
    uint64_t * uval) const
```

See [QueryIntText\(\)](#).

8.22.3.46 QueryUnsignedAttribute()

```
XML_Error tinyxml2::XMLElement::QueryUnsignedAttribute (
    const char * name,
    unsigned int * value) const [inline]
```

See [QueryIntAttribute\(\)](#).

8.22.3.47 QueryUnsignedText()

```
XML_Error tinyxml2::XMLElement::QueryUnsignedText (
    unsigned * uval) const
```

See [QueryIntText\(\)](#).

8.22.3.48 SetAttribute() [1/8]

```
void tinyxml2::XMLElement::SetAttribute (
    const char * name,
    bool value) [inline]
```

Sets the named attribute to value.

8.22.3.49 SetAttribute() [2/8]

```
void tinyxml2::XMLElement::SetAttribute (
    const char * name,
    const char * value) [inline]
```

Sets the named attribute to value.

8.22.3.50 SetAttribute() [3/8]

```
void tinyxml2::XMLElement::SetAttribute (
    const char * name,
    double value) [inline]
```

Sets the named attribute to value.

8.22.3.51 SetAttribute() [4/8]

```
void tinyxml2::XMLElement::SetAttribute (
    const char * name,
    float value) [inline]
```

Sets the named attribute to value.

8.22.3.52 SetAttribute() [5/8]

```
void tinyxml2::XMLElement::SetAttribute (
    const char * name,
    int value) [inline]
```

Sets the named attribute to value.

8.22.3.53 SetAttribute() [6/8]

```
void tinyxml2::XMLElement::SetAttribute (
    const char * name,
    int64_t value) [inline]
```

Sets the named attribute to value.

8.22.3.54 SetAttribute() [7/8]

```
void tinyxml2::XMLElement::SetAttribute (
    const char * name,
    uint64_t value) [inline]
```

Sets the named attribute to value.

8.22.3.55 SetAttribute() [8/8]

```
void tinyxml2::XMLElement::SetAttribute (
    const char * name,
    unsigned value) [inline]
```

Sets the named attribute to value.

8.22.3.56 SetName()

```
void tinyxml2::XMLElement::SetName (
    const char * str,
    bool staticMem = false) [inline]
```

Set the name of the element.

8.22.3.57 SetText() [1/8]

```
void tinyxml2::XMLElement::SetText (
    bool value)
```

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

8.22.3.58 SetText() [2/8]

```
void tinyxml2::XMLElement::SetText (
    const char * inText)
```

Convenience function for easy access to the text inside an element. Although easy and concise, [SetText\(\)](#) is limited compared to creating an [XMLText](#) child and mutating it directly.

If the first child of 'this' is a [XMLText](#), [SetText\(\)](#) sets its value to the given string, otherwise it will create a first child that is an [XMLText](#).

This is a convenient method for setting the text of simple contained text:

```
<foo>This is text</foo>
fooElement->SetText( "Hullaballoo!" );
<foo>Hullaballoo!</foo>
```

Note that this function can be misleading. If the element foo was created from this XML:

```
<foo><b>This is text</b></foo>
```

then it will not change "This is text", but rather prefix it with a text element:

```
<foo>Hullaballoo!<b>This is text</b></foo>
```

For this XML:

```
<foo />
```

[SetText\(\)](#) will generate

```
<foo>Hullaballoo!</foo>
```

8.22.3.59 SetText() [3/8]

```
void tinyxml2::XMLElement::SetText (  
    double value)
```

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

8.22.3.60 SetText() [4/8]

```
void tinyxml2::XMLElement::SetText (  
    float value)
```

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

8.22.3.61 SetText() [5/8]

```
void tinyxml2::XMLElement::SetText (  
    int value)
```

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

8.22.3.62 SetText() [6/8]

```
void tinyxml2::XMLElement::SetText (  
    int64_t value)
```

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

8.22.3.63 SetText() [7/8]

```
void tinyxml2::XMLElement::SetText (  
    uint64_t value)
```

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

8.22.3.64 SetText() [8/8]

```
void tinyxml2::XMLElement::SetText (  
    unsigned value)
```

Convenience method for setting text inside an element. See [SetText\(\)](#) for important limitations.

8.22.3.65 ShallowClone()

```
XMLNode * tinyxml2::XMLElement::ShallowClone (  
    XMLDocument * document) const [override], [virtual]
```

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (this->GetDocument())

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

8.22.3.66 ShallowEqual()

```
bool tinyxml2::XMLElement::ShallowEqual (  
    const XMLNode * compare) const [override], [virtual]
```

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

8.22.3.67 ToElement() [1/2]

```
virtual const XMLElement * tinyxml2::XMLElement::ToElement () const [inline], [override],  
[virtual]
```

Reimplemented from [tinyxml2::XMLNode](#).

8.22.3.68 ToElement() [2/2]

```
virtual XMLElement * tinyxml2::XMLElement::ToElement () [inline], [override], [virtual]
```

Safely cast to an [Element](#), or null.

Reimplemented from [tinyxml2::XMLNode](#).

8.22.3.69 Unsigned64Attribute()

```
uint64_t tinyxml2::XMLElement::Unsigned64Attribute (  
    const char * name,  
    uint64_t defaultValue = 0) const
```

See [IntAttribute\(\)](#).

8.22.3.70 Unsigned64Text()

```
uint64_t tinyxml2::XMLElement::Unsigned64Text (
    uint64_t defaultValue = 0) const
```

See [QueryIntText\(\)](#).

8.22.3.71 UnsignedAttribute()

```
unsigned tinyxml2::XMLElement::UnsignedAttribute (
    const char * name,
    unsigned defaultValue = 0) const
```

See [IntAttribute\(\)](#).

8.22.3.72 UnsignedText()

```
unsigned tinyxml2::XMLElement::UnsignedText (
    unsigned defaultValue = 0) const
```

See [QueryIntText\(\)](#).

8.22.4 Friends And Related Symbol Documentation

8.22.4.1 XMLDocument

```
friend class XMLDocument [friend]
```

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

- [app/include/tinyxml2.h](#)
- [app/src/tinyxml2.cpp](#)

8.23 tinyxml2::XMLHandle Class Reference

```
#include <tinyxml2.h>
```

Public Member Functions

- [XMLHandle](#) ([XMLNode](#) *node)
Create a handle from any node (at any depth of the tree.) This can be a null pointer.
- [XMLHandle](#) ([XMLNode](#) &node)
Create a handle from a node.
- [XMLHandle](#) (const [XMLHandle](#) &ref)
Copy constructor.
- [XMLHandle](#) & [operator=](#) (const [XMLHandle](#) &ref)
Assignment.
- [XMLHandle](#) [FirstChild](#) ()
Get the first child of this handle.
- [XMLHandle](#) [FirstChildElement](#) (const char *name=0)
Get the first child element of this handle.
- [XMLHandle](#) [LastChild](#) ()
Get the last child of this handle.
- [XMLHandle](#) [LastChildElement](#) (const char *name=0)
Get the last child element of this handle.
- [XMLHandle](#) [PreviousSibling](#) ()
Get the previous sibling of this handle.
- [XMLHandle](#) [PreviousSiblingElement](#) (const char *name=0)
Get the previous sibling element of this handle.
- [XMLHandle](#) [NextSibling](#) ()
Get the next sibling of this handle.
- [XMLHandle](#) [NextSiblingElement](#) (const char *name=0)
Get the next sibling element of this handle.
- [XMLNode](#) * [ToNode](#) ()
Safe cast to [XMLNode](#). This can return null.
- [XMLElement](#) * [ToElement](#) ()
Safe cast to [XMLElement](#). This can return null.
- [XMLText](#) * [ToText](#) ()
Safe cast to [XMLText](#). This can return null.
- [XMLUnknown](#) * [ToUnknown](#) ()
Safe cast to [XMLUnknown](#). This can return null.
- [XMLDeclaration](#) * [ToDeclaration](#) ()
Safe cast to [XMLDeclaration](#). This can return null.

8.23.1 Detailed Description

A [XMLHandle](#) is a class that wraps a node pointer with null checks; this is an incredibly useful thing. Note that [XMLHandle](#) is not part of the TinyXML-2 DOM structure. It is a separate utility class.

Take an example:

```
<Document>
  <Element attributeA = "valueA">
    <Child attributeB = "value1" />
    <Child attributeB = "value2" />
  </Element>
</Document>
```

Assuming you want the value of "attributeB" in the 2nd "Child" element, it's very easy to write a *lot* of code that looks like:

```
XMLElement* root = document.FirstChildElement( "Document" );
if ( root )
{
    XMLElement* element = root->FirstChildElement( "Element" );
    if ( element )
    {
        XMLElement* child = element->FirstChildElement( "Child" );
        if ( child )
        {
            XMLElement* child2 = child->NextSiblingElement( "Child" );
            if ( child2 )
            {
                // Finally do something useful.
            }
        }
    }
}
```

And that doesn't even cover "else" cases. [XMLHandle](#) addresses the verbosity of such code. A [XMLHandle](#) checks for null pointers so it is perfectly safe and correct to use:

```
XMLHandle docHandle( &document );
XMLElement* child2 = docHandle.FirstChildElement( "Document" ).FirstChildElement( "Element" ).FirstChildElement( "Child" );
if ( child2 )
{
    // do something useful
}
```

Which is MUCH more concise and useful.

It is also safe to copy handles - internally they are nothing more than node pointers.

```
XMLHandle handleCopy = handle;
```

See also [XMLConstHandle](#), which is the same as [XMLHandle](#), but operates on const objects.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 XMLHandle() [1/3]

```
tinycl2::XMLHandle::XMLHandle (
    XMLNode * node) [inline], [explicit]
```

Create a handle from any node (at any depth of the tree.) This can be a null pointer.

8.23.2.2 XMLHandle() [2/3]

```
tinycl2::XMLHandle::XMLHandle (
    XMLNode & node) [inline], [explicit]
```

Create a handle from a node.

8.23.2.3 XMLHandle() [3/3]

```
tinyxml2::XMLHandle::XMLHandle (  
    const XMLHandle & ref) [inline]
```

Copy constructor.

8.23.3 Member Function Documentation

8.23.3.1 FirstChild()

```
XMLHandle tinyxml2::XMLHandle::FirstChild () [inline]
```

Get the first child of this handle.

8.23.3.2 FirstChildElement()

```
XMLHandle tinyxml2::XMLHandle::FirstChildElement (  
    const char * name = 0) [inline]
```

Get the first child element of this handle.

8.23.3.3 LastChild()

```
XMLHandle tinyxml2::XMLHandle::LastChild () [inline]
```

Get the last child of this handle.

8.23.3.4 LastChildElement()

```
XMLHandle tinyxml2::XMLHandle::LastChildElement (  
    const char * name = 0) [inline]
```

Get the last child element of this handle.

8.23.3.5 NextSibling()

```
XMLHandle tinyxml2::XMLHandle::NextSibling () [inline]
```

Get the next sibling of this handle.

8.23.3.6 NextSiblingElement()

```
XMLHandle tinyxml2::XMLHandle::NextSiblingElement (  
    const char * name = 0) [inline]
```

Get the next sibling element of this handle.

8.23.3.7 operator=()

```
XMLHandle & tinyxml2::XMLHandle::operator= (  
    const XMLHandle & ref) [inline]
```

Assignment.

8.23.3.8 PreviousSibling()

```
XMLHandle tinyxml2::XMLHandle::PreviousSibling () [inline]
```

Get the previous sibling of this handle.

8.23.3.9 PreviousSiblingElement()

```
XMLHandle tinyxml2::XMLHandle::PreviousSiblingElement (  
    const char * name = 0) [inline]
```

Get the previous sibling element of this handle.

8.23.3.10 ToDeclaration()

```
XMLDeclaration * tinyxml2::XMLHandle::ToDeclaration () [inline]
```

Safe cast to [XMLDeclaration](#). This can return null.

8.23.3.11 ToElement()

```
XMLElement * tinyxml2::XMLHandle::ToElement () [inline]
```

Safe cast to [XMLElement](#). This can return null.

8.23.3.12 ToNode()

```
XMLNode * tinyxml2::XMLHandle::ToNode () [inline]
```

Safe cast to [XMLNode](#). This can return null.

8.23.3.13 ToText()

```
XMLText * tinyxml2::XMLHandle::ToText () [inline]
```

Safe cast to [XMLText](#). This can return null.

8.23.3.14 ToUnknown()

```
XMLUnknown * tinyxml2::XMLHandle::ToUnknown () [inline]
```

Safe cast to [XMLUnknown](#). This can return null.

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

- [app/include/tinyxml2.h](#)

8.24 tinyxml2::XMLNode Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLNode:

Collaboration diagram for tinyxml2::XMLNode:

Public Member Functions

- const [XMLDocument](#) * [GetDocument](#) () const
Get the [XMLDocument](#) that owns this [XMLNode](#).
- [XMLDocument](#) * [GetDocument](#) ()
Get the [XMLDocument](#) that owns this [XMLNode](#).
- virtual [XMLElement](#) * [ToElement](#) ()
Safely cast to an [Element](#), or null.
- virtual [XMLText](#) * [ToText](#) ()
Safely cast to [Text](#), or null.
- virtual [XMLComment](#) * [ToComment](#) ()
Safely cast to a [Comment](#), or null.
- virtual [XMLDocument](#) * [ToDocument](#) ()
Safely cast to a [Document](#), or null.
- virtual [XMLDeclaration](#) * [ToDeclaration](#) ()
Safely cast to a [Declaration](#), or null.
- virtual [XMLUnknown](#) * [ToUnknown](#) ()
Safely cast to an [Unknown](#), or null.
- virtual const [XMLElement](#) * [ToElement](#) () const
- virtual const [XMLText](#) * [ToText](#) () const
- virtual const [XMLComment](#) * [ToComment](#) () const
- virtual const [XMLDocument](#) * [ToDocument](#) () const
- virtual const [XMLDeclaration](#) * [ToDeclaration](#) () const
- virtual const [XMLUnknown](#) * [ToUnknown](#) () const
- int [ChildElementCount](#) (const char *value) const
- int [ChildElementCount](#) () const
- const char * [Value](#) () const
- void [SetValue](#) (const char *val, bool staticMem=false)
- int [GetLineNum](#) () const
Gets the line number the node is in, if the document was parsed from a file.
- const [XMLNode](#) * [Parent](#) () const
Get the parent of this node on the DOM.

- [XMLNode](#) * [Parent](#) ()
- bool [NoChildren](#) () const
Returns true if this node has no children.
- const [XMLNode](#) * [FirstChild](#) () const
Get the first child node, or null if none exists.
- [XMLNode](#) * [FirstChild](#) ()
- const [XMLElement](#) * [FirstChildElement](#) (const char *name=0) const
- [XMLElement](#) * [FirstChildElement](#) (const char *name=0)
- const [XMLNode](#) * [LastChild](#) () const
Get the last child node, or null if none exists.
- [XMLNode](#) * [LastChild](#) ()
- const [XMLElement](#) * [LastChildElement](#) (const char *name=0) const
- [XMLElement](#) * [LastChildElement](#) (const char *name=0)
- const [XMLNode](#) * [PreviousSibling](#) () const
Get the previous (left) sibling node of this node.
- [XMLNode](#) * [PreviousSibling](#) ()
- const [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0) const
Get the previous (left) sibling element of this node, with an optionally supplied name.
- [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0)
- const [XMLNode](#) * [NextSibling](#) () const
Get the next (right) sibling node of this node.
- [XMLNode](#) * [NextSibling](#) ()
- const [XMLElement](#) * [NextSiblingElement](#) (const char *name=0) const
Get the next (right) sibling element of this node, with an optionally supplied name.
- [XMLElement](#) * [NextSiblingElement](#) (const char *name=0)
- [XMLNode](#) * [InsertEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [LinkEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertFirstChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertAfterChild](#) ([XMLNode](#) *afterThis, [XMLNode](#) *addThis)
- void [DeleteChildren](#) ()
- void [DeleteChild](#) ([XMLNode](#) *node)
- virtual [XMLNode](#) * [ShallowClone](#) ([XMLDocument](#) *document) const =0
- [XMLNode](#) * [DeepClone](#) ([XMLDocument](#) *target) const
- virtual bool [ShallowEqual](#) (const [XMLNode](#) *compare) const =0
- virtual bool [Accept](#) ([XMLVisitor](#) *visitor) const =0
- void [SetUserData](#) (void *userData)
- void * [GetUserData](#) () const

Protected Member Functions

- [XMLNode](#) ([XMLDocument](#) *)
- virtual ~[XMLNode](#) ()
- virtual char * [ParseDeep](#) (char *p, [StrPair](#) *parentEndTag, int *curLineNumPtr)

Protected Attributes

- [XMLDocument](#) * [_document](#)
- [XMLNode](#) * [_parent](#)
- [StrPair](#) [_value](#)
- int [_parseLineNum](#)
- [XMLNode](#) * [_firstChild](#)
- [XMLNode](#) * [_lastChild](#)
- [XMLNode](#) * [_prev](#)
- [XMLNode](#) * [_next](#)
- void * [_userData](#)

Friends

- class [XMLDocument](#)
- class [XMLElement](#)

8.24.1 Detailed Description

[XMLNode](#) is a base class for every object that is in the XML Document Object Model (DOM), except [XMLAttributes](#). Nodes have siblings, a parent, and children which can be navigated. A node is always in a [XMLDocument](#). The type of a [XMLNode](#) can be queried, and it can be cast to its more defined type.

A [XMLDocument](#) allocates memory for all its Nodes. When the [XMLDocument](#) gets deleted, all its Nodes will also be deleted.

```
A Document can contain: Element (container or leaf)
                        Comment (leaf)
                        Unknown (leaf)
                        Declaration( leaf )
```

```
An Element can contain: Element (container or leaf)
                        Text (leaf)
                        Attributes (not on tree)
                        Comment (leaf)
                        Unknown (leaf)
```

8.24.2 Constructor & Destructor Documentation

8.24.2.1 XMLNode()

```
tinyxml2::XMLNode::XMLNode (
    XMLDocument * doc) [explicit], [protected]
```

8.24.2.2 ~XMLNode()

```
tinyxml2::XMLNode::~~XMLNode () [protected], [virtual]
```

8.24.3 Member Function Documentation

8.24.3.1 Accept()

```
virtual bool tinyxml2::XMLNode::Accept (
    XMLVisitor * visitor) const [pure virtual]
```

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the [XMLVisitor](#) interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implemented in [tinyxml2::XMLComment](#), [tinyxml2::XMLDeclaration](#), [tinyxml2::XMLDocument](#), [tinyxml2::XMLElement](#), [tinyxml2::XMLText](#), and [tinyxml2::XMLUnknown](#).

8.24.3.2 ChildElementCount() [1/2]

```
int tinyxml2::XMLNode::ChildElementCount () const
```

8.24.3.3 ChildElementCount() [2/2]

```
int tinyxml2::XMLNode::ChildElementCount (
    const char * value) const
```

8.24.3.4 DeepClone()

```
XMLNode * tinyxml2::XMLNode::DeepClone (
    XMLDocument * target) const
```

Make a copy of this node and all its children.

If the 'target' is null, then the nodes will be allocated in the current document. If 'target' is specified, the memory will be allocated in the specified [XMLDocument](#).

NOTE: This is probably not the correct tool to copy a document, since XMLDocuments can have multiple top level XMLNodes. You probably want to use [XMLDocument::DeepCopy\(\)](#)

8.24.3.5 DeleteChild()

```
void tinyxml2::XMLNode::DeleteChild (
    XMLNode * node)
```

Delete a child of this node.

8.24.3.6 DeleteChildren()

```
void tinyxml2::XMLNode::DeleteChildren ()
```

Delete all the children of this node.

8.24.3.7 FirstChild() [1/2]

```
XMLNode * tinyxml2::XMLNode::FirstChild () [inline]
```

8.24.3.8 FirstChild() [2/2]

```
const XMLNode * tinyxml2::XMLNode::FirstChild () const [inline]
```

Get the first child node, or null if none exists.

8.24.3.9 FirstChildElement() [1/2]

```
XMLElement * tinyxml2::XMLNode::FirstChildElement (  
    const char * name = 0) [inline]
```

8.24.3.10 FirstChildElement() [2/2]

```
const XMLElement * tinyxml2::XMLNode::FirstChildElement (  
    const char * name = 0) const
```

Get the first child element, or optionally the first child element with the specified name.

8.24.3.11 GetDocument() [1/2]

```
XMLDocument * tinyxml2::XMLNode::GetDocument () [inline]
```

Get the [XMLDocument](#) that owns this [XMLNode](#).

8.24.3.12 GetDocument() [2/2]

```
const XMLDocument * tinyxml2::XMLNode::GetDocument () const [inline]
```

Get the [XMLDocument](#) that owns this [XMLNode](#).

8.24.3.13 GetLineNum()

```
int tinyxml2::XMLNode::GetLineNum () const [inline]
```

Gets the line number the node is in, if the document was parsed from a file.

8.24.3.14 GetUserData()

```
void * tinyxml2::XMLNode::GetUserData () const [inline]
```

Get user data set into the [XMLNode](#). TinyXML-2 in no way processes or interprets user data. It is initially 0.

8.24.3.15 InsertAfterChild()

```
XMLNode * tinyxml2::XMLNode::InsertAfterChild (
    XMLNode * afterThis,
    XMLNode * addThis)
```

Add a node after the specified child node. If the child node is already part of the document, it is moved from its old location to the new location. Returns the addThis argument or 0 if the afterThis node is not a child of this node, or if the node does not belong to the same document.

8.24.3.16 InsertEndChild()

```
XMLNode * tinyxml2::XMLNode::InsertEndChild (
    XMLNode * addThis)
```

Add a child node as the last (right) child. If the child node is already part of the document, it is moved from its old location to the new location. Returns the addThis argument or 0 if the node does not belong to the same document.

8.24.3.17 InsertFirstChild()

```
XMLNode * tinyxml2::XMLNode::InsertFirstChild (
    XMLNode * addThis)
```

Add a child node as the first (left) child. If the child node is already part of the document, it is moved from its old location to the new location. Returns the addThis argument or 0 if the node does not belong to the same document.

8.24.3.18 LastChild() [1/2]

```
XMLNode * tinyxml2::XMLNode::LastChild () [inline]
```

8.24.3.19 LastChild() [2/2]

```
const XMLNode * tinyxml2::XMLNode::LastChild () const [inline]
```

Get the last child node, or null if none exists.

8.24.3.20 LastChildElement() [1/2]

```
XMLElement * tinyxml2::XMLNode::LastChildElement (
    const char * name = 0) [inline]
```

8.24.3.21 LastChildElement() [2/2]

```
const XMLElement * tinyxml2::XMLNode::LastChildElement (
    const char * name = 0) const
```

Get the last child element or optionally the last child element with the specified name.

8.24.3.22 LinkEndChild()

```
XMLNode * tinyxml2::XMLNode::LinkEndChild (
    XMLNode * addThis) [inline]
```

8.24.3.23 NextSibling() [1/2]

```
XMLNode * tinyxml2::XMLNode::NextSibling () [inline]
```

8.24.3.24 NextSibling() [2/2]

```
const XMLNode * tinyxml2::XMLNode::NextSibling () const [inline]
```

Get the next (right) sibling node of this node.

8.24.3.25 NextSiblingElement() [1/2]

```
XMLElement * tinyxml2::XMLNode::NextSiblingElement (
    const char * name = 0) [inline]
```

8.24.3.26 NextSiblingElement() [2/2]

```
const XMLElement * tinyxml2::XMLNode::NextSiblingElement (
    const char * name = 0) const
```

Get the next (right) sibling element of this node, with an optionally supplied name.

8.24.3.27 NoChildren()

```
bool tinyxml2::XMLNode::NoChildren () const [inline]
```

Returns true if this node has no children.

8.24.3.28 Parent() [1/2]

```
XMLNode * tinyxml2::XMLNode::Parent () [inline]
```

8.24.3.29 Parent() [2/2]

```
const XMLNode * tinyxml2::XMLNode::Parent () const [inline]
```

Get the parent of this node on the DOM.

8.24.3.30 ParseDeep()

```
char * tinyxml2::XMLNode::ParseDeep (
    char * p,
    StrPair * parentEndTag,
    int * curLineNumPtr) [protected], [virtual]
```

Reimplemented in [tinyxml2::XMLComment](#), [tinyxml2::XMLDeclaration](#), [tinyxml2::XMLElement](#), [tinyxml2::XMLText](#), and [tinyxml2::XMLUnknown](#).

8.24.3.31 PreviousSibling() [1/2]

```
XMLNode * tinyxml2::XMLNode::PreviousSibling () [inline]
```

8.24.3.32 PreviousSibling() [2/2]

```
const XMLNode * tinyxml2::XMLNode::PreviousSibling () const [inline]
```

Get the previous (left) sibling node of this node.

8.24.3.33 PreviousSiblingElement() [1/2]

```
XMLElement * tinyxml2::XMLNode::PreviousSiblingElement (
    const char * name = 0) [inline]
```

8.24.3.34 PreviousSiblingElement() [2/2]

```
const XMLElement * tinyxml2::XMLNode::PreviousSiblingElement (
    const char * name = 0) const
```

Get the previous (left) sibling element of this node, with an optionally supplied name.

8.24.3.35 SetUserData()

```
void tinyxml2::XMLNode::SetUserData (
    void * userData) [inline]
```

Set user data into the [XMLNode](#). TinyXML-2 in no way processes or interprets user data. It is initially 0.

8.24.3.36 SetValue()

```
void tinyxml2::XMLNode::SetValue (
    const char * val,
    bool staticMem = false)
```

Set the Value of an XML node.

See also

[Value\(\)](#)

8.24.3.37 ShallowClone()

```
virtual XMLNode * tinyxml2::XMLNode::ShallowClone (  
    XMLDocument * document) const [pure virtual]
```

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (this->GetDocument())

Note: if called on a [XMLDocument](#), this will return null.

Implemented in [tinyxml2::XMLComment](#), [tinyxml2::XMLDeclaration](#), [tinyxml2::XMLDocument](#), [tinyxml2::XMLElement](#), [tinyxml2::XMLText](#), and [tinyxml2::XMLUnknown](#).

8.24.3.38 ShallowEqual()

```
virtual bool tinyxml2::XMLNode::ShallowEqual (  
    const XMLNode * compare) const [pure virtual]
```

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implemented in [tinyxml2::XMLComment](#), [tinyxml2::XMLDeclaration](#), [tinyxml2::XMLDocument](#), [tinyxml2::XMLElement](#), [tinyxml2::XMLText](#), and [tinyxml2::XMLUnknown](#).

8.24.3.39 ToComment() [1/2]

```
virtual XMLComment * tinyxml2::XMLNode::ToComment () [inline], [virtual]
```

Safely cast to a Comment, or null.

Reimplemented in [tinyxml2::XMLComment](#).

8.24.3.40 ToComment() [2/2]

```
virtual const XMLComment * tinyxml2::XMLNode::ToComment () const [inline], [virtual]
```

Reimplemented in [tinyxml2::XMLComment](#).

8.24.3.41 ToDeclaration() [1/2]

```
virtual XMLDeclaration * tinyxml2::XMLNode::ToDeclaration () [inline], [virtual]
```

Safely cast to a Declaration, or null.

Reimplemented in [tinyxml2::XMLDeclaration](#).

8.24.3.42 ToDeclaration() [2/2]

```
virtual const XMLDeclaration * tinyxml2::XMLNode::ToDeclaration () const [inline], [virtual]
```

Reimplemented in [tinyxml2::XMLDeclaration](#).

8.24.3.43 ToDocument() [1/2]

```
virtual XMLDocument * tinyxml2::XMLNode::ToDocument () [inline], [virtual]
```

Safely cast to a Document, or null.

Reimplemented in [tinyxml2::XMLDocument](#).

8.24.3.44 ToDocument() [2/2]

```
virtual const XMLDocument * tinyxml2::XMLNode::ToDocument () const [inline], [virtual]
```

Reimplemented in [tinyxml2::XMLDocument](#).

8.24.3.45 ToElement() [1/2]

```
virtual XMLElement * tinyxml2::XMLNode::ToElement () [inline], [virtual]
```

Safely cast to an [Element](#), or null.

Reimplemented in [tinyxml2::XMLElement](#).

8.24.3.46 ToElement() [2/2]

```
virtual const XMLElement * tinyxml2::XMLNode::ToElement () const [inline], [virtual]
```

Reimplemented in [tinyxml2::XMLElement](#).

8.24.3.47 ToText() [1/2]

```
virtual XMLText * tinyxml2::XMLNode::ToText () [inline], [virtual]
```

Safely cast to Text, or null.

Reimplemented in [tinyxml2::XMLText](#).

8.24.3.48 ToText() [2/2]

```
virtual const XMLText * tinyxml2::XMLNode::ToText () const [inline], [virtual]
```

Reimplemented in [tinyxml2::XMLText](#).

8.24.3.49 ToUnknown() [1/2]

```
virtual XMLUnknown * tinyxml2::XMLNode::ToUnknown () [inline], [virtual]
```

Safely cast to an Unknown, or null.

Reimplemented in [tinyxml2::XMLUnknown](#).

8.24.3.50 ToUnknown() [2/2]

```
virtual const XMLUnknown * tinyxml2::XMLNode::ToUnknown () const [inline], [virtual]
```

Reimplemented in [tinyxml2::XMLUnknown](#).

8.24.3.51 Value()

```
const char * tinyxml2::XMLNode::Value () const
```

The meaning of 'value' changes for the specific type.

```
Document:    empty (NULL is returned, not an empty string)
Element:     name of the element
Comment:     the comment text
Unknown:     the tag contents
Text:        the text string
```

8.24.4 Friends And Related Symbol Documentation**8.24.4.1 XMLDocument**

```
friend class XMLDocument [friend]
```

8.24.4.2 XMLElement

```
friend class XMLElement [friend]
```

8.24.5 Member Data Documentation**8.24.5.1 _document**

```
XMLDocument* tinyxml2::XMLNode::_document [protected]
```

8.24.5.2 _firstChild

```
XMLNode* tinyxml2::XMLNode::_firstChild [protected]
```

8.24.5.3 `_lastChild`

```
XMLNode* tinyxml2::XMLNode::_lastChild [protected]
```

8.24.5.4 `_next`

```
XMLNode* tinyxml2::XMLNode::_next [protected]
```

8.24.5.5 `_parent`

```
XMLNode* tinyxml2::XMLNode::_parent [protected]
```

8.24.5.6 `_parseLineNum`

```
int tinyxml2::XMLNode::_parseLineNum [protected]
```

8.24.5.7 `_prev`

```
XMLNode* tinyxml2::XMLNode::_prev [protected]
```

8.24.5.8 `_userData`

```
void* tinyxml2::XMLNode::_userData [protected]
```

8.24.5.9 `_value`

```
StrPair tinyxml2::XMLNode::_value [mutable], [protected]
```

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

- [app/include/tinyxml2.h](#)
- [app/src/tinyxml2.cpp](#)

8.25 `tinyxml2::XMLPrinter` Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for `tinyxml2::XMLPrinter`:

Collaboration diagram for `tinyxml2::XMLPrinter`:

Public Types

- enum [EscapeAposCharsInAttributes](#) { [ESCAPE_APOS_CHARS_IN_ATTRIBUTES](#), [DONT_ESCAPE_APOS_CHARS_IN_ATTRIBUTES](#) }

Public Member Functions

- [XMLPrinter](#) (FILE *file=0, bool compact=false, int depth=0, [EscapeAposCharsInAttributes](#) aposInAttributes=[ESCAPE_APOS_CHARS_IN_ATTRIBUTES](#))
- virtual [~XMLPrinter](#) ()
- void [PushHeader](#) (bool writeBOM, bool writeDeclaration)
- void [OpenElement](#) (const char *name, bool compactMode=false)
- void [PushAttribute](#) (const char *name, const char *value)
 - If streaming, add an attribute to an open element.*
- void [PushAttribute](#) (const char *name, int value)
- void [PushAttribute](#) (const char *name, unsigned value)
- void [PushAttribute](#) (const char *name, int64_t value)
- void [PushAttribute](#) (const char *name, uint64_t value)
- void [PushAttribute](#) (const char *name, bool value)
- void [PushAttribute](#) (const char *name, double value)
- virtual void [CloseElement](#) (bool compactMode=false)
 - If streaming, close the [Element](#).*
- void [PushText](#) (const char *text, bool cdata=false)
 - Add a text node.*
- void [PushText](#) (int value)
 - Add a text node from an integer.*
- void [PushText](#) (unsigned value)
 - Add a text node from an unsigned.*
- void [PushText](#) (int64_t value)
 - Add a text node from a signed 64bit integer.*
- void [PushText](#) (uint64_t value)
 - Add a text node from an unsigned 64bit integer.*
- void [PushText](#) (bool value)
 - Add a text node from a bool.*
- void [PushText](#) (float value)
 - Add a text node from a float.*
- void [PushText](#) (double value)
 - Add a text node from a double.*
- void [PushComment](#) (const char *comment)
 - Add a comment.*
- void [PushDeclaration](#) (const char *value)
- void [PushUnknown](#) (const char *value)
- virtual bool [VisitEnter](#) (const [XMLDocument](#) &) override
 - Visit a document.*
- virtual bool [VisitExit](#) (const [XMLDocument](#) &) override
 - Visit a document.*
- virtual bool [VisitEnter](#) (const [XMLElement](#) &element, const [XMLAttribute](#) *attribute) override
 - Visit an element.*
- virtual bool [VisitExit](#) (const [XMLElement](#) &element) override
 - Visit an element.*
- virtual bool [Visit](#) (const [XMLText](#) &text) override

Visit a text node.

- virtual bool [Visit](#) (const [XMLComment](#) &comment) override

Visit a comment node.

- virtual bool [Visit](#) (const [XMLDeclaration](#) &declaration) override

Visit a declaration.

- virtual bool [Visit](#) (const [XMLUnknown](#) &unknown) override

Visit an unknown node.

- const char * [CStr](#) () const
- size_t [CStrSize](#) () const
- void [ClearBuffer](#) (bool resetToFirstElement=true)

Public Member Functions inherited from [tinyxml2::XMLVisitor](#)

- virtual [~XMLVisitor](#) ()

Protected Member Functions

- virtual bool [CompactMode](#) (const [XMLElement](#) &)
- virtual void [PrintSpace](#) (int depth)
- virtual void [Print](#) (const char *format,...)
- virtual void [Write](#) (const char *data, size_t size)
- virtual void [Putc](#) (char ch)
- void [Write](#) (const char *data)
- void [SealElementIfJustOpened](#) ()

Protected Attributes

- bool [_elementJustOpened](#)
- [DynArray](#)< const char *, 10 > [_stack](#)

8.25.1 Detailed Description

Printing functionality. The [XMLPrinter](#) gives you more options than the [XMLDocument::Print\(\)](#) method.

It can:

1. Print to memory.
2. Print to a file you provide.
3. Print XML without a [XMLDocument](#).

Print to Memory

```
XMLPrinter printer;
doc.Print( &printer );
SomeFunction( printer.CStr() );
```

Print to a [File](#)

You provide the file pointer.

```
XMLPrinter printer( fp );
doc.Print( &printer );
```

Print without a [XMLDocument](#)

When loading, an XML parser is very useful. However, sometimes when saving, it just gets in the way. The code is often set up for streaming, and constructing the DOM is just overhead.

The Printer supports the streaming case. The following code prints out a trivially simple XML file without ever creating an XML document.

```
XMLPrinter printer( fp );
printer.OpenElement( "foo" );
printer.PushAttribute( "foo", "bar" );
printer.CloseElement();
```

8.25.2 Member Enumeration Documentation

8.25.2.1 EscapeAposCharsInAttributes

```
enum tinyxml2::XMLPrinter::EscapeAposCharsInAttributes
```

Enumerator

ESCAPE_APOS_CHARS_IN_ATTRIBUTES	
DONT_ESCAPE_APOS_CHARS_IN_ATTRIBUTES	

8.25.3 Constructor & Destructor Documentation

8.25.3.1 XMLPrinter()

```
tinyxml2::XMLPrinter::XMLPrinter (
    FILE * file = 0,
    bool compact = false,
    int depth = 0,
    EscapeAposCharsInAttributes aposInAttributes = ESCAPE_APOS_CHARS_IN_ATTRIBUTES)
```

Construct the printer. If the FILE* is specified, this will print to the FILE. Else it will print to memory, and the result is available in [CStr\(\)](#). If 'compact' is set to true, then output is created with only required whitespace and newlines.

8.25.3.2 ~XMLPrinter()

```
virtual tinyxml2::XMLPrinter::~~XMLPrinter () [inline], [virtual]
```

8.25.4 Member Function Documentation

8.25.4.1 ClearBuffer()

```
void tinyxml2::XMLPrinter::ClearBuffer (
    bool resetToFirstElement = true) [inline]
```

If in print to memory mode, reset the buffer to the beginning.

8.25.4.2 CloseElement()

```
void tinyxml2::XMLPrinter::CloseElement (
    bool compactMode = false) [virtual]
```

If streaming, close the [Element](#).

8.25.4.3 CompactMode()

```
virtual bool tinyxml2::XMLPrinter::CompactMode (
    const XMLElement & ) [inline], [protected], [virtual]
```

8.25.4.4 CStr()

```
const char * tinyxml2::XMLPrinter::CStr () const [inline]
```

If in print to memory mode, return a pointer to the XML file in memory.

8.25.4.5 CStrSize()

```
size_t tinyxml2::XMLPrinter::CStrSize () const [inline]
```

If in print to memory mode, return the size of the XML file in memory. (Note the size returned includes the terminating null.)

8.25.4.6 OpenElement()

```
void tinyxml2::XMLPrinter::OpenElement (
    const char * name,
    bool compactMode = false)
```

If streaming, start writing an element. The element must be closed with [CloseElement\(\)](#)

8.25.4.7 Print()

```
void tinyxml2::XMLPrinter::Print (
    const char * format,
    ...) [protected], [virtual]
```

8.25.4.8 PrintSpace()

```
void tinyxml2::XMLPrinter::PrintSpace (  
    int depth) [protected], [virtual]
```

Prints out the space before an element. You may override to change the space and tabs used. A [PrintSpace\(\)](#) override should call [Print\(\)](#).

8.25.4.9 PushAttribute() [1/7]

```
void tinyxml2::XMLPrinter::PushAttribute (  
    const char * name,  
    bool value)
```

8.25.4.10 PushAttribute() [2/7]

```
void tinyxml2::XMLPrinter::PushAttribute (  
    const char * name,  
    const char * value)
```

If streaming, add an attribute to an open element.

8.25.4.11 PushAttribute() [3/7]

```
void tinyxml2::XMLPrinter::PushAttribute (  
    const char * name,  
    double value)
```

8.25.4.12 PushAttribute() [4/7]

```
void tinyxml2::XMLPrinter::PushAttribute (  
    const char * name,  
    int value)
```

8.25.4.13 PushAttribute() [5/7]

```
void tinyxml2::XMLPrinter::PushAttribute (  
    const char * name,  
    int64_t value)
```

8.25.4.14 PushAttribute() [6/7]

```
void tinyxml2::XMLPrinter::PushAttribute (  
    const char * name,  
    uint64_t value)
```

8.25.4.15 PushAttribute() [7/7]

```
void tinyxml2::XMLPrinter::PushAttribute (
    const char * name,
    unsigned value)
```

8.25.4.16 PushComment()

```
void tinyxml2::XMLPrinter::PushComment (
    const char * comment)
```

Add a comment.

8.25.4.17 PushDeclaration()

```
void tinyxml2::XMLPrinter::PushDeclaration (
    const char * value)
```

8.25.4.18 PushHeader()

```
void tinyxml2::XMLPrinter::PushHeader (
    bool writeBOM,
    bool writeDeclaration)
```

If streaming, write the BOM and declaration.

8.25.4.19 PushText() [1/8]

```
void tinyxml2::XMLPrinter::PushText (
    bool value)
```

Add a text node from a bool.

8.25.4.20 PushText() [2/8]

```
void tinyxml2::XMLPrinter::PushText (
    const char * text,
    bool cdata = false)
```

Add a text node.

8.25.4.21 PushText() [3/8]

```
void tinyxml2::XMLPrinter::PushText (
    double value)
```

Add a text node from a double.

8.25.4.22 PushText() [4/8]

```
void tinycl2::XMLPrinter::PushText (
    float value)
```

Add a text node from a float.

8.25.4.23 PushText() [5/8]

```
void tinycl2::XMLPrinter::PushText (
    int value)
```

Add a text node from an integer.

8.25.4.24 PushText() [6/8]

```
void tinycl2::XMLPrinter::PushText (
    int64_t value)
```

Add a text node from a signed 64bit integer.

8.25.4.25 PushText() [7/8]

```
void tinycl2::XMLPrinter::PushText (
    uint64_t value)
```

Add a text node from an unsigned 64bit integer.

8.25.4.26 PushText() [8/8]

```
void tinycl2::XMLPrinter::PushText (
    unsigned value)
```

Add a text node from an unsigned.

8.25.4.27 PushUnknown()

```
void tinycl2::XMLPrinter::PushUnknown (
    const char * value)
```

8.25.4.28 Putc()

```
void tinycl2::XMLPrinter::Putc (
    char ch) [protected], [virtual]
```

8.25.4.29 SealElementIfJustOpened()

```
void tinyxml2::XMLPrinter::SealElementIfJustOpened () [protected]
```

8.25.4.30 Visit() [1/4]

```
bool tinyxml2::XMLPrinter::Visit (
    const XMLComment & ) [override], [virtual]
```

Visit a comment node.

Reimplemented from [tinyxml2::XMLVisitor](#).

8.25.4.31 Visit() [2/4]

```
bool tinyxml2::XMLPrinter::Visit (
    const XMLDeclaration & ) [override], [virtual]
```

Visit a declaration.

Reimplemented from [tinyxml2::XMLVisitor](#).

8.25.4.32 Visit() [3/4]

```
bool tinyxml2::XMLPrinter::Visit (
    const XMLText & ) [override], [virtual]
```

Visit a text node.

Reimplemented from [tinyxml2::XMLVisitor](#).

8.25.4.33 Visit() [4/4]

```
bool tinyxml2::XMLPrinter::Visit (
    const XMLUnknown & ) [override], [virtual]
```

Visit an unknown node.

Reimplemented from [tinyxml2::XMLVisitor](#).

8.25.4.34 VisitEnter() [1/2]

```
bool tinyxml2::XMLPrinter::VisitEnter (
    const XMLDocument & ) [override], [virtual]
```

Visit a document.

Reimplemented from [tinyxml2::XMLVisitor](#).

8.25.4.35 VisitEnter() [2/2]

```
bool tinyxml2::XMLPrinter::VisitEnter (
    const XMLElement & ,
    const XMLAttribute * ) [override], [virtual]
```

Visit an element.

Reimplemented from [tinyxml2::XMLVisitor](#).

8.25.4.36 VisitExit() [1/2]

```
virtual bool tinyxml2::XMLPrinter::VisitExit (
    const XMLDocument & ) [inline], [override], [virtual]
```

Visit a document.

Reimplemented from [tinyxml2::XMLVisitor](#).

8.25.4.37 VisitExit() [2/2]

```
bool tinyxml2::XMLPrinter::VisitExit (
    const XMLElement & ) [override], [virtual]
```

Visit an element.

Reimplemented from [tinyxml2::XMLVisitor](#).

8.25.4.38 Write() [1/2]

```
void tinyxml2::XMLPrinter::Write (
    const char * data) [inline], [protected]
```

8.25.4.39 Write() [2/2]

```
void tinyxml2::XMLPrinter::Write (
    const char * data,
    size_t size) [protected], [virtual]
```

8.25.5 Member Data Documentation

8.25.5.1 _elementJustOpened

```
bool tinyxml2::XMLPrinter::_elementJustOpened [protected]
```

8.25.5.2 `_stack`

```
DynArray< const char*, 10 > tinyxml2::XMLPrinter::_stack [protected]
```

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

- [app/include/tinyxml2.h](#)
- [app/src/tinyxml2.cpp](#)

8.26 `tinyxml2::XMLText` Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for `tinyxml2::XMLText`:

Collaboration diagram for `tinyxml2::XMLText`:

Public Member Functions

- virtual bool [Accept](#) ([XMLVisitor](#) *visitor) const override
- virtual [XMLText](#) * [ToText](#) () override
Safely cast to [Text](#), or null.
- virtual const [XMLText](#) * [ToText](#) () const override
- void [SetCDATA](#) (bool isCDATA)
Declare whether this should be CDATA or standard text.
- bool [CDATA](#) () const
Returns true if this is a CDATA text element.
- virtual [XMLNode](#) * [ShallowClone](#) ([XMLDocument](#) *document) const override
- virtual bool [ShallowEqual](#) (const [XMLNode](#) *compare) const override

Public Member Functions inherited from [tinyxml2::XMLNode](#)

- const [XMLDocument](#) * [GetDocument](#) () const
Get the [XMLDocument](#) that owns this [XMLNode](#).
- [XMLDocument](#) * [GetDocument](#) ()
Get the [XMLDocument](#) that owns this [XMLNode](#).
- virtual [XMLElement](#) * [ToElement](#) ()
Safely cast to an [Element](#), or null.
- virtual [XMLComment](#) * [ToComment](#) ()
Safely cast to a [Comment](#), or null.
- virtual [XMLDocument](#) * [ToDocument](#) ()
Safely cast to a [Document](#), or null.
- virtual [XMLDeclaration](#) * [ToDeclaration](#) ()
Safely cast to a [Declaration](#), or null.
- virtual [XMLUnknown](#) * [ToUnknown](#) ()
Safely cast to an [Unknown](#), or null.
- virtual const [XMLElement](#) * [ToElement](#) () const
- virtual const [XMLComment](#) * [ToComment](#) () const
- virtual const [XMLDocument](#) * [ToDocument](#) () const

- virtual const [XMLDeclaration](#) * [ToDeclaration](#) () const
- virtual const [XMLUnknown](#) * [ToUnknown](#) () const
- int [ChildElementCount](#) (const char *value) const
- int [ChildElementCount](#) () const
- const char * [Value](#) () const
- void [SetValue](#) (const char *val, bool staticMem=false)
- int [GetLineNum](#) () const

Gets the line number the node is in, if the document was parsed from a file.
- const [XMLNode](#) * [Parent](#) () const

Get the parent of this node on the DOM.
- [XMLNode](#) * [Parent](#) ()
- bool [NoChildren](#) () const

Returns true if this node has no children.
- const [XMLNode](#) * [FirstChild](#) () const

Get the first child node, or null if none exists.
- [XMLNode](#) * [FirstChild](#) ()
- const [XMLElement](#) * [FirstChildElement](#) (const char *name=0) const
- [XMLElement](#) * [FirstChildElement](#) (const char *name=0)
- const [XMLNode](#) * [LastChild](#) () const

Get the last child node, or null if none exists.
- [XMLNode](#) * [LastChild](#) ()
- const [XMLElement](#) * [LastChildElement](#) (const char *name=0) const
- [XMLElement](#) * [LastChildElement](#) (const char *name=0)
- const [XMLNode](#) * [PreviousSibling](#) () const

Get the previous (left) sibling node of this node.
- [XMLNode](#) * [PreviousSibling](#) ()
- const [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0) const

Get the previous (left) sibling element of this node, with an optionally supplied name.
- [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0)
- const [XMLNode](#) * [NextSibling](#) () const

Get the next (right) sibling node of this node.
- [XMLNode](#) * [NextSibling](#) ()
- const [XMLElement](#) * [NextSiblingElement](#) (const char *name=0) const

Get the next (right) sibling element of this node, with an optionally supplied name.
- [XMLElement](#) * [NextSiblingElement](#) (const char *name=0)
- [XMLNode](#) * [InsertEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [LinkEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertFirstChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertAfterChild](#) ([XMLNode](#) *afterThis, [XMLNode](#) *addThis)
- void [DeleteChildren](#) ()
- void [DeleteChild](#) ([XMLNode](#) *node)
- [XMLNode](#) * [DeepClone](#) ([XMLDocument](#) *target) const
- void [SetUserData](#) (void *userData)
- void * [GetUserData](#) () const

Protected Member Functions

- [XMLText](#) ([XMLDocument](#) *doc)
- virtual [~XMLText](#) ()
- char * [ParseDeep](#) (char *p, [StrPair](#) *parentEndTag, int *curLineNumPtr) override

Protected Member Functions inherited from [tinyxml2::XMLNode](#)

- [XMLNode](#) ([XMLDocument](#) *)
- virtual [~XMLNode](#) ()

Friends

- class [XMLDocument](#)

Additional Inherited Members

Protected Attributes inherited from [tinyxml2::XMLNode](#)

- [XMLDocument](#) * [_document](#)
- [XMLNode](#) * [_parent](#)
- [StrPair](#) [_value](#)
- int [_parseLineNum](#)
- [XMLNode](#) * [_firstChild](#)
- [XMLNode](#) * [_lastChild](#)
- [XMLNode](#) * [_prev](#)
- [XMLNode](#) * [_next](#)
- void * [_userData](#)

8.26.1 Detailed Description

XML text.

Note that a text node can have child element nodes, for example:

```
<root>This is <b>bold</b></root>
```

A text node can have 2 ways to output the text. "normal" output and CDATA. It will default to the mode it was parsed from the XML file and you generally want to leave it alone, but you can change the output mode with [SetCDATA\(\)](#) and query it with [CDATA\(\)](#).

8.26.2 Constructor & Destructor Documentation

8.26.2.1 XMLText()

```
tinyxml2::XMLText::XMLText (
    XMLDocument * doc) [inline], [explicit], [protected]
```

8.26.2.2 ~XMLText()

```
virtual tinyxml2::XMLText::~~XMLText () [inline], [protected], [virtual]
```

8.26.3 Member Function Documentation

8.26.3.1 Accept()

```
bool tinyxml2::XMLText::Accept (
    XMLVisitor * visitor) const [override], [virtual]
```

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the [XMLVisitor](#) interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinyxml2::XMLNode](#).

8.26.3.2 CData()

```
bool tinyxml2::XMLText::CData () const [inline]
```

Returns true if this is a CDATA text element.

8.26.3.3 ParseDeep()

```
char * tinyxml2::XMLText::ParseDeep (
    char * p,
    StrPair * parentEndTag,
    int * curLineNumPtr) [override], [protected], [virtual]
```

Reimplemented from [tinyxml2::XMLNode](#).

8.26.3.4 SetCData()

```
void tinyxml2::XMLText::SetCData (
    bool isCData) [inline]
```

Declare whether this should be CDATA or standard text.

8.26.3.5 ShallowClone()

```
XMLNode * tinyxml2::XMLText::ShallowClone (
    XMLDocument * document) const [override], [virtual]
```

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (this->GetDocument())

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

8.26.3.6 ShallowEqual()

```
bool tinyxml2::XMLText::ShallowEqual (
    const XMLNode * compare) const [override], [virtual]
```

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

8.26.3.7 ToText() [1/2]

```
virtual const XMLText * tinyxml2::XMLText::ToText () const [inline], [override], [virtual]
```

Reimplemented from [tinyxml2::XMLNode](#).

8.26.3.8 ToText() [2/2]

```
virtual XMLText * tinyxml2::XMLText::ToText () [inline], [override], [virtual]
```

Safely cast to Text, or null.

Reimplemented from [tinyxml2::XMLNode](#).

8.26.4 Friends And Related Symbol Documentation

8.26.4.1 XMLDocument

```
friend class XMLDocument [friend]
```

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

- app/include/[tinyxml2.h](#)
- app/src/[tinyxml2.cpp](#)

8.27 tinyxml2::XMLUnknown Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLUnknown:

Collaboration diagram for tinyxml2::XMLUnknown:

Public Member Functions

- virtual [XMLUnknown](#) * [ToUnknown](#) () override
Safely cast to an Unknown, or null.
- virtual const [XMLUnknown](#) * [ToUnknown](#) () const override
- virtual bool [Accept](#) ([XMLVisitor](#) *visitor) const override
- virtual [XMLNode](#) * [ShallowClone](#) ([XMLDocument](#) *document) const override
- virtual bool [ShallowEqual](#) (const [XMLNode](#) *compare) const override

Public Member Functions inherited from [tinyxml2::XMLNode](#)

- const [XMLDocument](#) * [GetDocument](#) () const
Get the XMLDocument that owns this XMLNode.
- [XMLDocument](#) * [GetDocument](#) ()
Get the XMLDocument that owns this XMLNode.
- virtual [XMLElement](#) * [ToElement](#) ()
Safely cast to an Element, or null.
- virtual [XMLText](#) * [ToText](#) ()
Safely cast to Text, or null.
- virtual [XMLComment](#) * [ToComment](#) ()
Safely cast to a Comment, or null.
- virtual [XMLDocument](#) * [ToDocument](#) ()
Safely cast to a Document, or null.
- virtual [XMLDeclaration](#) * [ToDeclaration](#) ()
Safely cast to a Declaration, or null.
- virtual const [XMLElement](#) * [ToElement](#) () const
- virtual const [XMLText](#) * [ToText](#) () const
- virtual const [XMLComment](#) * [ToComment](#) () const
- virtual const [XMLDocument](#) * [ToDocument](#) () const
- virtual const [XMLDeclaration](#) * [ToDeclaration](#) () const
- int [ChildElementCount](#) (const char *value) const
- int [ChildElementCount](#) () const
- const char * [Value](#) () const
- void [SetValue](#) (const char *val, bool staticMem=false)
- int [GetLineNum](#) () const
Gets the line number the node is in, if the document was parsed from a file.
- const [XMLNode](#) * [Parent](#) () const
Get the parent of this node on the DOM.
- [XMLNode](#) * [Parent](#) ()
- bool [NoChildren](#) () const
Returns true if this node has no children.
- const [XMLNode](#) * [FirstChild](#) () const

Get the first child node, or null if none exists.

- [XMLNode](#) * [FirstChild](#) ()
- const [XMLElement](#) * [FirstChildElement](#) (const char *name=0) const
- [XMLElement](#) * [FirstChildElement](#) (const char *name=0)
- const [XMLNode](#) * [LastChild](#) () const

Get the last child node, or null if none exists.

- [XMLNode](#) * [LastChild](#) ()
- const [XMLElement](#) * [LastChildElement](#) (const char *name=0) const
- [XMLElement](#) * [LastChildElement](#) (const char *name=0)
- const [XMLNode](#) * [PreviousSibling](#) () const

Get the previous (left) sibling node of this node.

- [XMLNode](#) * [PreviousSibling](#) ()
- const [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0) const

Get the previous (left) sibling element of this node, with an optionally supplied name.

- [XMLElement](#) * [PreviousSiblingElement](#) (const char *name=0)
- const [XMLNode](#) * [NextSibling](#) () const

Get the next (right) sibling node of this node.

- [XMLNode](#) * [NextSibling](#) ()
- const [XMLElement](#) * [NextSiblingElement](#) (const char *name=0) const

Get the next (right) sibling element of this node, with an optionally supplied name.

- [XMLElement](#) * [NextSiblingElement](#) (const char *name=0)
- [XMLNode](#) * [InsertEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [LinkEndChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertFirstChild](#) ([XMLNode](#) *addThis)
- [XMLNode](#) * [InsertAfterChild](#) ([XMLNode](#) *afterThis, [XMLNode](#) *addThis)
- void [DeleteChildren](#) ()
- void [DeleteChild](#) ([XMLNode](#) *node)
- [XMLNode](#) * [DeepClone](#) ([XMLDocument](#) *target) const
- void [SetUserData](#) (void *userData)
- void * [GetUserData](#) () const

Protected Member Functions

- [XMLUnknown](#) ([XMLDocument](#) *doc)
- virtual [~XMLUnknown](#) ()
- char * [ParseDeep](#) (char *p, [StrPair](#) *parentEndTag, int *curLineNumPtr) override

Protected Member Functions inherited from [tinyxml2::XMLNode](#)

- [XMLNode](#) ([XMLDocument](#) *)
- virtual [~XMLNode](#) ()

Friends

- class [XMLDocument](#)

Additional Inherited Members

Protected Attributes inherited from [tinyxml2::XMLNode](#)

- [XMLDocument](#) * [_document](#)
- [XMLNode](#) * [_parent](#)
- [StrPair](#) [_value](#)
- int [_parseLineNum](#)
- [XMLNode](#) * [_firstChild](#)
- [XMLNode](#) * [_lastChild](#)
- [XMLNode](#) * [_prev](#)
- [XMLNode](#) * [_next](#)
- void * [_userData](#)

8.27.1 Detailed Description

Any tag that TinyXML-2 doesn't recognize is saved as an unknown. It is a tag of text, but should not be modified. It will be written back to the XML, unchanged, when the file is saved.

DTD tags get thrown into XMLUnknowns.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 XMLUnknown()

```
tinyxml2::XMLUnknown::XMLUnknown (
    XMLDocument * doc) [explicit], [protected]
```

8.27.2.2 ~XMLUnknown()

```
tinyxml2::XMLUnknown::~~XMLUnknown () [protected], [virtual]
```

8.27.3 Member Function Documentation

8.27.3.1 Accept()

```
bool tinyxml2::XMLUnknown::Accept (
    XMLVisitor * visitor) const [override], [virtual]
```

Accept a hierarchical visit of the nodes in the TinyXML-2 DOM. Every node in the XML tree will be conditionally visited and the host will be called back via the [XMLVisitor](#) interface.

This is essentially a SAX interface for TinyXML-2. (Note however it doesn't re-parse the XML for the callbacks, so the performance of TinyXML-2 is unchanged by using this interface versus any other.)

The interface has been based on ideas from:

- <http://www.saxproject.org/>
- <http://c2.com/cgi/wiki?HierarchicalVisitorPattern>

Which are both good references for "visiting".

An example of using [Accept\(\)](#):

```
XMLPrinter printer;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements [tinyxml2::XMLNode](#).

8.27.3.2 ParseDeep()

```
char * tinyxml2::XMLUnknown::ParseDeep (  
    char * p,  
    StrPair * parentEndTag,  
    int * curLineNumPtr) [override], [protected], [virtual]
```

Reimplemented from [tinyxml2::XMLNode](#).

8.27.3.3 ShallowClone()

```
XMLNode * tinyxml2::XMLUnknown::ShallowClone (  
    XMLDocument * document) const [override], [virtual]
```

Make a copy of this node, but not its children. You may pass in a Document pointer that will be the owner of the new Node. If the 'document' is null, then the node returned will be allocated from the current Document. (this->[GetDocument\(\)](#))

Note: if called on a [XMLDocument](#), this will return null.

Implements [tinyxml2::XMLNode](#).

8.27.3.4 ShallowEqual()

```
bool tinyxml2::XMLUnknown::ShallowEqual (  
    const XMLNode * compare) const [override], [virtual]
```

Test if 2 nodes are the same, but don't test children. The 2 nodes do not need to be in the same Document.

Note: if called on a [XMLDocument](#), this will return false.

Implements [tinyxml2::XMLNode](#).

8.27.3.5 ToUnknown() [1/2]

```
virtual const XMLUnknown * tinyxml2::XMLUnknown::ToUnknown () const [inline], [override],  
[virtual]
```

Reimplemented from [tinyxml2::XMLNode](#).

8.27.3.6 ToUnknown() [2/2]

```
virtual XMLUnknown * tinyxml2::XMLUnknown::ToUnknown () [inline], [override], [virtual]
```

Safely cast to an Unknown, or null.

Reimplemented from [tinyxml2::XMLNode](#).

8.27.4 Friends And Related Symbol Documentation

8.27.4.1 XMLDocument

```
friend class XMLDocument [friend]
```

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

- [app/include/tinyxml2.h](#)
- [app/src/tinyxml2.cpp](#)

8.28 tinyxml2::XMLUtil Class Reference

```
#include <tinyxml2.h>
```

Static Public Member Functions

- static const char * [SkipWhiteSpace](#) (const char *p, int *curLineNumPtr)
- static char * [SkipWhiteSpace](#) (char *const p, int *curLineNumPtr)
- static bool [IsWhiteSpace](#) (char p)
- static bool [IsNameStartChar](#) (unsigned char ch)
- static bool [IsNameChar](#) (unsigned char ch)
- static bool [IsPrefixHex](#) (const char *p)
- static bool [StringEqual](#) (const char *p, const char *q, int nChar=INT_MAX)
- static bool [IsUTF8Continuation](#) (const char p)
- static const char * [ReadBOM](#) (const char *p, bool *hasBOM)
- static const char * [GetCharacterRef](#) (const char *p, char *value, int *length)
- static void [ConvertUTF32ToUTF8](#) (unsigned long input, char *output, int *length)
- static void [ToStr](#) (int v, char *buffer, int bufferSize)
- static void [ToStr](#) (unsigned v, char *buffer, int bufferSize)
- static void [ToStr](#) (bool v, char *buffer, int bufferSize)
- static void [ToStr](#) (float v, char *buffer, int bufferSize)
- static void [ToStr](#) (double v, char *buffer, int bufferSize)
- static void [ToStr](#) (int64_t v, char *buffer, int bufferSize)
- static void [ToStr](#) (uint64_t v, char *buffer, int bufferSize)
- static bool [ToInt](#) (const char *str, int *value)
- static bool [ToUnsigned](#) (const char *str, unsigned *value)
- static bool [ToBool](#) (const char *str, bool *value)
- static bool [ToFloat](#) (const char *str, float *value)
- static bool [ToDouble](#) (const char *str, double *value)
- static bool [ToInt64](#) (const char *str, int64_t *value)
- static bool [ToUnsigned64](#) (const char *str, uint64_t *value)
- static void [SetBoolSerialization](#) (const char *writeTrue, const char *writeFalse)

8.28.1 Member Function Documentation

8.28.1.1 ConvertUTF32ToUTF8()

```
void tinyxml2::XMLUtil::ConvertUTF32ToUTF8 (
    unsigned long input,
    char * output,
    int * length) [static]
```

8.28.1.2 GetCharacterRef()

```
const char * tinyxml2::XMLUtil::GetCharacterRef (  
    const char * p,  
    char * value,  
    int * length) [static]
```

8.28.1.3 IsNameChar()

```
bool tinyxml2::XMLUtil::IsNameChar (  
    unsigned char ch) [inline], [static]
```

8.28.1.4 IsNameStartChar()

```
bool tinyxml2::XMLUtil::IsNameStartChar (  
    unsigned char ch) [inline], [static]
```

8.28.1.5 IsPrefixHex()

```
bool tinyxml2::XMLUtil::IsPrefixHex (  
    const char * p) [inline], [static]
```

8.28.1.6 IsUTF8Continuation()

```
bool tinyxml2::XMLUtil::IsUTF8Continuation (  
    const char p) [inline], [static]
```

8.28.1.7 IsWhiteSpace()

```
bool tinyxml2::XMLUtil::IsWhiteSpace (  
    char p) [inline], [static]
```

8.28.1.8 ReadBOM()

```
const char * tinyxml2::XMLUtil::ReadBOM (  
    const char * p,  
    bool * hasBOM) [static]
```

8.28.1.9 SetBoolSerialization()

```
void tinyxml2::XMLUtil::SetBoolSerialization (  
    const char * writeTrue,  
    const char * writeFalse) [static]
```

8.28.1.10 SkipWhiteSpace() [1/2]

```
char * tinyxml2::XMLUtil::SkipWhiteSpace (  
    char *const p,  
    int * curLineNumPtr) [inline], [static]
```

8.28.1.11 SkipWhiteSpace() [2/2]

```
const char * tinyxml2::XMLUtil::SkipWhiteSpace (  
    const char * p,  
    int * curLineNumPtr) [inline], [static]
```

8.28.1.12 StringEqual()

```
bool tinyxml2::XMLUtil::StringEqual (  
    const char * p,  
    const char * q,  
    int nChar = INT_MAX) [inline], [static]
```

8.28.1.13 ToBool()

```
bool tinyxml2::XMLUtil::ToBool (  
    const char * str,  
    bool * value) [static]
```

8.28.1.14 ToDouble()

```
bool tinyxml2::XMLUtil::ToDouble (  
    const char * str,  
    double * value) [static]
```

8.28.1.15 ToFloat()

```
bool tinyxml2::XMLUtil::ToFloat (  
    const char * str,  
    float * value) [static]
```

8.28.1.16 ToInt()

```
bool tinyxml2::XMLUtil::ToInt (  
    const char * str,  
    int * value) [static]
```

8.28.1.17 ToInt64()

```
bool tinyxml2::XMLUtil::ToInt64 (
    const char * str,
    int64_t * value) [static]
```

8.28.1.18 ToStr() [1/7]

```
void tinyxml2::XMLUtil::ToStr (
    bool v,
    char * buffer,
    int bufferSize) [static]
```

8.28.1.19 ToStr() [2/7]

```
void tinyxml2::XMLUtil::ToStr (
    double v,
    char * buffer,
    int bufferSize) [static]
```

8.28.1.20 ToStr() [3/7]

```
void tinyxml2::XMLUtil::ToStr (
    float v,
    char * buffer,
    int bufferSize) [static]
```

8.28.1.21 ToStr() [4/7]

```
void tinyxml2::XMLUtil::ToStr (
    int v,
    char * buffer,
    int bufferSize) [static]
```

8.28.1.22 ToStr() [5/7]

```
void tinyxml2::XMLUtil::ToStr (
    int64_t v,
    char * buffer,
    int bufferSize) [static]
```

8.28.1.23 ToStr() [6/7]

```
void tinyxml2::XMLUtil::ToStr (
    uint64_t v,
    char * buffer,
    int bufferSize) [static]
```

8.28.1.24 ToStr() [7/7]

```
void tinyxml2::XMLUtil::ToStr (
    unsigned v,
    char * buffer,
    int bufferSize) [static]
```

8.28.1.25 ToUnsigned()

```
bool tinyxml2::XMLUtil::ToUnsigned (
    const char * str,
    unsigned * value) [static]
```

8.28.1.26 ToUnsigned64()

```
bool tinyxml2::XMLUtil::ToUnsigned64 (
    const char * str,
    uint64_t * value) [static]
```

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

- [app/include/tinyxml2.h](#)
- [app/src/tinyxml2.cpp](#)

8.29 tinyxml2::XMLVisitor Class Reference

```
#include <tinyxml2.h>
```

Inheritance diagram for tinyxml2::XMLVisitor:

Public Member Functions

- virtual [~XMLVisitor](#) ()
- virtual bool [VisitEnter](#) (const [XMLDocument](#) &)
Visit a document.
- virtual bool [VisitExit](#) (const [XMLDocument](#) &)
Visit a document.
- virtual bool [VisitEnter](#) (const [XMLElement](#) &, const [XMLAttribute](#) *)
Visit an element.
- virtual bool [VisitExit](#) (const [XMLElement](#) &)
Visit an element.
- virtual bool [Visit](#) (const [XMLDeclaration](#) &)
Visit a declaration.
- virtual bool [Visit](#) (const [XMLText](#) &)
Visit a text node.
- virtual bool [Visit](#) (const [XMLComment](#) &)
Visit a comment node.
- virtual bool [Visit](#) (const [XMLUnknown](#) &)
Visit an unknown node.

8.29.1 Detailed Description

Implements the interface to the "Visitor pattern" (see the `Accept()` method.) If you call the `Accept()` method, it requires being passed a [XMLVisitor](#) class to handle callbacks. For nodes that contain other nodes ([Document](#), [Element](#)) you will get called with a `VisitEnter/VisitExit` pair. Nodes that are always leafs are simply called with `Visit()`.

If you return 'true' from a `Visit` method, recursive parsing will continue. If you return false, **no children of this node or its siblings** will be visited.

All flavors of `Visit` methods have a default implementation that returns 'true' (continue visiting). You need to only override methods that are interesting to you.

Generally `Accept()` is called on the [XMLDocument](#), although all nodes support visiting.

You should never change the document from a callback.

See also

[XMLNode::Accept\(\)](#)

8.29.2 Constructor & Destructor Documentation

8.29.2.1 ~XMLVisitor()

```
virtual tinyxml2::XMLVisitor::~XMLVisitor () [inline], [virtual]
```

8.29.3 Member Function Documentation

8.29.3.1 Visit() [1/4]

```
virtual bool tinyxml2::XMLVisitor::Visit (
    const XMLComment & ) [inline], [virtual]
```

Visit a comment node.

Reimplemented in [tinyxml2::XMLPrinter](#).

8.29.3.2 Visit() [2/4]

```
virtual bool tinyxml2::XMLVisitor::Visit (
    const XMLDeclaration & ) [inline], [virtual]
```

Visit a declaration.

Reimplemented in [tinyxml2::XMLPrinter](#).

8.29.3.3 Visit() [3/4]

```
virtual bool tinyxml2::XMLVisitor::Visit (
    const XMLText & ) [inline], [virtual]
```

Visit a text node.

Reimplemented in [tinyxml2::XMLPrinter](#).

8.29.3.4 Visit() [4/4]

```
virtual bool tinyxml2::XMLVisitor::Visit (
    const XMLUnknown & ) [inline], [virtual]
```

Visit an unknown node.

Reimplemented in [tinyxml2::XMLPrinter](#).

8.29.3.5 VisitEnter() [1/2]

```
virtual bool tinyxml2::XMLVisitor::VisitEnter (
    const XMLDocument & ) [inline], [virtual]
```

Visit a document.

Reimplemented in [tinyxml2::XMLPrinter](#).

8.29.3.6 VisitEnter() [2/2]

```
virtual bool tinyxml2::XMLVisitor::VisitEnter (
    const XMLElement & ,
    const XMLAttribute * ) [inline], [virtual]
```

Visit an element.

Reimplemented in [tinyxml2::XMLPrinter](#).

8.29.3.7 VisitExit() [1/2]

```
virtual bool tinyxml2::XMLVisitor::VisitExit (
    const XMLDocument & ) [inline], [virtual]
```

Visit a document.

Reimplemented in [tinyxml2::XMLPrinter](#).

8.29.3.8 VisitExit() [2/2]

```
virtual bool tinyxml2::XMLVisitor::VisitExit (
    const XMLElement & ) [inline], [virtual]
```

Visit an element.

Reimplemented in [tinyxml2::XMLPrinter](#).

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

- [app/include/tinyxml2.h](#)

Chapter 9

File Documentation

9.1 app/include/app.hpp File Reference

```
#include <iostream>
#include "menu.hpp"
#include "fileSystem.hpp"
```

Include dependency graph for app.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [App](#)
Main application logic.

9.2 app.hpp

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <iostream>
00004
00005 #include "menu.hpp"
00006 #include "fileSystem.hpp"
00007
00008
00013 class App {
00014     public:
00015         App();
00016
00017         void run();
00018     private:
00019         Menu mainMenu;
00020         FileSystem fs;
00021
00022         void loadSave();
00023         void statistics();
00024         void searches();
00025         void operations();
00026         void advanced();
00027 };
00028
```

9.3 app/include/date.hpp File Reference

```
#include <iostream>
#include <string>
#include <cstdint>
#include <filesystem>
```

Include dependency graph for date.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [Date](#)

Handle date operations and storage.

9.4 date.hpp

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <iostream>
00004 #include <string>
00005 #include <cstdint>
00006 #include <filesystem>
00007
00008
00013 class Date {
00014     public:
00015         Date();
00016         Date(std::uint16_t day, std::uint16_t month, std::uint16_t year);
00017         Date(const std::string &date);
00018
00019         static Date convertFileTime(const std::filesystem::file_time_type &ftime);
00020         static Date now();
00021
00022         std::string getFormattedDate() const;
00023         std::uint16_t getDay() const;
00024         std::uint16_t getMonth() const;
00025         std::uint16_t getYear() const;
00026     private:
00027         std::uint16_t day, month, year;
00028
00029         void parse(const std::string &dateStr);
00030 };
00031
```

9.5 app/include/element.hpp File Reference

```
#include <string>
#include <ostream>
#include "filename.hpp"
```

Include dependency graph for element.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [Element](#)

Base class for a filesystem element.

Enumerations

- enum class `ElementType` { `Folder` , `File` }

9.5.1 Enumeration Type Documentation

9.5.1.1 ElementType

```
enum class ElementType [strong]
```

Enumerator

Folder	
File	

9.6 element.hpp

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <string>
00004 #include <ostream>
00005
00006 #include "filename.hpp"
00007
00008 enum class ElementType { Folder, File };
00009
00014 class Element {
00015     public:
00016         Element(const std::string& name);
00017         virtual ~Element() = default;
00018
00019         virtual bool isFile() const = 0;
00020         virtual bool isFolder() const = 0;
00021
00022         // Getters
00023         const Filename getName() const;
00024         Filename& getName();
00025         // Setters
00026         void setName(const std::string& name);
00027     protected:
00028         Filename name;
00029 };
00030
```

9.7 app/include/file.hpp File Reference

```
#include <iostream>
#include <string>
#include <stdint>
#include "filename.hpp"
#include "date.hpp"
#include "element.hpp"
```

Include dependency graph for file.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [File](#)

Handle all file related operations.

9.8 file.hpp

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include <iostream>
00004 #include <string>
00005 #include <cstdint>
00006
00007 #include "filename.hpp"
00008 #include "date.hpp"
00009 #include "element.hpp"
00010
00011
00016 class File : public Element {
00017     public:
00018         File(const std::string &filename);
00019         File(const std::string &filename, Date date, const std::uintmax_t size);
00020         File(const std::string &filename, const std::string &date, const std::uintmax_t size);
00021
00022         // Setters
00023         void setDate(const Date &newDate);
00024         // Getters
00025         std::uintmax_t getSize() const;
00026         const Date getDate() const;
00027
00028         bool isFile() const override { return true; }
00029         bool isFolder() const override { return false; }
00030     private:
00031         std::uintmax_t size;
00032         Date date;
00033 };
00034
```

9.9 app/include/filename.hpp File Reference

```

#include <iostream>
#include <string>
#include <cstdint>

```

Include dependency graph for filename.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [Filename](#)

Handle a file/folder name.

9.10 filename.hpp

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include <iostream>
00004 #include <string>
00005 #include <cstdint>
00006

```

```

00007
00012 class Filename {
00013     public:
00014         Filename(const std::string &fullname);
00015         Filename(const std::string &name, const std::string &extention);
00016
00017         void generateSequentialName(std::uint16_t counter);
00018         // Setters
00019         void setExtension(const std::string &newExtension);
00020         void setName(const std::string &newName);
00021         // Getters
00022         std::string getFullname() const;
00023         std::string getName() const;
00024         std::string getExtension() const;
00025     private:
00026         std::string name;
00027         std::string extension;
00028         std::string getExtension(const std::string& fullname);
00029         std::string getName(const std::string& fullname);
00030 };
00031

```

9.11 app/include/fileSystem.hpp File Reference

```

#include <iostream>
#include <string>
#include <list>
#include <cstdint>
#include <memory>
#include <optional>
#include "folder.hpp"
#include "element.hpp"

```

Include dependency graph for fileSystem.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [FileSystem](#)

Handle a filesystem and its operations.

9.12 fileSystem.hpp

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include <iostream>
00004 #include <string>
00005 #include <list>
00006 #include <cstdint>
00007 #include <memory>
00008 #include <optional>
00009
00010 #include "folder.hpp"
00011 #include "element.hpp"
00012
00013
00018 class FileSystem {
00019     public:
00020         FileSystem();
00021         FileSystem(const std::string &rootPath);
00022
00023         bool load(); // 1
00024         bool load(const std::string &rootPath); // 1
00025
00026         void clear();
00027
00028         // Stats

```

```

00029         std::uint32_t countFiles() const; // 2
00030         std::uint32_t countFolders() const; // 3
00031         std::uintmax_t memory() const; // 4
00032
00033         std::string *mostElementsFolder() const; // 5
00034         std::string *leastElementsFolder() const; // 6
00035         std::string *largestFile() const; // 7
00036         std::string *largestFolder() const; // 8
00037
00038         // XML
00039         void saveToXML(const std::string &s) const; // 11
00040         bool readFromXML(const std::string &s); // 12
00041
00042         // File operations
00043         bool removeAll(const std::string &name, ElementType type); // 10
00044         bool moveFile(const std::string &file, const std::string &newDir); // 13
00045         bool moveFolder(const std::string &oldDir, const std::string &newDir); // 14
00046         std::string *getFileDate(const std::string &file); // 15
00047         void renameAllFiles(const std::string &currentName, const std::string &newName); // 19
00048         bool copyBatch(const std::string &pattern, const std::string &originDir, const std::string
&destinDir); // 21
00049
00050         // Search operations
00051         std::optional<std::string> search(const std::string &name, ElementType type); // 9
00052         void searchAllFolders(std::list<std::string> &li, const std::string &folder) const; // 17
00053         void searchAllFiles(std::list<std::string> &li, const std::string &file) const; // 18
00054
00055         // Others
00056         bool checkDupFiles(); // 20
00057         void tree(std::ostream &out, std::ostream *mirror = nullptr); // 16
00058
00059         // Setters
00060         void setPath(const std::string& path);
00061
00062         // Getters
00063         const std::string& getPath() const;
00064     private:
00065         std::unique_ptr<Folder> root;
00066         std::string path; // Path to the root directory
00067 };
00068

```

9.13 app/include/folder.hpp File Reference

```

#include <iostream>
#include <string>
#include <vector>
#include <memory>
#include <filesystem>
#include <cstdint>
#include <list>
#include <unordered_set>
#include "tinyxml2.h"
#include "file.hpp"
#include "element.hpp"

```

Include dependency graph for folder.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [Folder](#)

Handle all folder related operations.

Variables

- constexpr std::uint16_t SPACES_PER_LEVEL = 4

9.13.1 Variable Documentation

9.13.1.1 SPACES_PER_LEVEL

```
std::uint16_t SPACES_PER_LEVEL = 4 [constexpr]
```

9.14 folder.hpp

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <iostream>
00004 #include <string>
00005 #include <vector>
00006 #include <memory>
00007 #include <filesystem>
00008 #include <stdint>
00009 #include <list>
00010 #include <unordered_set>
00011 // tinyxml2 library
00012 #include "tinyxml2.h"
00013
00014 #include "file.hpp"
00015 #include "element.hpp"
00016
00017 constexpr std::uint16_t SPACES_PER_LEVEL = 4;
00018
00019 namespace fs = std::filesystem;
00020 namespace xml = tinyxml2;
00021
00022 class Folder : public Element {
00023 public:
00024     Folder(std::string name, Folder *father);
00025
00026     bool load(const fs::path& path);
00027
00028     void add(std::unique_ptr<Element> element);
00029     std::unique_ptr<Element> remove(const std::string& name, ElementType type);
00030
00031     bool copyBatch(const std::string &pattern, Folder *destin);
00032
00033     std::uint32_t countFiles() const;
00034     std::uint32_t countFolders() const;
00035     std::uintmax_t memory() const;
00036
00037     const Folder *mostElementsFolder() const;
00038     const Folder *leastElementsFolder() const;
00039     const File *largestFile() const;
00040     const Folder *largestFolder(bool isRoot) const;
00041
00042     void saveToXML(xml::XMLDocument &doc, xml::XMLElement *parentElem) const;
00043     void readFromXML(xml::XMLElement *dirElem);
00044
00045     std::string searchFolder(const std::string& name) const;
00046     void searchAllFolders(std::list<std::string> &li, const std::string& name, const std::string&
00047 path) const;
00048     std::string searchFile(const std::string& name) const;
00049     void searchAllFiles(std::list<std::string> &li, const std::string& name, const std::string&
00050 path) const;
00051
00052     bool checkDupFiles(std::unordered_set<std::string>& names);
00053     void tree(const std::string &prefix, bool isLast, std::ostream &out, std::ostream *mirror)
00054 const;
00055
00056     bool removeAll(const std::string &name, ElementType type);
00057     void renameAllFiles(const std::string &currentName, const std::string &newName);
00058
00059     bool hasFile(const std::string &name) const;
00060     // Setters
00061     void setParent(Folder *parent);
00062     // Getters
00063     Folder *getFolderByName(const std::string& name) const;
00064     File *getFileByName(const std::string& name) const;
00065     Folder *getFolderByFileName(const std::string& name) const;
00066     Folder* getParent() const;
00067     const std::string getName() const;
00068
00069 }
```

```

00070         bool isFile() const override { return false; }
00071         bool isFolder() const override { return true; }
00072     private:
00073         std::vector<std::unique_ptr<Element>> elements;
00074         Folder *root;
00075 };

```

9.15 app/include/input.hpp File Reference

```

#include <string>
#include <iostream>
#include <algorithm>

```

Include dependency graph for input.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [Input](#)
Handle input from the user.

9.16 input.hpp

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include <string>
00004 #include <iostream>
00005 #include <algorithm>
00006
00007
00012 class Input {
00013     public:
00014         static std::string getString(const std::string& prompt, bool allowEmpty = false);
00015
00016         static void wait();
00017
00018     private:
00019         static std::string trim(const std::string& str);
00020 };
00021

```

9.17 app/include/menu.hpp File Reference

```

#include <string>
#include <vector>

```

Include dependency graph for menu.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [Menu](#)
Handle menu output and option chosen.

9.18 menu.hpp

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <string>
00004 #include <vector>
00005
00010 class Menu {
00011     public:
00012         Menu(const std::string& title, const std::vector<std::string>& options);
00013
00014         static bool askYesNo(const std::string& question, bool clearTerminal = false);
00015
00016         int show(bool clearTerminal = true);
00017     private:
00018         std::string title;
00019         std::vector<std::string> options;
00020 };
00021
```

9.19 app/include/systemConfig.hpp File Reference

```
#include <locale>
#include <iostream>
#include <locale.h>
```

Include dependency graph for systemConfig.hpp:

Classes

- class [SystemConfig](#)
Configure system output.

9.20 systemConfig.hpp

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <locale>
00004 #include <iostream>
00005
00006 #ifndef _WIN32
00007     #include <windows.h> // using namespace std gives compilation error (DO NOT USE on .hpp files)
00008 #endif
00009 #include <locale.h>
00010
00015 class SystemConfig {
00016     public:
00021         static void setUTF8() {
00022             #ifndef _WIN32
00023                 // SetConsoleOutputCP returns 0 if there's an error
00024                 if ((SetConsoleOutputCP(CP_UTF8) == 0) || (SetConsoleCP(CP_UTF8) == 0)) {
00025                     std::cout << "Ocorreu um erro ao configurar o terminal do Windows para UTF-8." <<
std::endl;
00026                     std::cout << "A aplicação irá continuar. Desformatação será visível. Para resolver,
reinicie a aplicação." << std::endl;
00027                 }
00028                 setlocale(LC_NUMERIC, "Portuguese"); // Floats now use ',' instead of '.'
00029             #else
00030                 setlocale(LC_ALL, "pt_PT.UTF-8");
00031             #endif
00032         }
00033     };
00034
```

9.21 app/include/tinyxml2.h File Reference

```
#include <cctype>
#include <climits>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <stdint.h>
```

Include dependency graph for tinyxml2.h: This graph shows which files directly or indirectly include this file:

Classes

- class [tinyxml2::StrPair](#)
- class [tinyxml2::DynArray< T, INITIAL_SIZE >](#)
- class [tinyxml2::MemPool](#)
- class [tinyxml2::MemPoolT< ITEM_SIZE >](#)
- class [tinyxml2::XMLVisitor](#)
- class [tinyxml2::XMLUtil](#)
- class [tinyxml2::XMLNode](#)
- class [tinyxml2::XMLText](#)
- class [tinyxml2::XMLComment](#)
- class [tinyxml2::XMLDeclaration](#)
- class [tinyxml2::XMLUnknown](#)
- class [tinyxml2::XMLAttribute](#)
- class [tinyxml2::XMLElement](#)
- class [tinyxml2::XMLDocument](#)
- class [tinyxml2::XMLHandle](#)
- class [tinyxml2::XMLConstHandle](#)
- class [tinyxml2::XMLPrinter](#)

Namespaces

- namespace [tinyxml2](#)

Macros

- #define [TINYXML2_LIB](#)
- #define [TIXMLASSERT\(x\)](#)
- #define [TINYXML2_MAJOR_VERSION](#) 11
- #define [TINYXML2_MINOR_VERSION](#) 0
- #define [TINYXML2_PATCH_VERSION](#) 0

Enumerations

- enum [tinyxml2::XMLError](#) {
[tinyxml2::XML_SUCCESS](#) = 0 , [tinyxml2::XML_NO_ATTRIBUTE](#) , [tinyxml2::XML_WRONG_ATTRIBUTE_TYPE](#) ,
[tinyxml2::XML_ERROR_FILE_NOT_FOUND](#) ,
[tinyxml2::XML_ERROR_FILE_COULD_NOT_BE_OPENED](#) , [tinyxml2::XML_ERROR_FILE_READ_ERROR](#) ,
[tinyxml2::XML_ERROR_PARSING_ELEMENT](#) , [tinyxml2::XML_ERROR_PARSING_ATTRIBUTE](#) ,
[tinyxml2::XML_ERROR_PARSING_TEXT](#) , [tinyxml2::XML_ERROR_PARSING_CDATA](#) , [tinyxml2::XML_ERROR_PARSING_COMMENT](#) ,
[tinyxml2::XML_ERROR_PARSING_DECLARATION](#) ,
[tinyxml2::XML_ERROR_PARSING_UNKNOWN](#) , [tinyxml2::XML_ERROR_EMPTY_DOCUMENT](#) , [tinyxml2::XML_ERROR_MISPLACED_XML_PI](#) ,
[tinyxml2::XML_ERROR_PARSING](#) ,
[tinyxml2::XML_CAN_NOT_CONVERT_TEXT](#) , [tinyxml2::XML_NO_TEXT_NODE](#) , [tinyxml2::XML_ELEMENT_DEPTH_EXCEEDED](#) ,
[tinyxml2::XML_ERROR_COUNT](#) }
- enum [tinyxml2::Whitespace](#) { [tinyxml2::PRESERVE_WHITESPACE](#) , [tinyxml2::COLLAPSE_WHITESPACE](#) ,
[tinyxml2::PEDANTIC_WHITESPACE](#) }

9.21.1 Macro Definition Documentation

9.21.1.1 TINYXML2_LIB

```
#define TINYXML2_LIB
```

9.21.1.2 TINYXML2_MAJOR_VERSION

```
#define TINYXML2_MAJOR_VERSION 11
```

9.21.1.3 TINYXML2_MINOR_VERSION

```
#define TINYXML2_MINOR_VERSION 0
```

9.21.1.4 TINYXML2_PATCH_VERSION

```
#define TINYXML2_PATCH_VERSION 0
```

9.21.1.5 TIXMLASSERT

```
#define TIXMLASSERT(  
    x)
```

Value:

```
do {} while(false)
```

9.22 tinyxml2.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 Original code by Lee Thomason (www.grinninglizard.com)
00003
00004 This software is provided 'as-is', without any express or implied
00005 warranty. In no event will the authors be held liable for any
00006 damages arising from the use of this software.
00007
00008 Permission is granted to anyone to use this software for any
00009 purpose, including commercial applications, and to alter it and
00010 redistribute it freely, subject to the following restrictions:
00011
00012 1. The origin of this software must not be misrepresented; you must
00013 not claim that you wrote the original software. If you use this
00014 software in a product, an acknowledgment in the product documentation
00015 would be appreciated but is not required.
00016
00017 2. Altered source versions must be plainly marked as such, and
00018 must not be misrepresented as being the original software.
00019
00020 3. This notice may not be removed or altered from any source
00021 distribution.
00022 */
00023
00024 #ifndef TINYXML2_INCLUDED
00025 #define TINYXML2_INCLUDED
00026
00027 #if defined(ANDROID_NDK) || defined(__BORLANDC__) || defined(__QNXNTO__)
00028 #    include <ctype.h>
```

```

00029 # include <limits.h>
00030 # include <stdio.h>
00031 # include <stdlib.h>
00032 # include <string.h>
00033 # if defined(__PS3__)
00034 #     include <stddef.h>
00035 # endif
00036 #else
00037 #     include <cctype>
00038 #     include <climits>
00039 #     include <cstdio>
00040 #     include <stdlib.h>
00041 #     include <cstring>
00042 #endif
00043 #include <stdint.h>
00044
00045 /*
00046     gcc:
00047         g++ -Wall -DTINYXML2_DEBUG tinyxml2.cpp xmltest.cpp -o gccxmltest.exe
00048
00049     Formatting, Artistic Style:
00050         AStyle.exe --style=1tbs --indent-switches --break-closing-brackets --indent-preprocessor
00051         tinyxml2.cpp tinyxml2.h
00052 */
00053 #if defined( _DEBUG ) || defined ( __DEBUG__ )
00054 #     ifndef TINYXML2_DEBUG
00055 #         define TINYXML2_DEBUG
00056 #     endif
00057 #endif
00058
00059 #ifdef _MSC_VER
00060 #     pragma warning(push)
00061 #     pragma warning(disable: 4251)
00062 #endif
00063
00064 #ifdef _MSC_VER
00065 #     ifdef TINYXML2_EXPORT
00066 #         define TINYXML2_LIB __declspec(dllexport)
00067 #     elif defined(TINYXML2_IMPORT)
00068 #         define TINYXML2_LIB __declspec(dllimport)
00069 #     else
00070 #         define TINYXML2_LIB
00071 #     endif
00072 #elif __GNUC__ >= 4
00073 #     define TINYXML2_LIB __attribute__((visibility("default")))
00074 #else
00075 #     define TINYXML2_LIB
00076 #endif
00077
00078
00079 #if !defined(TIXMLASSERT)
00080 #if defined(TINYXML2_DEBUG)
00081 #     if defined(_MSC_VER)
00082 #         // "(void)0," is for suppressing C4127 warning in "assert(false)", "assert(true)" and the like
00083 #         define TIXMLASSERT( x )    do { if ( !(void)0,(x)) { __debugbreak(); } } while(false)
00084 #     elif defined (ANDROID_NDK)
00085 #         include <android/log.h>
00086 #         define TIXMLASSERT( x )    do { if ( !(x)) { __android_log_assert( "assert", "grinliz",
"ASSERT in '%s' at %d.", __FILE__, __LINE__ ); } } while(false)
00087 #     else
00088 #         include <assert.h>
00089 #         define TIXMLASSERT          assert
00090 #     endif
00091 #else
00092 #     define TIXMLASSERT( x )    do {} while(false)
00093 #endif
00094 #endif
00095
00096 /* Versioning, past 1.0.14:
00097     http://semver.org/
00098 */
00099 static const int TIXML2_MAJOR_VERSION = 11;
00100 static const int TIXML2_MINOR_VERSION = 0;
00101 static const int TIXML2_PATCH_VERSION = 0;
00102
00103 #define TINYXML2_MAJOR_VERSION 11
00104 #define TINYXML2_MINOR_VERSION 0
00105 #define TINYXML2_PATCH_VERSION 0
00106
00107 // A fixed element depth limit is problematic. There needs to be a
00108 // limit to avoid a stack overflow. However, that limit varies per
00109 // system, and the capacity of the stack. On the other hand, it's a trivial
00110 // attack that can result from ill, malicious, or even correctly formed XML,
00111 // so there needs to be a limit in place.
00112 static const int TINYXML2_MAX_ELEMENT_DEPTH = 500;
00113

```

```

00114 namespace tinyxml2
00115 {
00116     class XMLDocument;
00117     class XMLElement;
00118     class XMLAttribute;
00119     class XMLComment;
00120     class XMLText;
00121     class XMLDeclaration;
00122     class XMLUnknown;
00123     class XMLPrinter;
00124
00125     /*
00126     A class that wraps strings. Normally stores the start and end
00127     pointers into the XML file itself, and will apply normalization
00128     and entity translation if actually read. Can also store (and memory
00129     manage) a traditional char[]
00130
00131     Isn't clear why TINYXML2_LIB is needed; but seems to fix #719
00132     */
00133     class TINYXML2_LIB StrPair
00134     {
00135     public:
00136         enum Mode {
00137             NEEDS_ENTITY_PROCESSING      = 0x01,
00138             NEEDS_NEWLINE_NORMALIZATION  = 0x02,
00139             NEEDS_WHITESPACE_COLLAPSING  = 0x04,
00140
00141             TEXT_ELEMENT                  = NEEDS_ENTITY_PROCESSING | NEEDS_NEWLINE_NORMALIZATION,
00142             TEXT_ELEMENT_LEAVE_ENTITIES  = NEEDS_NEWLINE_NORMALIZATION,
00143             ATTRIBUTE_NAME                = 0,
00144             ATTRIBUTE_VALUE               = NEEDS_ENTITY_PROCESSING | NEEDS_NEWLINE_NORMALIZATION,
00145             ATTRIBUTE_VALUE_LEAVE_ENTITIES = NEEDS_NEWLINE_NORMALIZATION,
00146             COMMENT                       = NEEDS_NEWLINE_NORMALIZATION
00147         };
00148
00149         StrPair() : _flags( 0 ), _start( 0 ), _end( 0 ) {}
00150         ~StrPair();
00151
00152         void Set( char* start, char* end, int flags ) {
00153             TIXMLASSERT( start );
00154             TIXMLASSERT( end );
00155             Reset();
00156             _start = start;
00157             _end = end;
00158             _flags = flags | NEEDS_FLUSH;
00159         }
00160
00161         const char* GetStr();
00162
00163         bool Empty() const {
00164             return _start == _end;
00165         }
00166
00167         void SetInternedStr( const char* str ) {
00168             Reset();
00169             _start = const_cast<char*>(str);
00170         }
00171
00172         void SetStr( const char* str, int flags=0 );
00173
00174         char* ParseText( char* in, const char* endTag, int strFlags, int* curLineNumPtr );
00175         char* ParseName( char* in );
00176
00177         void TransferTo( StrPair* other );
00178         void Reset();
00179
00180     private:
00181         void CollapseWhitespace();
00182
00183         enum {
00184             NEEDS_FLUSH = 0x100,
00185             NEEDS_DELETE = 0x200
00186         };
00187
00188         int _flags;
00189         char* _start;
00190         char* _end;
00191
00192         StrPair( const StrPair& other ); // not supported
00193         void operator=( const StrPair& other ); // not supported, use TransferTo()
00194     };
00195
00196
00197     /*
00198     A dynamic array of Plain Old Data. Doesn't support constructors, etc.
00199     Has a small initial memory pool, so that low or no usage will not
00200     cause a call to new/delete

```

```

00201 */
00202 template <class T, size_t INITIAL_SIZE>
00203 class DynArray
00204 {
00205 public:
00206     DynArray() :
00207         _mem( _pool ),
00208         _allocated( INITIAL_SIZE ),
00209         _size( 0 )
00210     {
00211     }
00212
00213     ~DynArray() {
00214         if ( _mem != _pool ) {
00215             delete [] _mem;
00216         }
00217     }
00218
00219     void Clear() {
00220         _size = 0;
00221     }
00222
00223     void Push( T t ) {
00224         TIXMLASSERT( _size < INT_MAX );
00225         EnsureCapacity( _size+1 );
00226         _mem[_size] = t;
00227         ++_size;
00228     }
00229
00230     T* PushArr( size_t count ) {
00231         TIXMLASSERT( _size <= SIZE_MAX - count );
00232         EnsureCapacity( _size+count );
00233         T* ret = &_mem[_size];
00234         _size += count;
00235         return ret;
00236     }
00237
00238     T Pop() {
00239         TIXMLASSERT( _size > 0 );
00240         --_size;
00241         return _mem[_size];
00242     }
00243
00244     void PopArr( size_t count ) {
00245         TIXMLASSERT( _size >= count );
00246         _size -= count;
00247     }
00248
00249     bool Empty() const {
00250         return _size == 0;
00251     }
00252
00253     T& operator[](size_t i) {
00254         TIXMLASSERT( i < _size );
00255         return _mem[i];
00256     }
00257
00258     const T& operator[](size_t i) const {
00259         TIXMLASSERT( i < _size );
00260         return _mem[i];
00261     }
00262
00263     const T& PeekTop() const {
00264         TIXMLASSERT( _size > 0 );
00265         return _mem[ _size - 1 ];
00266     }
00267
00268     size_t Size() const {
00269         return _size;
00270     }
00271
00272     size_t Capacity() const {
00273         TIXMLASSERT( _allocated >= INITIAL_SIZE );
00274         return _allocated;
00275     }
00276
00277     void SwapRemove( size_t i ) {
00278         TIXMLASSERT( i < _size );
00279         TIXMLASSERT( _size > 0 );
00280         _mem[i] = _mem[_size - 1];
00281         --_size;
00282     }
00283
00284     const T* Mem() const {
00285         TIXMLASSERT( _mem );
00286         return _mem;
00287     }

```

```

00288
00289     T* Mem() {
00290         TIXMLASSERT( !_mem );
00291         return _mem;
00292     }
00293
00294 private:
00295     DynArray( const DynArray& ); // not supported
00296     void operator=( const DynArray& ); // not supported
00297
00298     void EnsureCapacity( size_t cap ) {
00299         TIXMLASSERT( cap > 0 );
00300         if ( cap > _allocated ) {
00301             TIXMLASSERT( cap <= SIZE_MAX / 2 / sizeof(T) );
00302             const size_t newAllocated = cap * 2;
00303             T* newMem = new T[newAllocated];
00304             TIXMLASSERT( newAllocated >= _size );
00305             memcpy( newMem, _mem, sizeof(T) * _size ); // warning: not using constructors, only works
00306         }
00307         delete [] _mem;
00308         _mem = newMem;
00309         _allocated = newAllocated;
00310     }
00311
00312     }
00313
00314     T* _mem;
00315     T _pool[INITIAL_SIZE];
00316     size_t _allocated; // objects allocated
00317     size_t _size;      // number objects in use
00318 };
00319
00320
00321 /*
00322     Parent virtual class of a pool for fast allocation
00323     and deallocation of objects.
00324 */
00325 class MemPool
00326 {
00327 public:
00328     MemPool() {}
00329     virtual ~MemPool() {}
00330
00331     virtual size_t ItemSize() const = 0;
00332     virtual void* Alloc() = 0;
00333     virtual void Free( void* ) = 0;
00334     virtual void SetTracked() = 0;
00335 };
00336
00337
00338 /*
00339     Template child class to create pools of the correct type.
00340 */
00341 template< size_t ITEM_SIZE >
00342 class MemPoolT : public MemPool
00343 {
00344 public:
00345     MemPoolT() : _blockPtrs(), _root(0), _currentAllocs(0), _nAllocs(0), _maxAllocs(0), _nUntracked(0)
00346     {}
00347     ~MemPoolT() {
00348         MemPoolT< ITEM_SIZE >::Clear();
00349     }
00350
00351     void Clear() {
00352         // Delete the blocks.
00353         while( !_blockPtrs.Empty() ) {
00354             Block* lastBlock = _blockPtrs.Pop();
00355             delete lastBlock;
00356         }
00357         _root = 0;
00358         _currentAllocs = 0;
00359         _nAllocs = 0;
00360         _maxAllocs = 0;
00361         _nUntracked = 0;
00362     }
00363
00364     virtual size_t ItemSize() const override {
00365         return ITEM_SIZE;
00366     }
00367
00368     size_t CurrentAllocs() const {
00369         return _currentAllocs;
00370     }
00371
00372     virtual void* Alloc() override {
00373         if ( !_root ) {
00374             // Need a new block.

```

```

00373         Block* block = new Block;
00374         _blockPtrs.Push( block );
00375
00376         Item* blockItems = block->items;
00377         for( size_t i = 0; i < ITEMS_PER_BLOCK - 1; ++i ) {
00378             blockItems[i].next = &(blockItems[i + 1]);
00379         }
00380         blockItems[ITEMS_PER_BLOCK - 1].next = 0;
00381         _root = blockItems;
00382     }
00383     Item* const result = _root;
00384     TIXMLASSERT( result != 0 );
00385     _root = _root->next;
00386
00387     ++_currentAllocs;
00388     if ( _currentAllocs > _maxAllocs ) {
00389         _maxAllocs = _currentAllocs;
00390     }
00391     ++_nAllocs;
00392     ++_nUntracked;
00393     return result;
00394 }
00395
00396 virtual void Free( void* mem ) override {
00397     if ( !mem ) {
00398         return;
00399     }
00400     --_currentAllocs;
00401     Item* item = static_cast<Item*>( mem );
00402 #ifdef TINYXML2_DEBUG
00403     memset( item, 0xfe, sizeof( *item ) );
00404 #endif
00405     item->next = _root;
00406     _root = item;
00407 }
00408 void Trace( const char* name ) {
00409     printf( "Mempool %s watermark=%d [%dk] current=%d size=%d nAlloc=%d blocks=%d\n",
00410         name, _maxAllocs, _maxAllocs * ITEM_SIZE / 1024, _currentAllocs,
00411         ITEM_SIZE, _nAllocs, _blockPtrs.Size() );
00412 }
00413
00414 void SetTracked() override {
00415     --_nUntracked;
00416 }
00417
00418 size_t Untracked() const {
00419     return _nUntracked;
00420 }
00421
00422 // This number is perf sensitive. 4k seems like a good tradeoff on my machine.
00423 // The test file is large, 170k.
00424 // Release:      VS2010 gcc(no opt)
00425 //      1k:      4000
00426 //      2k:      4000
00427 //      4k:      3900      21000
00428 //      16k:     5200
00429 //      32k:     4300
00430 //      64k:     4000      21000
00431 // Declared public because some compilers do not accept to use ITEMS_PER_BLOCK
00432 // in private part if ITEMS_PER_BLOCK is private
00433 enum { ITEMS_PER_BLOCK = (4 * 1024) / ITEM_SIZE };
00434
00435 private:
00436     MemPoolT( const MemPoolT& ); // not supported
00437     void operator=( const MemPoolT& ); // not supported
00438
00439     union Item {
00440         Item* next;
00441         char itemData[static_cast<size_t>(ITEM_SIZE)];
00442     };
00443     struct Block {
00444         Item items[ITEMS_PER_BLOCK];
00445     };
00446     DynArray< Block*, 10 > _blockPtrs;
00447     Item* _root;
00448
00449     size_t _currentAllocs;
00450     size_t _nAllocs;
00451     size_t _maxAllocs;
00452     size_t _nUntracked;
00453 };
00454
00455
00456
00457 class TINYXML2_LIB XMLVisitor
00458 {
00459 public:

```

```

00479     virtual ~XMLVisitor() {}
00480
00482     virtual bool VisitEnter( const XMLDocument& /*doc*/ )          {
00483         return true;
00484     }
00486     virtual bool VisitExit( const XMLDocument& /*doc*/ )          {
00487         return true;
00488     }
00489
00491     virtual bool VisitEnter( const XMLElement& /*element*/, const XMLAttribute* /*firstAttribute*/ )
00492     {
00493         return true;
00494     }
00495     virtual bool VisitExit( const XMLElement& /*element*/ )        {
00496         return true;
00497     }
00498
00500     virtual bool Visit( const XMLDeclaration& /*declaration*/ )    {
00501         return true;
00502     }
00504     virtual bool Visit( const XMLText& /*text*/ )                  {
00505         return true;
00506     }
00508     virtual bool Visit( const XMLComment& /*comment*/ )            {
00509         return true;
00510     }
00512     virtual bool Visit( const XMLUnknown& /*unknown*/ )            {
00513         return true;
00514     }
00515 };
00516
00517 // WARNING: must match XMLDocument::_errorNames[]
00518 enum XMLError {
00519     XML_SUCCESS = 0,
00520     XML_NO_ATTRIBUTE,
00521     XML_WRONG_ATTRIBUTE_TYPE,
00522     XML_ERROR_FILE_NOT_FOUND,
00523     XML_ERROR_FILE_COULD_NOT_BE_OPENED,
00524     XML_ERROR_FILE_READ_ERROR,
00525     XML_ERROR_PARSING_ELEMENT,
00526     XML_ERROR_PARSING_ATTRIBUTE,
00527     XML_ERROR_PARSING_TEXT,
00528     XML_ERROR_PARSING_CDATA,
00529     XML_ERROR_PARSING_COMMENT,
00530     XML_ERROR_PARSING_DECLARATION,
00531     XML_ERROR_PARSING_UNKNOWN,
00532     XML_ERROR_EMPTY_DOCUMENT,
00533     XML_ERROR_MISMATCHED_ELEMENT,
00534     XML_ERROR_PARSING,
00535     XML_CAN_NOT_CONVERT_TEXT,
00536     XML_NO_TEXT_NODE,
00537     XML_ELEMENT_DEPTH_EXCEEDED,
00538
00539     XML_ERROR_COUNT
00540 };
00541
00542
00543 /*
00544     Utility functionality.
00545 */
00546 class TINYXML2_LIB XMLUtil
00547 {
00548 public:
00549     static const char* SkipWhiteSpace( const char* p, int* curLineNumPtr ) {
00550         TIXMLASSERT( p );
00551
00552         while( IsWhiteSpace(*p) ) {
00553             if (curLineNumPtr && *p == '\n') {
00554                 ++(*curLineNumPtr);
00555             }
00556             ++p;
00557         }
00558         TIXMLASSERT( p );
00559         return p;
00560     }
00561     static char* SkipWhiteSpace( char* const p, int* curLineNumPtr ) {
00562         return const_cast<char*>( SkipWhiteSpace( const_cast<const char*>(p), curLineNumPtr ) );
00563     }
00564
00565     // Anything in the high order range of UTF-8 is assumed to not be whitespace. This isn't
00566     // correct, but simple, and usually works.
00567     static bool IsWhiteSpace( char p ) {
00568         return !IsUTF8Continuation(p) && isspace( static_cast<unsigned char>(p) );
00569     }
00570
00571     inline static bool IsNameStartChar( unsigned char ch ) {
00572         if ( ch >= 128 ) {

```

```

00573         // This is a heuristic guess in attempt to not implement Unicode-aware isalpha()
00574         return true;
00575     }
00576     if ( isalpha( ch ) ) {
00577         return true;
00578     }
00579     return ch == ':' || ch == '_';
00580 }
00581
00582 inline static bool IsNameChar( unsigned char ch ) {
00583     return IsNameStartChar( ch )
00584         || isdigit( ch )
00585         || ch == '.'
00586         || ch == '-';
00587 }
00588
00589 inline static bool IsPrefixHex( const char* p ) {
00590     p = SkipWhiteSpace(p, 0);
00591     return p && *p == '0' && ( *(p + 1) == 'x' || *(p + 1) == 'X' );
00592 }
00593
00594 inline static bool StringEqual( const char* p, const char* q, int nChar=INT_MAX ) {
00595     if ( p == q ) {
00596         return true;
00597     }
00598     TIXMLASSERT( p );
00599     TIXMLASSERT( q );
00600     TIXMLASSERT( nChar >= 0 );
00601     return strncmp( p, q, static_cast<size_t>(nChar) ) == 0;
00602 }
00603
00604 inline static bool IsUTF8Continuation( const char p ) {
00605     return ( p & 0x80 ) != 0;
00606 }
00607
00608 static const char* ReadBOM( const char* p, bool* hasBOM );
00609 // p is the starting location,
00610 // the UTF-8 value of the entity will be placed in value, and length filled in.
00611 static const char* GetCharacterRef( const char* p, char* value, int* length );
00612 static void ConvertUTF32ToUTF8( unsigned long input, char* output, int* length );
00613
00614 // converts primitive types to strings
00615 static void ToStr( int v, char* buffer, int bufferSize );
00616 static void ToStr( unsigned v, char* buffer, int bufferSize );
00617 static void ToStr( bool v, char* buffer, int bufferSize );
00618 static void ToStr( float v, char* buffer, int bufferSize );
00619 static void ToStr( double v, char* buffer, int bufferSize );
00620 static void ToStr(int64_t v, char* buffer, int bufferSize);
00621 static void ToStr(uint64_t v, char* buffer, int bufferSize);
00622
00623 // converts strings to primitive types
00624 static bool ToInt( const char* str, int* value );
00625 static bool ToUnsigned( const char* str, unsigned* value );
00626 static bool ToBool( const char* str, bool* value );
00627 static bool ToFloat( const char* str, float* value );
00628 static bool ToDouble( const char* str, double* value );
00629 static bool ToInt64(const char* str, int64_t* value);
00630 static bool ToUnsigned64(const char* str, uint64_t* value);
00631 // Changes what is serialized for a boolean value.
00632 // Default to "true" and "false". Shouldn't be changed
00633 // unless you have a special testing or compatibility need.
00634 // Be careful: static, global, & not thread safe.
00635 // Be sure to set static const memory as parameters.
00636 static void SetBoolSerialization(const char* writeTrue, const char* writeFalse);
00637
00638 private:
00639     static const char* writeBoolTrue;
00640     static const char* writeBoolFalse;
00641 };
00642
00643
00644 class TINYXML2_LIB XMLNode
00645 {
00646     friend class XMLDocument;
00647     friend class XMLElement;
00648 public:
00649     const XMLDocument* GetDocument() const {
00650         TIXMLASSERT( _document );
00651         return _document;
00652     }
00653     XMLDocument* GetDocument() {
00654         TIXMLASSERT( _document );
00655         return _document;
00656     }
00657     virtual XMLElement* ToElement() {

```

```

00688         return 0;
00689     }
00691     virtual XMLText*      ToText()      {
00692         return 0;
00693     }
00695     virtual XMLComment*   ToComment()   {
00696         return 0;
00697     }
00699     virtual XMLDocument*  ToDocument()  {
00700         return 0;
00701     }
00703     virtual XMLDeclaration* ToDeclaration() {
00704         return 0;
00705     }
00707     virtual XMLUnknown*   ToUnknown()   {
00708         return 0;
00709     }
00710
00711     virtual const XMLElement*      ToElement() const {
00712         return 0;
00713     }
00714     virtual const XMLText*          ToText() const {
00715         return 0;
00716     }
00717     virtual const XMLComment*       ToComment() const {
00718         return 0;
00719     }
00720     virtual const XMLDocument*      ToDocument() const {
00721         return 0;
00722     }
00723     virtual const XMLDeclaration*   ToDeclaration() const {
00724         return 0;
00725     }
00726     virtual const XMLUnknown*       ToUnknown() const {
00727         return 0;
00728     }
00729
00730     // ChildElementCount was originally suggested by msteiger on the sourceforge page for TinyXML and
    modified by KB1SPH for TinyXML-2.
00731
00732     int ChildElementCount(const char *value) const;
00733
00734     int ChildElementCount() const;
00735
00736     const char* Value() const;
00745
00746     void SetValue( const char* val, bool staticMem=false );
00751
00753     int GetLineNum() const { return _parseLineNum; }
00754
00756     const XMLNode* Parent() const {
00757         return _parent;
00758     }
00759
00760     XMLNode* Parent() {
00761         return _parent;
00762     }
00763
00765     bool NoChildren() const {
00766         return !_firstChild;
00767     }
00768
00770     const XMLNode* FirstChild() const {
00771         return _firstChild;
00772     }
00773
00774     XMLNode* FirstChild() {
00775         return _firstChild;
00776     }
00777
00781     const XMLElement* FirstChildElement( const char* name = 0 ) const;
00782
00783     XMLElement* FirstChildElement( const char* name = 0 ) {
00784         return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->FirstChildElement( name ));
00785     }
00786
00788     const XMLNode* LastChild() const {
00789         return _lastChild;
00790     }
00791
00792     XMLNode* LastChild() {
00793         return _lastChild;
00794     }
00795
00799     const XMLElement* LastChildElement( const char* name = 0 ) const;
00800
00801     XMLElement* LastChildElement( const char* name = 0 ) {

```

```

00802         return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->LastChildElement(name) );
00803     }
00804
00806     const XMLNode* PreviousSibling() const {
00807         return _prev;
00808     }
00809
00810     XMLNode* PreviousSibling() {
00811         return _prev;
00812     }
00813
00815     const XMLElement* PreviousSiblingElement( const char* name = 0 ) const ;
00816
00817     XMLElement* PreviousSiblingElement( const char* name = 0 ) {
00818         return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->PreviousSiblingElement( name
    ) );
00819     }
00820
00822     const XMLNode* NextSibling() const {
00823         return _next;
00824     }
00825
00826     XMLNode* NextSibling() {
00827         return _next;
00828     }
00829
00831     const XMLElement* NextSiblingElement( const char* name = 0 ) const;
00832
00833     XMLElement* NextSiblingElement( const char* name = 0 ) {
00834         return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->NextSiblingElement( name ) );
00835     }
00836
00844     XMLNode* InsertEndChild( XMLNode* addThis );
00845
00846     XMLNode* LinkEndChild( XMLNode* addThis ) {
00847         return InsertEndChild( addThis );
00848     }
00856     XMLNode* InsertFirstChild( XMLNode* addThis );
00865     XMLNode* InsertAfterChild( XMLNode* afterThis, XMLNode* addThis );
00866
00870     void DeleteChildren();
00871
00875     void DeleteChild( XMLNode* node );
00876
00886     virtual XMLNode* ShallowClone( XMLDocument* document ) const = 0;
00887
00901     XMLNode* DeepClone( XMLDocument* target ) const;
00902
00909     virtual bool ShallowEqual( const XMLNode* compare ) const = 0;
00910
00933     virtual bool Accept( XMLVisitor* visitor ) const = 0;
00934
00940     void SetUserData(void* userData) { _userData = userData; }
00941
00947     void* GetUserData() const { return _userData; }
00948
00949 protected:
00950     explicit XMLNode( XMLDocument* );
00951     virtual ~XMLNode();
00952
00953     virtual char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr);
00954
00955     XMLDocument* _document;
00956     XMLNode* _parent;
00957     mutable StrPair _value;
00958     int _parseLineNum;
00959
00960     XMLNode* _firstChild;
00961     XMLNode* _lastChild;
00962
00963     XMLNode* _prev;
00964     XMLNode* _next;
00965
00966     void* _userData;
00967
00968 private:
00969     MemPool* _memPool;
00970     void Unlink( XMLNode* child );
00971     static void DeleteNode( XMLNode* node );
00972     void InsertChildPreamble( XMLNode* insertThis ) const;
00973     const XMLElement* ToElementWithName( const char* name ) const;
00974
00975     XMLNode( const XMLNode& ); // not supported
00976     XMLNode& operator=( const XMLNode& ); // not supported
00977 };
00978
00979

```

```

00992 class TINYXML2_LIB XMLText : public XMLNode
00993 {
00994     friend class XMLDocument;
00995 public:
00996     virtual bool Accept( XMLVisitor* visitor ) const override;
00997
00998     virtual XMLText* ToText() override {
00999         return this;
01000     }
01001     virtual const XMLText* ToText() const override {
01002         return this;
01003     }
01004
01006     void SetCDATA( bool isCDATA ) {
01007         _isCDATA = isCDATA;
01008     }
01010     bool CDATA() const {
01011         return _isCDATA;
01012     }
01013
01014     virtual XMLNode* ShallowClone( XMLDocument* document ) const override;
01015     virtual bool ShallowEqual( const XMLNode* compare ) const override;
01016
01017 protected:
01018     explicit XMLText( XMLDocument* doc ) : XMLNode( doc ), _isCDATA( false ) {}
01019     virtual ~XMLText() {}
01020
01021     char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr ) override;
01022
01023 private:
01024     bool _isCDATA;
01025
01026     XMLText( const XMLText& ); // not supported
01027     XMLText& operator=( const XMLText& ); // not supported
01028 };
01029
01030
01032 class TINYXML2_LIB XMLComment : public XMLNode
01033 {
01034     friend class XMLDocument;
01035 public:
01036     virtual XMLComment* ToComment() override {
01037         return this;
01038     }
01039     virtual const XMLComment* ToComment() const override {
01040         return this;
01041     }
01042
01043     virtual bool Accept( XMLVisitor* visitor ) const override;
01044
01045     virtual XMLNode* ShallowClone( XMLDocument* document ) const override;
01046     virtual bool ShallowEqual( const XMLNode* compare ) const override;
01047
01048 protected:
01049     explicit XMLComment( XMLDocument* doc );
01050     virtual ~XMLComment();
01051
01052     char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr ) override;
01053
01054 private:
01055     XMLComment( const XMLComment& ); // not supported
01056     XMLComment& operator=( const XMLComment& ); // not supported
01057 };
01058
01059
01071 class TINYXML2_LIB XMLDeclaration : public XMLNode
01072 {
01073     friend class XMLDocument;
01074 public:
01075     virtual XMLDeclaration* ToDeclaration() override {
01076         return this;
01077     }
01078     virtual const XMLDeclaration* ToDeclaration() const override {
01079         return this;
01080     }
01081
01082     virtual bool Accept( XMLVisitor* visitor ) const override;
01083
01084     virtual XMLNode* ShallowClone( XMLDocument* document ) const override;
01085     virtual bool ShallowEqual( const XMLNode* compare ) const override;
01086
01087 protected:
01088     explicit XMLDeclaration( XMLDocument* doc );
01089     virtual ~XMLDeclaration();
01090
01091     char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr ) override;
01092

```

```

01093 private:
01094     XMLDeclaration( const XMLDeclaration& );    // not supported
01095     XMLDeclaration& operator=( const XMLDeclaration& ); // not supported
01096 };
01097
01098
01106 class TINYXML2_LIB XMLUnknown : public XMLNode
01107 {
01108     friend class XMLDocument;
01109 public:
01110     virtual XMLUnknown* ToUnknown() override      {
01111         return this;
01112     }
01113     virtual const XMLUnknown* ToUnknown() const override {
01114         return this;
01115     }
01116
01117     virtual bool Accept( XMLVisitor* visitor ) const override;
01118
01119     virtual XMLNode* ShallowClone( XMLDocument* document ) const override;
01120     virtual bool ShallowEqual( const XMLNode* compare ) const override;
01121
01122 protected:
01123     explicit XMLUnknown( XMLDocument* doc );
01124     virtual ~XMLUnknown();
01125
01126     char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr ) override;
01127
01128 private:
01129     XMLUnknown( const XMLUnknown& );    // not supported
01130     XMLUnknown& operator=( const XMLUnknown& ); // not supported
01131 };
01132
01133
01141 class TINYXML2_LIB XMLAttribute
01142 {
01143     friend class XMLElement;
01144 public:
01145     const char* Name() const;
01146
01147     const char* Value() const;
01148
01149     int GetLineNum() const { return _parseLineNum; }
01150
01151     const XMLAttribute* Next() const {
01152         return _next;
01153     }
01154
01155     int IntValue() const {
01156         int i = 0;
01157         QueryIntValue(&i);
01158         return i;
01159     }
01160
01161     int64_t Int64Value() const {
01162         int64_t i = 0;
01163         QueryInt64Value(&i);
01164         return i;
01165     }
01166
01167     uint64_t Unsigned64Value() const {
01168         uint64_t i = 0;
01169         QueryUnsigned64Value(&i);
01170         return i;
01171     }
01172
01173     unsigned UnsignedValue() const {
01174         unsigned i=0;
01175         QueryUnsignedValue( &i );
01176         return i;
01177     }
01178
01179     bool BoolValue() const {
01180         bool b=false;
01181         QueryBoolValue( &b );
01182         return b;
01183     }
01184
01185     double DoubleValue() const {
01186         double d=0;
01187         QueryDoubleValue( &d );
01188         return d;
01189     }
01190
01191     float FloatValue() const {
01192         float f=0;
01193         QueryFloatValue( &f );
01194         return f;
01195     }
01196
01197 }

```

```

01205
01210 XMLError QueryIntValue( int* value ) const;
01212 XMLError QueryUnsignedValue( unsigned int* value ) const;
01214 XMLError QueryInt64Value( int64_t* value ) const;
01216 XMLError QueryUnsigned64Value( uint64_t* value ) const;
01218 XMLError QueryBoolValue( bool* value ) const;
01220 XMLError QueryDoubleValue( double* value ) const;
01222 XMLError QueryFloatValue( float* value ) const;
01223
01225 void SetAttribute( const char* value );
01227 void SetAttribute( int value );
01229 void SetAttribute( unsigned value );
01231 void SetAttribute( int64_t value );
01233 void SetAttribute( uint64_t value );
01235 void SetAttribute( bool value );
01237 void SetAttribute( double value );
01239 void SetAttribute( float value );
01240
01241 private:
01242     enum { BUF_SIZE = 200 };
01243
01244     XMLAttribute() : _name(), _value(), _parseLineNum( 0 ), _next( 0 ), _memPool( 0 ) {}
01245     virtual ~XMLAttribute() {}
01246
01247     XMLAttribute( const XMLAttribute& ); // not supported
01248     void operator=( const XMLAttribute& ); // not supported
01249     void SetName( const char* name );
01250
01251     char* ParseDeep( char* p, bool processEntities, int* curLineNumPtr );
01252
01253     mutable StrPair _name;
01254     mutable StrPair _value;
01255     int _parseLineNum;
01256     XMLAttribute* _next;
01257     MemPool* _memPool;
01258 };
01259
01260
01265 class TINYXML2_LIB XMLElement : public XMLNode
01266 {
01267     friend class XMLDocument;
01268 public:
01270     const char* Name() const {
01271         return Value();
01272     }
01274     void SetName( const char* str, bool staticMem=false ) {
01275         SetValue( str, staticMem );
01276     }
01277
01278     virtual XMLElement* ToElement() override {
01279         return this;
01280     }
01281     virtual const XMLElement* ToElement() const override {
01282         return this;
01283     }
01284     virtual bool Accept( XMLVisitor* visitor ) const override;
01285
01309     const char* Attribute( const char* name, const char* value=0 ) const;
01310
01317     int IntAttribute( const char* name, int defaultValue = 0 ) const;
01319     unsigned UnsignedAttribute( const char* name, unsigned defaultValue = 0 ) const;
01321     int64_t Int64Attribute( const char* name, int64_t defaultValue = 0 ) const;
01323     uint64_t Unsigned64Attribute( const char* name, uint64_t defaultValue = 0 ) const;
01325     bool BoolAttribute( const char* name, bool defaultValue = false ) const;
01327     double DoubleAttribute( const char* name, double defaultValue = 0 ) const;
01329     float FloatAttribute( const char* name, float defaultValue = 0 ) const;
01330
01344     XMLError QueryIntAttribute( const char* name, int* value ) const {
01345         const XMLAttribute* a = FindAttribute( name );
01346         if ( !a ) {
01347             return XML_NO_ATTRIBUTE;
01348         }
01349         return a->QueryIntValue( value );
01350     }
01351
01353     XMLError QueryUnsignedAttribute( const char* name, unsigned int* value ) const {
01354         const XMLAttribute* a = FindAttribute( name );
01355         if ( !a ) {
01356             return XML_NO_ATTRIBUTE;
01357         }
01358         return a->QueryUnsignedValue( value );
01359     }
01360
01362     XMLError QueryInt64Attribute( const char* name, int64_t* value ) const {
01363         const XMLAttribute* a = FindAttribute( name );
01364         if ( !a ) {
01365             return XML_NO_ATTRIBUTE;

```

```

01366     }
01367     return a->QueryInt64Value(value);
01368 }
01369
01370 XMLError QueryUnsigned64Attribute(const char* name, uint64_t* value) const {
01371     const XMLAttribute* a = FindAttribute(name);
01372     if(!a) {
01373         return XML_NO_ATTRIBUTE;
01374     }
01375     return a->QueryUnsigned64Value(value);
01376 }
01377
01378 XMLError QueryBoolAttribute( const char* name, bool* value ) const {
01379     const XMLAttribute* a = FindAttribute( name );
01380     if ( !a ) {
01381         return XML_NO_ATTRIBUTE;
01382     }
01383     return a->QueryBoolValue( value );
01384 }
01385
01386 XMLError QueryDoubleAttribute( const char* name, double* value ) const {
01387     const XMLAttribute* a = FindAttribute( name );
01388     if ( !a ) {
01389         return XML_NO_ATTRIBUTE;
01390     }
01391     return a->QueryDoubleValue( value );
01392 }
01393
01394 XMLError QueryFloatAttribute( const char* name, float* value ) const {
01395     const XMLAttribute* a = FindAttribute( name );
01396     if ( !a ) {
01397         return XML_NO_ATTRIBUTE;
01398     }
01399     return a->QueryFloatValue( value );
01400 }
01401
01402 XMLError QueryStringAttribute(const char* name, const char** value) const {
01403     const XMLAttribute* a = FindAttribute(name);
01404     if (!a) {
01405         return XML_NO_ATTRIBUTE;
01406     }
01407     *value = a->Value();
01408     return XML_SUCCESS;
01409 }
01410
01411 XMLError QueryAttribute( const char* name, int* value ) const {
01412     return QueryIntAttribute( name, value );
01413 }
01414
01415 XMLError QueryAttribute( const char* name, unsigned int* value ) const {
01416     return QueryUnsignedAttribute( name, value );
01417 }
01418
01419 XMLError QueryAttribute(const char* name, int64_t* value) const {
01420     return QueryInt64Attribute(name, value);
01421 }
01422
01423 XMLError QueryAttribute(const char* name, uint64_t* value) const {
01424     return QueryUnsigned64Attribute(name, value);
01425 }
01426
01427 XMLError QueryAttribute( const char* name, bool* value ) const {
01428     return QueryBoolAttribute( name, value );
01429 }
01430
01431 XMLError QueryAttribute( const char* name, double* value ) const {
01432     return QueryDoubleAttribute( name, value );
01433 }
01434
01435 XMLError QueryAttribute( const char* name, float* value ) const {
01436     return QueryFloatAttribute( name, value );
01437 }
01438
01439 XMLError QueryAttribute(const char* name, const char** value) const {
01440     return QueryStringAttribute(name, value);
01441 }
01442
01443 void SetAttribute( const char* name, const char* value ) {
01444     XMLAttribute* a = FindOrCreateAttribute( name );
01445     a->SetAttribute( value );
01446 }
01447
01448 void SetAttribute( const char* name, int value ) {
01449     XMLAttribute* a = FindOrCreateAttribute( name );
01450     a->SetAttribute( value );
01451 }
01452
01453 void SetAttribute( const char* name, unsigned value ) {
01454     XMLAttribute* a = FindOrCreateAttribute( name );

```

```

01478     a->SetAttribute( value );
01479 }
01480
01482 void SetAttribute(const char* name, int64_t value) {
01483     XMLAttribute* a = FindOrCreateAttribute(name);
01484     a->SetAttribute(value);
01485 }
01486
01488 void SetAttribute(const char* name, uint64_t value) {
01489     XMLAttribute* a = FindOrCreateAttribute(name);
01490     a->SetAttribute(value);
01491 }
01492
01494 void SetAttribute( const char* name, bool value )          {
01495     XMLAttribute* a = FindOrCreateAttribute( name );
01496     a->SetAttribute( value );
01497 }
01499 void SetAttribute( const char* name, double value )      {
01500     XMLAttribute* a = FindOrCreateAttribute( name );
01501     a->SetAttribute( value );
01502 }
01504 void SetAttribute( const char* name, float value )       {
01505     XMLAttribute* a = FindOrCreateAttribute( name );
01506     a->SetAttribute( value );
01507 }
01508
01512 void DeleteAttribute( const char* name );
01513
01515 const XMLAttribute* FirstAttribute() const {
01516     return _rootAttribute;
01517 }
01519 const XMLAttribute* FindAttribute( const char* name ) const;
01520
01549 const char* GetText() const;
01550
01585 void SetText( const char* inText );
01587 void SetText( int value );
01589 void SetText( unsigned value );
01591 void SetText( int64_t value );
01593 void SetText( uint64_t value );
01595 void SetText( bool value );
01597 void SetText( double value );
01599 void SetText( float value );
01600
01627 XMLError QueryIntText( int* ival ) const;
01629 XMLError QueryUnsignedText( unsigned* uval ) const;
01631 XMLError QueryInt64Text( int64_t* uval ) const;
01633 XMLError QueryUnsigned64Text( uint64_t* uval ) const;
01635 XMLError QueryBoolText( bool* bval ) const;
01637 XMLError QueryDoubleText( double* dval ) const;
01639 XMLError QueryFloatText( float* fval ) const;
01640
01641 int IntText( int defaultValue = 0 ) const;
01642
01644 unsigned UnsignedText( unsigned defaultValue = 0 ) const;
01646 int64_t Int64Text( int64_t defaultValue = 0 ) const;
01648 uint64_t Unsigned64Text( uint64_t defaultValue = 0 ) const;
01650 bool BoolText( bool defaultValue = false ) const;
01652 double DoubleText( double defaultValue = 0 ) const;
01654 float FloatText( float defaultValue = 0 ) const;
01655
01660 XMLElement* InsertNewChildElement( const char* name );
01662 XMLComment* InsertNewComment( const char* comment );
01664 XMLText* InsertNewText( const char* text );
01666 XMLDeclaration* InsertNewDeclaration( const char* text );
01668 XMLUnknown* InsertNewUnknown( const char* text );
01669
01670
01671 // internal:
01672 enum ElementClosingType {
01673     OPEN,          // <foo>
01674     CLOSED,        // <foo/>
01675     CLOSING,       // </foo>
01676 };
01677 ElementClosingType ClosingType() const {
01678     return _closingType;
01679 }
01680 virtual XMLNode* ShallowClone( XMLDocument* document ) const override;
01681 virtual bool ShallowEqual( const XMLNode* compare ) const override;
01682
01683 protected:
01684     char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr ) override;
01685
01686 private:
01687     XMLElement( XMLDocument* doc );
01688     virtual ~XMLElement();
01689     XMLElement( const XMLElement& );    // not supported

```

```

01690     void operator=( const XMLElement& );    // not supported
01691
01692     XMLAttribute* FindOrCreateAttribute( const char* name );
01693     char* ParseAttributes( char* p, int* curLineNumPtr );
01694     static void DeleteAttribute( XMLAttribute* attribute );
01695     XMLAttribute* CreateAttribute();
01696
01697     enum { BUF_SIZE = 200 };
01698     ElementClosingType _closingType;
01699     // The attribute list is ordered; there is no 'lastAttribute'
01700     // because the list needs to be scanned for dupes before adding
01701     // a new attribute.
01702     XMLAttribute* _rootAttribute;
01703 };
01704
01705
01706 enum Whitespace {
01707     PRESERVE_WHITESPACE,
01708     COLLAPSE_WHITESPACE,
01709     PEDANTIC_WHITESPACE
01710 };
01711
01712
01713 class TINYXML2_LIB XMLDocument : public XMLNode
01714 {
01715     friend class XMLElement;
01716     // Gives access to SetError and Push/PopDepth, but over-access for everything else.
01717     // Wishing C++ had "internal" scope.
01718     friend class XMLNode;
01719     friend class XMLText;
01720     friend class XMLComment;
01721     friend class XMLDeclaration;
01722     friend class XMLUnknown;
01723 public:
01724     XMLDocument( bool processEntities = true, Whitespace whitespaceMode = PRESERVE_WHITESPACE );
01725     ~XMLDocument();
01726
01727     virtual XMLDocument* ToDocument() override {
01728         TIXMLASSERT( this == _document );
01729         return this;
01730     }
01731     virtual const XMLDocument* ToDocument() const override {
01732         TIXMLASSERT( this == _document );
01733         return this;
01734     }
01735
01736     XMLError Parse( const char* xml, size_t nBytes=static_cast<size_t>(-1) );
01737
01738     XMLError LoadFile( const char* filename );
01739
01740     XMLError LoadFile( FILE* );
01741
01742     XMLError SaveFile( const char* filename, bool compact = false );
01743
01744     XMLError SaveFile( FILE* fp, bool compact = false );
01745
01746     bool ProcessEntities() const {
01747         return _processEntities;
01748     }
01749     Whitespace WhitespaceMode() const {
01750         return _whitespaceMode;
01751     }
01752
01753     bool HasBOM() const {
01754         return _writeBOM;
01755     }
01756     void SetBOM( bool useBOM ) {
01757         _writeBOM = useBOM;
01758     }
01759
01760     XMLElement* RootElement() {
01761         return FirstChildElement();
01762     }
01763     const XMLElement* RootElement() const {
01764         return FirstChildElement();
01765     }
01766
01767     void Print( XMLPrinter* streamer=0 ) const;
01768     virtual bool Accept( XMLVisitor* visitor ) const override;
01769
01770     XMLElement* NewElement( const char* name );
01771     XMLComment* NewComment( const char* comment );
01772     XMLText* NewText( const char* text );
01773     XMLDeclaration* NewDeclaration( const char* text=0 );
01774     XMLUnknown* NewUnknown( const char* text );
01775
01776     void DeleteNode( XMLNode* node );

```

```

01878
01880     void ClearError();
01881
01883     bool Error() const {
01884         return _errorID != XML_SUCCESS;
01885     }
01887     XMLError ErrorID() const {
01888         return _errorID;
01889     }
01890     const char* ErrorName() const;
01891     static const char* ErrorIDToName(XMLError errorID);
01892
01896     const char* ErrorStr() const;
01897
01899     void PrintError() const;
01900
01902     int ErrorLineNum() const
01903     {
01904         return _errorLineNum;
01905     }
01906
01908     void Clear();
01909
01917     void DeepCopy(XMLDocument* target) const;
01918
01919     // internal
01920     char* Identify( char* p, XMLNode** node, bool first );
01921
01922     // internal
01923     void MarkInUse(const XMLNode* const);
01924
01925     virtual XMLNode* ShallowClone( XMLDocument* /*document*/ ) const override{
01926         return 0;
01927     }
01928     virtual bool ShallowEqual( const XMLNode* /*compare*/ ) const override{
01929         return false;
01930     }
01931
01932 private:
01933     XMLDocument( const XMLDocument& ); // not supported
01934     void operator=( const XMLDocument& ); // not supported
01935
01936     bool        _writeBOM;
01937     bool        _processEntities;
01938     XMLError    _errorID;
01939     Whitespace   _whitespaceMode;
01940     mutable StrPair _errorStr;
01941     int         _errorLineNum;
01942     char*       _charBuffer;
01943     int         _parseCurLineNum;
01944     int         _parsingDepth;
01945     // Memory tracking does add some overhead.
01946     // However, the code assumes that you don't
01947     // have a bunch of unlinked nodes around.
01948     // Therefore it takes less memory to track
01949     // in the document vs. a linked list in the XMLNode,
01950     // and the performance is the same.
01951     DynArray<XMLNode*, 10> _unlinked;
01952
01953     MemPoolT< sizeof(XMLElement) > _elementPool;
01954     MemPoolT< sizeof(XMLAttribute) > _attributePool;
01955     MemPoolT< sizeof(XMLText) > _textPool;
01956     MemPoolT< sizeof(XMLComment) > _commentPool;
01957
01958     static const char* _errorNames[XML_ERROR_COUNT];
01959
01960     void Parse();
01961
01962     void SetError( XMLError error, int lineNum, const char* format, ... );
01963
01964     // Something of an obvious security hole, once it was discovered.
01965     // Either an ill-formed XML or an excessively deep one can overflow
01966     // the stack. Track stack depth, and error out if needed.
01967     class DepthTracker {
01968     public:
01969         explicit DepthTracker(XMLDocument * document) {
01970             this->_document = document;
01971             document->PushDepth();
01972         }
01973         ~DepthTracker() {
01974             _document->PopDepth();
01975         }
01976     private:
01977         XMLDocument * _document;
01978     };
01979     void PushDepth();
01980     void PopDepth();

```

```

01981
01982     template<class NodeType, size_t PoolElementSize>
01983     NodeType* CreateUnlinkedNode( MemPoolT<PoolElementSize>& pool );
01984 };
01985
01986 template<class NodeType, size_t PoolElementSize>
01987 inline NodeType* XMLDocument::CreateUnlinkedNode( MemPoolT<PoolElementSize>& pool )
01988 {
01989     TIXMLASSERT( sizeof( NodeType ) == PoolElementSize );
01990     TIXMLASSERT( sizeof( NodeType ) == pool.ItemSize() );
01991     NodeType* returnNode = new (pool.Alloc()) NodeType( this );
01992     TIXMLASSERT( returnNode );
01993     returnNode->_memPool = &pool;
01994
01995     _unlinked.Push(returnNode);
01996     return returnNode;
01997 }
01998
02054 class TINYXML2_LIB XMLHandle
02055 {
02056 public:
02057     explicit XMLHandle( XMLNode* node ) : _node( node ) {
02058     }
02059     explicit XMLHandle( XMLNode& node ) : _node( &node ) {
02060     }
02061     XMLHandle( const XMLHandle& ref ) : _node( ref._node ) {
02062     }
02063     XMLHandle& operator=( const XMLHandle& ref ) {
02064         _node = ref._node;
02065         return *this;
02066     }
02067
02068     XMLHandle FirstChild() {
02069         return XMLHandle( _node ? _node->FirstChild() : 0 );
02070     }
02071     XMLHandle FirstChildElement( const char* name = 0 ) {
02072         return XMLHandle( _node ? _node->FirstChildElement( name ) : 0 );
02073     }
02074     XMLHandle LastChild() {
02075         return XMLHandle( _node ? _node->LastChild() : 0 );
02076     }
02077     XMLHandle LastChildElement( const char* name = 0 ) {
02078         return XMLHandle( _node ? _node->LastChildElement( name ) : 0 );
02079     }
02080     XMLHandle PreviousSibling() {
02081         return XMLHandle( _node ? _node->PreviousSibling() : 0 );
02082     }
02083     XMLHandle PreviousSiblingElement( const char* name = 0 ) {
02084         return XMLHandle( _node ? _node->PreviousSiblingElement( name ) : 0 );
02085     }
02086     XMLHandle NextSibling() {
02087         return XMLHandle( _node ? _node->NextSibling() : 0 );
02088     }
02089     XMLHandle NextSiblingElement( const char* name = 0 ) {
02090         return XMLHandle( _node ? _node->NextSiblingElement( name ) : 0 );
02091     }
02092
02093     XMLNode* ToNode() {
02094         return _node;
02095     }
02096     XMLElement* ToElement() {
02097         return ( _node ? _node->ToElement() : 0 );
02098     }
02099     XMLText* ToText() {
02100         return ( _node ? _node->ToText() : 0 );
02101     }
02102     XMLUnknown* ToUnknown() {
02103         return ( _node ? _node->ToUnknown() : 0 );
02104     }
02105     XMLDeclaration* ToDeclaration() {
02106         return ( _node ? _node->ToDeclaration() : 0 );
02107     }
02108 private:
02109     XMLNode* _node;
02110 };
02111
02135 class TINYXML2_LIB XMLConstHandle
02136 {
02137 public:
02138     explicit XMLConstHandle( const XMLNode* node ) : _node( node ) {
02139     }
02140     explicit XMLConstHandle( const XMLNode& node ) : _node( &node ) {
02141     }
02142     XMLConstHandle( const XMLConstHandle& ref ) : _node( ref._node ) {
02143     }

```

```

02144
02145     XMLConstHandle& operator=( const XMLConstHandle& ref ) {
02146         _node = ref._node;
02147         return *this;
02148     }
02149
02150     const XMLConstHandle FirstChild() const {
02151         return XMLConstHandle( _node ? _node->FirstChild() : 0 );
02152     }
02153     const XMLConstHandle FirstChildElement( const char* name = 0 ) const {
02154         return XMLConstHandle( _node ? _node->FirstChildElement( name ) : 0 );
02155     }
02156     const XMLConstHandle LastChild() const {
02157         return XMLConstHandle( _node ? _node->LastChild() : 0 );
02158     }
02159     const XMLConstHandle LastChildElement( const char* name = 0 ) const {
02160         return XMLConstHandle( _node ? _node->LastChildElement( name ) : 0 );
02161     }
02162     const XMLConstHandle PreviousSibling() const {
02163         return XMLConstHandle( _node ? _node->PreviousSibling() : 0 );
02164     }
02165     const XMLConstHandle PreviousSiblingElement( const char* name = 0 ) const {
02166         return XMLConstHandle( _node ? _node->PreviousSiblingElement( name ) : 0 );
02167     }
02168     const XMLConstHandle NextSibling() const {
02169         return XMLConstHandle( _node ? _node->NextSibling() : 0 );
02170     }
02171     const XMLConstHandle NextSiblingElement( const char* name = 0 ) const {
02172         return XMLConstHandle( _node ? _node->NextSiblingElement( name ) : 0 );
02173     }
02174
02175     const XMLNode* ToNode() const {
02176         return _node;
02177     }
02178     const XMLElement* ToElement() const {
02179         return ( _node ? _node->ToElement() : 0 );
02180     }
02181     const XMLText* ToText() const {
02182         return ( _node ? _node->ToText() : 0 );
02183     }
02184     const XMLUnknown* ToUnknown() const {
02185         return ( _node ? _node->ToUnknown() : 0 );
02186     }
02187     const XMLDeclaration* ToDeclaration() const {
02188         return ( _node ? _node->ToDeclaration() : 0 );
02189     }
02190 }
02191
02192 private:
02193     const XMLNode* _node;
02194 };
02195
02196
02239 class TINYXML2_LIB XMLPrinter : public XMLVisitor
02240 {
02241 public:
02242     enum EscapeAposCharsInAttributes {
02243         ESCAPE_APOS_CHARS_IN_ATTRIBUTES,
02244         DONT_ESCAPE_APOS_CHARS_IN_ATTRIBUTES
02245     };
02246
02253     XMLPrinter( FILE* file=0, bool compact = false, int depth = 0, EscapeAposCharsInAttributes
02254     aposInAttributes = ESCAPE_APOS_CHARS_IN_ATTRIBUTES );
02255     virtual ~XMLPrinter() {}
02256
02257     void PushHeader( bool writeBOM, bool writeDeclaration );
02261     void OpenElement( const char* name, bool compactMode=false );
02263     void PushAttribute( const char* name, const char* value );
02264     void PushAttribute( const char* name, int value );
02265     void PushAttribute( const char* name, unsigned value );
02266     void PushAttribute( const char* name, int64_t value );
02267     void PushAttribute( const char* name, uint64_t value );
02268     void PushAttribute( const char* name, bool value );
02269     void PushAttribute( const char* name, double value );
02271     virtual void CloseElement( bool compactMode=false );
02272
02274     void PushText( const char* text, bool cdata=false );
02276     void PushText( int value );
02278     void PushText( unsigned value );
02280     void PushText( int64_t value );
02282     void PushText( uint64_t value );
02284     void PushText( bool value );
02286     void PushText( float value );
02288     void PushText( double value );
02289
02291     void PushComment( const char* comment );
02292

```

```

02293     void PushDeclaration( const char* value );
02294     void PushUnknown( const char* value );
02295
02296     virtual bool VisitEnter( const XMLDocument& /*doc*/ ) override;
02297     virtual bool VisitExit( const XMLDocument& /*doc*/ ) override {
02298         return true;
02299     }
02300
02301     virtual bool VisitEnter( const XMLElement& element, const XMLAttribute* attribute ) override;
02302     virtual bool VisitExit( const XMLElement& element ) override;
02303
02304     virtual bool Visit( const XMLText& text ) override;
02305     virtual bool Visit( const XMLComment& comment ) override;
02306     virtual bool Visit( const XMLDeclaration& declaration ) override;
02307     virtual bool Visit( const XMLUnknown& unknown ) override;
02308
02313     const char* CStr() const {
02314         return _buffer.Mem();
02315     }
02321     size_t CStrSize() const {
02322         return _buffer.Size();
02323     }
02328     void ClearBuffer( bool resetToFirstElement = true ) {
02329         _buffer.Clear();
02330         _buffer.Push(0);
02331         _firstElement = resetToFirstElement;
02332     }
02333
02334 protected:
02335     virtual bool CompactMode( const XMLElement& ) { return _compactMode; }
02336
02340     virtual void PrintSpace( int depth );
02341     virtual void Print( const char* format, ... );
02342     virtual void Write( const char* data, size_t size );
02343     virtual void Putc( char ch );
02344
02345     inline void Write(const char* data) { Write(data, strlen(data)); }
02346
02347     void SealElementIfJustOpened();
02348     bool _elementJustOpened;
02349     DynArray< const char*, 10 > _stack;
02350
02351 private:
02356     void PrepareForNewNode( bool compactMode );
02357     void PrintString( const char*, bool restrictedEntitySet ); // prints out, after detecting
02358     entities.
02359
02359     bool _firstElement;
02360     FILE* _fp;
02361     int _depth;
02362     int _textDepth;
02363     bool _processEntities;
02364     bool _compactMode;
02365
02366     enum {
02367         ENTITY_RANGE = 64,
02368         BUF_SIZE = 200
02369     };
02370     bool _entityFlag[ENTITY_RANGE];
02371     bool _restrictedEntityFlag[ENTITY_RANGE];
02372
02373     DynArray< char, 20 > _buffer;
02374
02375     // Prohibit cloning, intentionally not implemented
02376     XMLPrinter( const XMLPrinter& );
02377     XMLPrinter& operator=( const XMLPrinter& );
02378 };
02379
02380 } // namespace tinyxml2
02381 } // namespace tinyxml2
02382
02383 #if defined(_MSC_VER)
02384 # pragma warning(pop)
02385 #endif
02386
02387 #endif // TINYXML2_INCLUDED

```

9.23 app/include/utils.hpp File Reference

```

#include <iostream>
#include <string>

```

Include dependency graph for utils.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct [Utils](#)

Functions with complementary use.

9.24 utils.hpp

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include <iostream>
00004 #include <string>
00005
00006
00011 struct Utils {
00019     static std::string zeroPadding(uint16_t value, std::size_t width) {
00020         std::string s = std::to_string(value);
00021         if (s.length() < width) s = std::string(width - s.length(), '0') + s;
00022         return s;
00023     };
00024
00032     static std::string extractAttribute(const std::string& line, const std::string& attr) {
00033         std::string key = attr + "=";
00034         size_t start = line.find(key);
00035
00036         if (start == std::string::npos) return "";
00037
00038         start += key.size();
00039         size_t end = line.find('"', start);
00040
00041         return line.substr(start, end - start);
00042     }
00043
00052     static bool hasPattern(const std::string &str, const std::string &pattern) {
00053         return str.find(pattern) != std::string::npos;
00054     }
00055
00060     static void clear() {
00061         #ifdef _WIN32
00062             system("cls");
00063         #else
00064             system("clear");
00065         #endif
00066     }
00067 };
00068

```

9.25 app/src/app.cpp File Reference

```

#include "app.hpp"
#include <iostream>
#include <string>
#include <fstream>
#include "input.hpp"
#include "utils.hpp"
Include dependency graph for app.cpp:

```

9.26 app/src/date.cpp File Reference

```

#include "date.hpp"
#include "utils.hpp"
Include dependency graph for date.cpp:

```

9.27 app/src/element.cpp File Reference

```
#include "element.hpp"
```

Include dependency graph for element.cpp:

9.28 app/src/file.cpp File Reference

```
#include "file.hpp"
```

Include dependency graph for file.cpp:

9.29 app/src/filename.cpp File Reference

```
#include "filename.hpp"
```

```
#include <sstream>
```

Include dependency graph for filename.cpp:

9.30 app/src/fileSystem.cpp File Reference

```
#include "fileSystem.hpp"
```

```
#include <filesystem>
```

```
#include <fstream>
```

```
#include <unordered_set>
```

```
#include "tinyxml2.h"
```

```
#include "utils.hpp"
```

Include dependency graph for fileSystem.cpp:

9.31 app/src/folder.cpp File Reference

```
#include "folder.hpp"
```

```
#include "date.hpp"
```

```
#include "utils.hpp"
```

Include dependency graph for folder.cpp:

9.32 app/src/input.cpp File Reference

```
#include "input.hpp"
```

```
#include <limits>
```

Include dependency graph for input.cpp:

9.33 app/src/main.cpp File Reference

```
#include "app.hpp"
```

Include dependency graph for main.cpp:

Functions

- int [main](#) ()

9.33.1 Function Documentation

9.33.1.1 main()

```
int main ()
```

9.34 app/src/menu.cpp File Reference

```
#include "menu.hpp"
#include <ftxui/component/component.hpp>
#include <ftxui/component/screen_interactive.hpp>
#include <ftxui/dom/elements.hpp>
#include "utils.hpp"
Include dependency graph for menu.cpp:
```

9.35 app/src/tinyxml2.cpp File Reference

```
#include "tinyxml2.h"
#include <new>
#include <cstdio>
#include <cstdlib>
Include dependency graph for tinyxml2.cpp:
```

Classes

- struct [tinyxml2::Entity](#)

Namespaces

- namespace [tinyxml2](#)

Macros

- #define [TIXML_SNPRINTF](#) snprintf
- #define [TIXML_VSNPRINTF](#) vsnprintf
- #define [TIXML_SSCANF](#) sscanf
- #define [TIXML_FSEEK](#) fseek
- #define [TIXML_FTELL](#) ftell

9.35.1 Macro Definition Documentation

9.35.1.1 TIXML_FSEEK

```
#define TIXML_FSEEK fseek
```

9.35.1.2 TIXML_FTELL

```
#define TIXML_FTELL ftell
```

9.35.1.3 TIXML_SNPRINTF

```
#define TIXML_SNPRINTF snprintf
```

9.35.1.4 TIXML_SSCANF

```
#define TIXML_SSCANF sscanf
```

9.35.1.5 TIXML_VSNPRINTF

```
#define TIXML_VSNPRINTF vsnprintf
```