Projeto Programação Orientada a Objetos

Generated by Doxygen 1.15.0

Chapter 1

Directory Hierarchy

1.1 Directories

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

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_ELEMEN
  , 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 scrutured in day, month and year.

Parameters

ftime	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 tinyxml2::DynArray< T, INITIAL_SIZE > Class Template Reference

```
#include <tinyxml2.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

name	

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

newName	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

newDate	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

counter	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

newExtension	New extension (without '.')

8.7.3.6 setName()

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

Update file's name.

Parameters

newName	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 occurences 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 &currentName, 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

pattern	Pattern to find in
originDir	
destinDir	

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

name	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

file	File to be moved
newFolder	New folder where the file will be moved into

Returns

true Sucess 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

oldDir	Folder to me moved
newDir	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

filename	Filename with extension

Returns

true Success

false Failure

8.8.3.18 removeAll()

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

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

Parameters

name	Name of the folder/file to remove
type	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

currentName	Current name to be changed
newName	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

filename	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

name	Name to search for

type	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

li	List where results will be placed
folder	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

li	List where results will be placed
folder	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

newPath	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

out	Where to show the tree
mirror	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 () 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 &currentName, 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

name	Name of the folder
father	Folder'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

element	Element 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

names	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

pattern	Pattern to find in the name of the file
destin	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

name	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

name	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

name	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

name	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 () 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

path	Path of the current folder to load

Returns

true Folder and all it's content loaded successfuly

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

dirElem	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

name	Name of the element to be removed
type	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

name	Name to remove
type	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

currentName	Current name to change
newName	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

doc	Document in tinyxml2 format
parentElem	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

li	List where to store the paths
name	Name to search
path	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

li	List where to store the paths
name	Name to search
path	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

name	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

name	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

parent	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

prefix	Prefix to the output string
isLast	Wether it is the last file/folder of the current folder
out	Output file
mirror	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

prompt	Prompt to show
allowEmpty	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 tinyxml2::MemPool Class Reference

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

Inheritance diagram for tinyxml2::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

title	Title of the menu
options	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

question	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_COLLAPSI
  = 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()

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

8.14.2.2 ∼StrPair()

```
tinyxml2::StrPair::∼StrPair ()
```

8.14.3 Member Function Documentation

8.14.3.1 Empty()

```
bool tinyxml2::StrPair::Empty () const  [inline]
```

8.14.3.2 GetStr()

```
const char ∗ tinyxml2::StrPair::GetStr ()
```

8.14.3.3 ParseName()

```
char ∗ tinyxml2::StrPair::ParseName (
             char ∗ in)
```

8.14.3.4 ParseText()

```
char ∗ tinyxml2::StrPair::ParseText (
             char ∗ in,
             const char ∗ endTag,
             int strFlags,
             int ∗ curLineNumPtr)
```

8.14.3.5 Reset()

```
void tinyxml2::StrPair::Reset ()
```

8.14.3.6 Set()

```
void tinyxml2::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

line	Line
attr	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

str	string
pattern	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

value	Value to pad with 0's
width	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()

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

See QueryIntValue.

8.17.2.10 QueryDoubleValue()

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

See QueryIntValue.

8.17.2.11 QueryFloatValue()

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

See QueryIntValue.

8.17.2.12 QueryInt64Value()

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

See QueryIntValue.

8.17.2.13 QueryIntValue()

```
XMLError 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()

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

See QueryIntValue.

8.17.2.15 QueryUnsignedValue()

```
XMLError 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()

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

8.20.2.2 ∼XMLDeclaration()

```
tinyxml2::XMLDeclaration::∼XMLDeclaration ()  [protected], [virtual]
```

8.20.3 Member Function Documentation

8.20.3.1 Accept()

```
bool tinyxml2::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;
tinyxmlDoc.Accept( &printer );
const char* xmlcstr = printer.CStr();
```

Implements tinyxml2::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, Tiny↩ XML-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]

```
XMLError 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]

```
XMLError 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().

- XMLError QueryUnsigned64Attribute (const char ∗name, uint64_t ∗value) const

  See QueryIntAttribute().
- XMLError QueryBoolAttribute (const char ∗name, bool ∗value) const

  See QueryIntAttribute().
- XMLError QueryDoubleAttribute (const char ∗name, double ∗value) const

  See QueryIntAttribute().
- XMLError QueryFloatAttribute (const char ∗name, float ∗value) const

  See QueryIntAttribute().
- XMLError QueryStringAttribute (const char ∗name, const char ∗∗value) const

  See QueryIntAttribute().
- XMLError QueryAttribute (const char ∗name, int ∗value) const
- XMLError QueryAttribute (const char ∗name, unsigned int ∗value) const
- XMLError QueryAttribute (const char ∗name, int64_t ∗value) const
- XMLError QueryAttribute (const char ∗name, uint64_t ∗value) const
- XMLError QueryAttribute (const char ∗name, bool ∗value) const
- XMLError QueryAttribute (const char ∗name, double ∗value) const
- XMLError QueryAttribute (const char ∗name, float ∗value) const
- XMLError 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]

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

8.22.3.26 QueryAttribute() [2/8]

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

8.22.3.27 QueryAttribute() [3/8]

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

8.22.3.28 QueryAttribute() [4/8]

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

8.22.3.29 QueryAttribute() [5/8]

```
XMLError 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]

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

8.22.3.31 QueryAttribute() [7/8]

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

8.22.3.32 QueryAttribute() [8/8]

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

8.22.3.33 QueryBoolAttribute()

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

See QueryIntAttribute().

8.22.3.34 QueryBoolText()

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

See QueryIntText().

8.22.3.35 QueryDoubleAttribute()

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

See QueryIntAttribute().

8.22.3.36 QueryDoubleText()

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

See QueryIntText().

8.22.3.37 QueryFloatAttribute()

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

See QueryIntAttribute().

8.22.3.38 QueryFloatText()

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

See QueryIntText().

8.22.3.39 QueryInt64Attribute()

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

See QueryIntAttribute().

8.22.3.40 QueryInt64Text()

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

See QueryIntText().

8.22.3.41 QueryIntAttribute()

```
XMLError 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()

```
XMLError 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()

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

See QueryIntAttribute().

8.22.3.44 QueryUnsigned64Attribute()

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

See QueryIntAttribute().

8.22.3.45 QueryUnsigned64Text()

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

See QueryIntText().

8.22.3.46 QueryUnsignedAttribute()

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

See QueryIntAttribute().

8.22.3.47 QueryUnsignedText()

```
XMLError 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" ).FirstChildElemer
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]

```
tinyxml2::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]

```
tinyxml2::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_ATT
  }

Public Member Functions

- XMLPrinter (FILE ∗file=0, bool compact=false, int depth=0, EscapeAposCharsInAttributes aposIn←
  Attributes=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 tinyxml2::XMLPrinter::PushText (
            float value)
```

Add a text node from a float.

8.25.4.23 PushText() [5/8]

```
void tinyxml2::XMLPrinter::PushText (
            int value)
```

Add a text node from an integer.

8.25.4.24 PushText() [6/8]

```
void tinyxml2::XMLPrinter::PushText (
            int64_t value)
```

Add a text node from a signed 64bit integer.

8.25.4.25 PushText() [7/8]

```
void tinyxml2::XMLPrinter::PushText (
            uint64_t value)
```

Add a text node from an unsigned 64bit integer.

8.25.4.26 PushText() [8/8]

```
void tinyxml2::XMLPrinter::PushText (
            unsigned value)
```

Add a text node from an unsigned.

8.25.4.27 PushUnknown()

```
void tinyxml2::XMLPrinter::PushUnknown (
            const char * value)
```

8.25.4.28 Putc()

```
void tinyxml2::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 next. "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.

```
#pragma once

#include <iostream>

#include "menu.hpp"
#include "fileSystem.hpp"


class App {
    public:
        App();

        void run();
    private:
        Menu mainMenu;
        FileSystem fs;

        void loadSave();
        void statistics();
        void searchs();
        void operations();
        void advanced();
};

```

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.

```
#pragma once

#include <iostream>
#include <string>
#include <cstdint>
#include <filesystem>


class Date {
    public:
        Date();
        Date(std::uint16_t day, std::uint16_t month, std::uint16_t year);
        Date(const std::string &date);

        static Date convertFileTime(const std::filesystem::file_time_type &ftime);
        static Date now();

        std::string getFormattedDate() const;
        std::uint16_t getDay() const;
        std::uint16_t getMonth() const;
        std::uint16_t getYear() const;
    private:
        std::uint16_t day, month, year;

        void parse(const std::string &dateStr);
};

```

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.

```
#pragma once

#include <string>
#include <ostream>

#include "filename.hpp"

enum class ElementType { Folder, File };

class Element {
    public:
        Element(const std::string& name);
        virtual ~Element() = default;

        virtual bool isFile() const = 0;
        virtual bool isFolder() const = 0;

        // Getters
        const Filename getName() const;
        Filename& getName();
        // Setters
        void setName(const std::string& name);
    protected:
        Filename name;
};

```

9.7 app/include/file.hpp File Reference

```
#include <iostream>
#include <string>
#include <cstdint>
#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.

```
#pragma once

#include <iostream>
#include <string>
#include <cstdint>

#include "filename.hpp"
#include "date.hpp"
#include "element.hpp"


class File : public Element {
    public:
        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);

        // Setters
        void setDate(const Date &newDate);
        // Getters
        std::uintmax_t getSize() const;
        const Date getDate() const;

        bool isFile() const override { return true; }
        bool isFolder() const override { return false; }
    private:
        std::uintmax_t size;
        Date date;
};

```

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.

```
#pragma once

#include <iostream>
#include <string>
#include <cstdint>

```

```

class Filename {
    public:
        Filename(const std::string &fullname);
        Filename(const std::string &name, const std::string &extention);

        void generateSequentialName(std::uint16_t counter);
        // Setters
        void setExtension(const std::string &newExtension);
        void setName(const std::string &newName);
        // Getters
        std::string getFullname() const;
        std::string getName() const;
        std::string getExtension() const;
    private:
        std::string name;
        std::string extension;
        std::string getExtension(const std::string& fullname);
        std::string getName(const std::string& fullname);
};

```

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.

```
#pragma once

#include <iostream>
#include <string>
#include <list>
#include <cstdint>
#include <memory>
#include <optional>

#include "folder.hpp"
#include "element.hpp"


class FileSystem {
    public:
        FileSystem();
        FileSystem(const std::string &rootPath);

        bool load(); // 1
        bool load(const std::string &rootPath); // 1

        void clear();

        // Stats
        std::uint32_t countFiles() const; // 2
        std::uint32_t countFolders() const; // 3
        std::uintmax_t memory() const; // 4

        std::string *mostElementsFolder() const; // 5
        std::string *leastElementsFolder() const; // 6
        std::string *largestFile() const; // 7
        std::string *largestFolder() const; // 8

        // XML
        void saveToXML(const std::string &s) const; // 11
        bool readFromXML(const std::string &s); // 12

        // File operations
        bool removeAll(const std::string &name, ElementType type); // 10
        bool moveFile(const std::string &file, const std::string &newDir); // 13
        bool moveFolder(const std::string &oldDir, const std::string &newDir); // 14
        std::string *getFileDate(const std::string &file); // 15
        void renameAllFiles(const std::string &currentName, const std::string &newName); // 19
        bool copyBatch(const std::string &pattern, const std::string &originDir, const std::string &destinDir); // 21

        // Search operations
        std::optional<std::string> search(const std::string &name, ElementType type); // 9
        void searchAllFolders(std::list<std::string> &li, const std::string &folder) const; // 17
        void searchAllFiles(std::list<std::string> &li, const std::string &file) const; // 18

        // Others
        bool checkDupFiles(); // 20
        void tree(std::ostream &out, std::ostream *mirror = nullptr); // 16

        // Setters
        void setPath(const std::string& path);

        // Getters
        const std::string& getPath() const;
    private:
        std::unique_ptr<Folder> root;
        std::string path; // Path to the root directory
};

```

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.

```
#pragma once

#include <iostream>
#include <string>
#include <vector>
#include <memory>
#include <filesystem>
#include <cstdint>
#include <list>
#include <unordered_set>
// tinyxml2 library
#include "tinyxml2.h"

#include "file.hpp"
#include "element.hpp"

constexpr std::uint16_t SPACES_PER_LEVEL = 4;

namespace fs = std::filesystem;
namespace xml = tinyxml2;

class Folder : public Element {
    public:
        Folder(std::string name, Folder *father);

        bool load(const fs::path& path);

        void add(std::unique_ptr<Element> element);
        std::unique_ptr<Element> remove(const std::string& name, ElementType type);

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

        std::uint32_t countFiles() const;
        std::uint32_t countFolders() const;
        std::uintmax_t memory() const;

        const Folder *mostElementsFolder() const;
        const Folder *leastElementsFolder() const;
        const File *largestFile() const;
        const Folder *largestFolder() const;

        void saveToXML(xml::XMLDocument &doc, xml::XMLElement *parentElem) const;
        void readFromXML(xml::XMLElement *dirElem);

        std::string searchFolder(const std::string& name) const;
        void searchAllFolders(std::list<std::string> &li, const std::string& name, const std::string&
path) const;
        std::string searchFile(const std::string& name) const;
        void searchAllFiles(std::list<std::string> &li, const std::string& name, const std::string&
path) const;

        bool checkDupFiles(std::unordered_set<std::string>& names);
        void tree(const std::string &prefix, bool isLast, std::ostream &out, std::ostream *mirror)
const;

        bool removeAll(const std::string &name, ElementType type);
        void renameAllFiles(const std::string &currentName, const std::string &newName);

        bool hasFile(const std::string &name) const;
        // Setters
        void setParent(Folder *parent);
        // Getters
        Folder *getFolderByName(const std::string& name) const;
        File *getFileByName(const std::string& name) const;
        Folder *getFolderByFileName(const std::string& name) const;
        Folder* getParent() const;
        const std::string getName() const;

        bool isFile() const override { return false; }
        bool isFolder() const override { return true; }
    private:
        std::vector<std::unique_ptr<Element>> elements;
        Folder *root;
};
```

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.

```
#pragma once

#include <string>
#include <iostream>
#include <algorithm>


class Input {
    public:
        static std::string getString(const std::string& prompt, bool allowEmpty = false);

        static void wait();

    private:
        static std::string trim(const std::string& str);
};

```

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.

```
#pragma once

#include <string>
#include <vector>

class Menu {
    public:
        Menu(const std::string& title, const std::vector<std::string>& options);

        static bool askYesNo(const std::string& question, bool clearTerminal = false);

        int show(bool clearTerminal = true);
    private:
        std::string title;
        std::vector<std::string> options;
};

```

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.

```
#pragma once

#include <locale>
#include <iostream>

#ifdef _WIN32
    #include <windows.h> // using namespace std gives compilation error (DO NOT USE on .hpp files)
#endif
#include <locale.h>

class SystemConfig {
    public:
        static void setUTF8() {
            #ifdef _WIN32
                // SetConsoleOutputCP returns 0 if there’s an error
                if ((SetConsoleOutputCP(CP_UTF8) == 0)||(SetConsoleCP(CP_UTF8) == 0)) {
                    std::cout « "Ocorreu um erro ao configurar o terminal do Windows para UTF-8." «
std::endl;
                    std::cout « "A aplicação irá continuar. Desformatação será visível. Para resolver,
reinicie a aplicação." « std::endl;
                }
                setlocale(LC_NUMERIC, "Portuguese"); // Floats now use ’,’ instead of ’.’
            #else
                setlocale(LC_ALL, "pt_PT.UTF-8");
            #endif
        }
};

```

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_C
  , tinyxml2::XML_ERROR_PARSING_DECLARATION ,
  tinyxml2::XML_ERROR_PARSING_UNKNOWN , tinyxml2::XML_ERROR_EMPTY_DOCUMENT , tinyxml2::XML_ERROR_MIS
  , tinyxml2::XML_ERROR_PARSING ,
  tinyxml2::XML_CAN_NOT_CONVERT_TEXT , tinyxml2::XML_NO_TEXT_NODE , tinyxml2::XML_ELEMENT_DEPTH_EXCEE
  , 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.

```
/*
Original code by Lee Thomason (www.grinninglizard.com)

This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any
damages arising from the use of this software.

Permission is granted to anyone to use this software for any
purpose, including commercial applications, and to alter it and
redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must
not claim that you wrote the original software. If you use this
software in a product, an acknowledgment in the product documentation
would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and
must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source
distribution.
*/

#ifndef TINYXML2_INCLUDED
#define TINYXML2_INCLUDED

#if defined(ANDROID_NDK) || defined(__BORLANDC__) || defined(__QNXNTO__)
#   include <ctype.h>
#   include <limits.h>
#   include <stdio.h>
#   include <stdlib.h>
#   include <string.h>
#   if defined(__PS3__)
#       include <stddef.h>
#   endif
#else
#   include <cctype>
#   include <climits>
#   include <cstdio>
#   include <cstdlib>
#   include <cstring>
#endif
#include <stdint.h>

/*
    gcc:
        g++ -Wall -DTINYXML2_DEBUG tinyxml2.cpp xmltest.cpp -o gccxmltest.exe

    Formatting, Artistic Style:
        AStyle.exe --style=1tbs --indent-switches --break-closing-brackets --indent-preprocessor
tinyxml2.cpp tinyxml2.h
*/

#if defined( _DEBUG ) || defined (__DEBUG__)
#   ifndef TINYXML2_DEBUG
#       define TINYXML2_DEBUG
#   endif
#endif

#ifdef _MSC_VER
#   pragma warning(push)
#   pragma warning(disable: 4251)
#endif

#ifdef _MSC_VER
#   ifdef TINYXML2_EXPORT
#       define TINYXML2_LIB __declspec(dllexport)
#   elif defined(TINYXML2_IMPORT)
#       define TINYXML2_LIB __declspec(dllimport)
#   else
#       define TINYXML2_LIB
#   endif
#elif __GNUC__ >= 4
#   define TINYXML2_LIB __attribute__((visibility("default")))
#else
#   define TINYXML2_LIB
#endif


#if !defined(TIXMLASSERT)
#if defined(TINYXML2_DEBUG)
#   if defined(_MSC_VER)
#       // "(void)0," is for suppressing C4127 warning in "assert(false)", "assert(true)" and the like
#       define TIXMLASSERT( x )           do { if ( !((void)0,(x))) { __debugbreak(); } } while(false)
#   elif defined (ANDROID_NDK)
#       include <android/log.h>
#       define TIXMLASSERT( x )           do { if ( !(x)) { __android_log_assert( "assert", "grinliz",
"ASSERT in '%s' at %d.", __FILE__, __LINE__ ); } } while(false)
#   else
#       include <assert.h>
#       define TIXMLASSERT                assert
#   endif
#else
#   define TIXMLASSERT( x )               do {} while(false)
#endif
#endif

/* Versioning, past 1.0.14:
    http://semver.org/
*/
static const int TIXML2_MAJOR_VERSION = 11;
static const int TIXML2_MINOR_VERSION = 0;
static const int TIXML2_PATCH_VERSION = 0;

#define TINYXML2_MAJOR_VERSION 11
#define TINYXML2_MINOR_VERSION 0
#define TINYXML2_PATCH_VERSION 0

// A fixed element depth limit is problematic. There needs to be a
// limit to avoid a stack overflow. However, that limit varies per
// system, and the capacity of the stack. On the other hand, it's a trivial
// attack that can result from ill, malicious, or even correctly formed XML,
// so there needs to be a limit in place.
static const int TINYXML2_MAX_ELEMENT_DEPTH = 500;

```

```
namespace tinyxml2
{
class XMLDocument;
class XMLElement;
class XMLAttribute;
class XMLComment;
class XMLText;
class XMLDeclaration;
class XMLUnknown;
class XMLPrinter;

/*
    A class that wraps strings. Normally stores the start and end
    pointers into the XML file itself, and will apply normalization
    and entity translation if actually read. Can also store (and memory
    manage) a traditional char[]

    Isn't clear why TINYXML2_LIB is needed; but seems to fix #719
*/
class TINYXML2_LIB StrPair
{
public:
    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
    };

    StrPair() : _flags( 0 ), _start( 0 ), _end( 0 ) {}
    ~StrPair();

    void Set( char* start, char* end, int flags ) {
        TIXMLASSERT( start );
        TIXMLASSERT( end );
        Reset();
        _start  = start;
        _end    = end;
        _flags  = flags | NEEDS_FLUSH;
    }

    const char* GetStr();

    bool Empty() const {
        return _start == _end;
    }

    void SetInternedStr( const char* str ) {
        Reset();
        _start = const_cast<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();

private:
    void CollapseWhitespace();

    enum {
        NEEDS_FLUSH = 0x100,
        NEEDS_DELETE = 0x200
    };

    int     _flags;
    char*   _start;
    char*   _end;

    StrPair( const StrPair& other );    // not supported
    void operator=( const StrPair& other ); // not supported, use TransferTo()
};


/*
    A dynamic array of Plain Old Data. Doesn't support constructors, etc.
    Has a small initial memory pool, so that low or no usage will not
    cause a call to new/delete
```

```
*/
template <class T, size_t INITIAL_SIZE>
class DynArray
{
public:
    DynArray() :
        _mem( _pool ),
        _allocated( INITIAL_SIZE ),
        _size( 0 )
    {
    }

    ~DynArray() {
        if ( _mem != _pool ) {
            delete [] _mem;
        }
    }

    void Clear() {
        _size = 0;
    }

    void Push( T t ) {
        TIXMLASSERT( _size < INT_MAX );
        EnsureCapacity( _size+1 );
        _mem[_size] = t;
        ++_size;
    }

    T* PushArr( size_t count ) {
        TIXMLASSERT( _size <= SIZE_MAX - count );
        EnsureCapacity( _size+count );
        T* ret = &_mem[_size];
        _size += count;
        return ret;
    }

    T Pop() {
        TIXMLASSERT( _size > 0 );
        --_size;
        return _mem[_size];
    }

    void PopArr( size_t count ) {
        TIXMLASSERT( _size >= count );
        _size -= count;
    }

    bool Empty() const                  {
        return _size == 0;
    }

    T& operator[](size_t i) {
        TIXMLASSERT( i < _size );
        return _mem[i];
    }

    const T& operator[](size_t i) const {
        TIXMLASSERT( i < _size );
        return _mem[i];
    }

    const T& PeekTop() const            {
        TIXMLASSERT( _size > 0 );
        return _mem[ _size - 1];
    }

    size_t Size() const {
        return _size;
    }

    size_t Capacity() const {
        TIXMLASSERT( _allocated >= INITIAL_SIZE );
        return _allocated;
    }

    void SwapRemove(size_t i) {
        TIXMLASSERT(i < _size);
        TIXMLASSERT(_size > 0);
        _mem[i] = _mem[_size - 1];
        --_size;
    }

    const T* Mem() const                {
        TIXMLASSERT( _mem );
        return _mem;
    }

    T* Mem() {
        TIXMLASSERT( _mem );
        return _mem;
    }

private:
    DynArray( const DynArray& ); // not supported
    void operator=( const DynArray& ); // not supported

    void EnsureCapacity( size_t cap ) {
        TIXMLASSERT( cap > 0 );
        if ( cap > _allocated ) {
            TIXMLASSERT( cap <= SIZE_MAX / 2 / sizeof(T));
            const size_t newAllocated = cap * 2;
            T* newMem = new T[newAllocated];
            TIXMLASSERT( newAllocated >= _size );
            memcpy( newMem, _mem, sizeof(T) * _size );  // warning: not using constructors, only works
for PODs
            if ( _mem != _pool ) {
                delete [] _mem;
            }
            _mem = newMem;
            _allocated = newAllocated;
        }
    }

    T*  _mem;
    T   _pool[INITIAL_SIZE];
    size_t _allocated;      // objects allocated
    size_t _size;           // number objects in use
};


/*
    Parent virtual class of a pool for fast allocation
    and deallocation of objects.
*/
class MemPool
{
public:
    MemPool() {}
    virtual ~MemPool() {}

    virtual size_t ItemSize() const = 0;
    virtual void* Alloc() = 0;
    virtual void Free( void* ) = 0;
    virtual void SetTracked() = 0;
};


/*
    Template child class to create pools of the correct type.
*/
template< size_t ITEM_SIZE >
class MemPoolT : public MemPool
{
public:
    MemPoolT() : _blockPtrs(), _root(0), _currentAllocs(0), _nAllocs(0), _maxAllocs(0), _nUntracked(0)
{}
    ~MemPoolT() {
        MemPoolT< ITEM_SIZE >::Clear();
    }

    void Clear() {
        // Delete the blocks.
        while( !_blockPtrs.Empty()) {
            Block* lastBlock = _blockPtrs.Pop();
            delete lastBlock;
        }
        _root = 0;
        _currentAllocs = 0;
        _nAllocs = 0;
        _maxAllocs = 0;
        _nUntracked = 0;
    }

    virtual size_t ItemSize() const override {
        return ITEM_SIZE;
    }
    size_t CurrentAllocs() const {
        return _currentAllocs;
    }

    virtual void* Alloc() override{
        if ( !_root ) {
            // Need a new block.
            Block* block = new Block;
            _blockPtrs.Push( block );

            Item* blockItems = block->items;
            for( size_t i = 0; i < ITEMS_PER_BLOCK - 1; ++i ) {
                blockItems[i].next = &(blockItems[i + 1]);
            }
            blockItems[ITEMS_PER_BLOCK - 1].next = 0;
            _root = blockItems;
        }
        Item* const result = _root;
        TIXMLASSERT( result != 0 );
        _root = _root->next;

        ++_currentAllocs;
        if ( _currentAllocs > _maxAllocs ) {
            _maxAllocs = _currentAllocs;
        }
        ++_nAllocs;
        ++_nUntracked;
        return result;
    }

    virtual void Free( void* mem ) override {
        if ( !mem ) {
            return;
        }
        --_currentAllocs;
        Item* item = static_cast<Item*>( mem );
#ifdef TINYXML2_DEBUG
        memset( item, 0xfe, sizeof( *item ) );
#endif
        item->next = _root;
        _root = item;
    }
    void Trace( const char* name ) {
        printf( "Mempool %s watermark=%d [%dk] current=%d size=%d nAlloc=%d blocks=%d\n",
                name, _maxAllocs, _maxAllocs * ITEM_SIZE / 1024, _currentAllocs,
                ITEM_SIZE, _nAllocs, _blockPtrs.Size() );
    }

    void SetTracked() override {
        --_nUntracked;
    }

    size_t Untracked() const {
        return _nUntracked;
    }

    // This number is perf sensitive. 4k seems like a good tradeoff on my machine.
    // The test file is large, 170k.
    // Release:     VS2010 gcc(no opt)
    //      1k:     4000
    //      2k:     4000
    //      4k:     3900    21000
    //      16k:    5200
    //      32k:    4300
    //      64k:    4000    21000
    // Declared public because some compilers do not accept to use ITEMS_PER_BLOCK
    // in private part if ITEMS_PER_BLOCK is private
    enum { ITEMS_PER_BLOCK = (4 * 1024) / ITEM_SIZE };

private:
    MemPoolT( const MemPoolT& ); // not supported
    void operator=( const MemPoolT& ); // not supported

    union Item {
        Item*   next;
        char    itemData[static_cast<size_t>(ITEM_SIZE)];
    };
    struct Block {
        Item items[ITEMS_PER_BLOCK];
    };
    DynArray< Block*, 10 > _blockPtrs;
    Item* _root;

    size_t _currentAllocs;
    size_t _nAllocs;
    size_t _maxAllocs;
    size_t _nUntracked;
};



class TINYXML2_LIB XMLVisitor
{
public:
    virtual ~XMLVisitor() {}

    virtual bool VisitEnter( const XMLDocument& /*doc*/ )           {
        return true;
    }
    virtual bool VisitExit( const XMLDocument& /*doc*/ )            {
        return true;
    }

    virtual bool VisitEnter( const XMLElement& /*element*/, const XMLAttribute* /*firstAttribute*/ )
{
        return true;
    }
    virtual bool VisitExit( const XMLElement& /*element*/ )         {
        return true;
    }

    virtual bool Visit( const XMLDeclaration& /*declaration*/ )     {
        return true;
    }
    virtual bool Visit( const XMLText& /*text*/ )                   {
        return true;
    }
    virtual bool Visit( const XMLComment& /*comment*/ )             {
        return true;
    }
    virtual bool Visit( const XMLUnknown& /*unknown*/ )             {
        return true;
    }
};

// WARNING: must match XMLDocument::_errorNames[]
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
};


/*
    Utility functionality.
*/
class TINYXML2_LIB XMLUtil
{
public:
    static const char* SkipWhiteSpace( const char* p, int* curLineNumPtr )  {
        TIXMLASSERT( p );

        while( IsWhiteSpace(*p) ) {
            if (curLineNumPtr && *p == '\n') {
                ++(*curLineNumPtr);
            }
            ++p;
        }
        TIXMLASSERT( p );
        return p;
    }
    static char* SkipWhiteSpace( char* const p, int* curLineNumPtr ) {
        return const_cast<char*>( SkipWhiteSpace( const_cast<const char*>(p), curLineNumPtr ) );
    }

    // Anything in the high order range of UTF-8 is assumed to not be whitespace. This isn't
    // correct, but simple, and usually works.
    static bool IsWhiteSpace( char p )                  {
        return !IsUTF8Continuation(p) && isspace( static_cast<unsigned char>(p) );
    }

    inline static bool IsNameStartChar( unsigned char ch ) {
        if ( ch >= 128 ) {
            // This is a heuristic guess in attempt to not implement Unicode-aware isalpha()
            return true;
        }
        if ( isalpha( ch ) ) {
            return true;
        }
        return ch == ':' || ch == '_';
    }

    inline static bool IsNameChar( unsigned char ch ) {
        return IsNameStartChar( ch )
               || isdigit( ch )
               || ch == '.'
               || ch == '-';
    }

    inline static bool IsPrefixHex( const char* p) {
        p = SkipWhiteSpace(p, 0);
        return p && *p == '0' && ( *(p + 1) == 'x' || *(p + 1) == 'X');
    }

    inline static bool StringEqual( const char* p, const char* q, int nChar=INT_MAX )  {
        if ( p == q ) {
            return true;
        }
        TIXMLASSERT( p );
        TIXMLASSERT( q );
        TIXMLASSERT( nChar >= 0 );
        return strncmp( p, q, static_cast<size_t>(nChar) ) == 0;
    }

    inline static bool IsUTF8Continuation( const char p ) {
        return ( p & 0x80 ) != 0;
    }

    static const char* ReadBOM( const char* p, bool* hasBOM );
    // p is the starting location,
    // the UTF-8 value of the entity will be placed in value, and length filled in.
    static const char* GetCharacterRef( const char* p, char* value, int* length );
    static void ConvertUTF32ToUTF8( unsigned long input, char* output, int* length );

    // converts primitive types to strings
    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);

    // converts strings to primitive types
    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);
    // Changes what is serialized for a boolean value.
    // Default to "true" and "false". Shouldn't be changed
    // unless you have a special testing or compatibility need.
    // Be careful: static, global, & not thread safe.
    // Be sure to set static const memory as parameters.
    static void SetBoolSerialization(const char* writeTrue, const char* writeFalse);

private:
    static const char* writeBoolTrue;
    static const char* writeBoolFalse;
};


class TINYXML2_LIB XMLNode
{
    friend class XMLDocument;
    friend class XMLElement;
public:

    const XMLDocument* GetDocument() const  {
        TIXMLASSERT( _document );
        return _document;
    }
    XMLDocument* GetDocument()              {
        TIXMLASSERT( _document );
        return _document;
    }

    virtual XMLElement*     ToElement()     {
        return 0;
    }
    virtual XMLText*        ToText()        {
        return 0;
    }
    virtual XMLComment*     ToComment()     {
        return 0;
    }
    virtual XMLDocument*    ToDocument()    {
        return 0;
    }
    virtual XMLDeclaration* ToDeclaration() {
        return 0;
    }
    virtual XMLUnknown*     ToUnknown()     {
        return 0;
    }

    virtual const XMLElement*       ToElement() const       {
        return 0;
    }
    virtual const XMLText*          ToText() const          {
        return 0;
    }
    virtual const XMLComment*       ToComment() const       {
        return 0;
    }
    virtual const XMLDocument*      ToDocument() const      {
        return 0;
    }
    virtual const XMLDeclaration*   ToDeclaration() const   {
        return 0;
    }
    virtual const XMLUnknown*       ToUnknown() const       {
        return 0;
    }

    // ChildElementCount was originally suggested by msteiger on the sourceforge page for TinyXML and modified by KB1SPH for TinyXML-2.

    int ChildElementCount(const char *value) const;

    int ChildElementCount() const;

    const char* Value() const;

    void SetValue( const char* val, bool staticMem=false );

    int GetLineNum() const { return _parseLineNum; }

    const XMLNode*  Parent() const          {
        return _parent;
    }

    XMLNode* Parent()                       {
        return _parent;
    }

    bool NoChildren() const                 {
        return !_firstChild;
    }

    const XMLNode*  FirstChild() const      {
        return _firstChild;
    }

    XMLNode*        FirstChild()            {
        return _firstChild;
    }

    const XMLElement* FirstChildElement( const char* name = 0 ) const;

    XMLElement* FirstChildElement( const char* name = 0 )   {
        return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->FirstChildElement( name ));
    }

    const XMLNode*  LastChild() const                       {
        return _lastChild;
    }

    XMLNode*        LastChild()                             {
        return _lastChild;
    }

    const XMLElement* LastChildElement( const char* name = 0 ) const;

    XMLElement* LastChildElement( const char* name = 0 )    {
        return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->LastChildElement(name) );
    }

    const XMLNode*  PreviousSibling() const                    {
        return _prev;
    }

    XMLNode*    PreviousSibling()                              {
        return _prev;
    }

    const XMLElement*   PreviousSiblingElement( const char* name = 0 ) const ;

    XMLElement* PreviousSiblingElement( const char* name = 0 ) {
        return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->PreviousSiblingElement( name
) );
    }

    const XMLNode*  NextSibling() const                        {
        return _next;
    }

    XMLNode*    NextSibling()                                  {
        return _next;
    }

    const XMLElement*   NextSiblingElement( const char* name = 0 ) const;

    XMLElement* NextSiblingElement( const char* name = 0 )  {
        return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->NextSiblingElement( name ) );
    }

    XMLNode* InsertEndChild( XMLNode* addThis );

    XMLNode* LinkEndChild( XMLNode* addThis )   {
        return InsertEndChild( 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)    { _userData = userData; }

    void* GetUserData() const           { return _userData; }

protected:
    explicit XMLNode( XMLDocument* );
    virtual ~XMLNode();

    virtual char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr);

    XMLDocument*    _document;
    XMLNode*        _parent;
    mutable StrPair _value;
    int             _parseLineNum;

    XMLNode*        _firstChild;
    XMLNode*        _lastChild;

    XMLNode*        _prev;
    XMLNode*        _next;

    void*           _userData;

private:
    MemPool*        _memPool;
    void Unlink( XMLNode* child );
    static void DeleteNode( XMLNode* node );
    void InsertChildPreamble( XMLNode* insertThis ) const;
    const XMLElement* ToElementWithName( const char* name ) const;

    XMLNode( const XMLNode& );  // not supported
    XMLNode& operator=( const XMLNode& );   // not supported
};


```

```
class TINYXML2_LIB XMLText : public XMLNode
{
    friend class XMLDocument;
public:
    virtual bool Accept( XMLVisitor* visitor ) const override;

    virtual XMLText* ToText() override      {
        return this;
    }
    virtual const XMLText* ToText() const override {
        return this;
    }

    void SetCData( bool isCData )           {
        _isCData = isCData;
    }
    bool CData() const                      {
        return _isCData;
    }

    virtual XMLNode* ShallowClone( XMLDocument* document ) const override;
    virtual bool ShallowEqual( const XMLNode* compare ) const override;

protected:
    explicit XMLText( XMLDocument* doc )    : XMLNode( doc ), _isCData( false ) {}
    virtual ~XMLText()                                              {}

    char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr ) override;

private:
    bool _isCData;

    XMLText( const XMLText& );  // not supported
    XMLText& operator=( const XMLText& );   // not supported
};


class TINYXML2_LIB XMLComment : public XMLNode
{
    friend class XMLDocument;
public:
    virtual XMLComment* ToComment() override        {
        return this;
    }
    virtual const XMLComment* ToComment() const override {
        return this;
    }

    virtual bool Accept( XMLVisitor* visitor ) const override;

    virtual XMLNode* ShallowClone( XMLDocument* document ) const override;
    virtual bool ShallowEqual( const XMLNode* compare ) const override;

protected:
    explicit XMLComment( XMLDocument* doc );
    virtual ~XMLComment();

    char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr) override;

private:
    XMLComment( const XMLComment& );    // not supported
    XMLComment& operator=( const XMLComment& ); // not supported
};


class TINYXML2_LIB XMLDeclaration : public XMLNode
{
    friend class XMLDocument;
public:
    virtual XMLDeclaration* ToDeclaration() override        {
        return this;
    }
    virtual const XMLDeclaration* ToDeclaration() const override {
        return this;
    }

    virtual bool Accept( XMLVisitor* visitor ) const override;

    virtual XMLNode* ShallowClone( XMLDocument* document ) const override;
    virtual bool ShallowEqual( const XMLNode* compare ) const override;

protected:
    explicit XMLDeclaration( XMLDocument* doc );
    virtual ~XMLDeclaration();

    char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr ) override;

```

```
private:
    XMLDeclaration( const XMLDeclaration& );    // not supported
    XMLDeclaration& operator=( const XMLDeclaration& ); // not supported
};


class TINYXML2_LIB XMLUnknown : public XMLNode
{
    friend class XMLDocument;
public:
    virtual XMLUnknown* ToUnknown() override        {
        return this;
    }
    virtual const XMLUnknown* ToUnknown() const override {
        return this;
    }

    virtual bool Accept( XMLVisitor* visitor ) const override;

    virtual XMLNode* ShallowClone( XMLDocument* document ) const override;
    virtual bool ShallowEqual( const XMLNode* compare ) const override;

protected:
    explicit XMLUnknown( XMLDocument* doc );
    virtual ~XMLUnknown();

    char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr ) override;

private:
    XMLUnknown( const XMLUnknown& );    // not supported
    XMLUnknown& operator=( const XMLUnknown& ); // not supported
};



class TINYXML2_LIB XMLAttribute
{
    friend class XMLElement;
public:
    const char* Name() const;

    const char* Value() const;

    int GetLineNum() const { return _parseLineNum; }

    const XMLAttribute* Next() const {
        return _next;
    }

    int IntValue() const {
        int i = 0;
        QueryIntValue(&i);
        return i;
    }

    int64_t Int64Value() const {
        int64_t i = 0;
        QueryInt64Value(&i);
        return i;
    }

    uint64_t Unsigned64Value() const {
        uint64_t i = 0;
        QueryUnsigned64Value(&i);
        return i;
    }

    unsigned UnsignedValue() const          {
        unsigned i=0;
        QueryUnsignedValue( &i );
        return i;
    }
    bool     BoolValue() const              {
        bool b=false;
        QueryBoolValue( &b );
        return b;
    }
    double   DoubleValue() const            {
        double d=0;
        QueryDoubleValue( &d );
        return d;
    }
    float    FloatValue() const             {
        float f=0;
        QueryFloatValue( &f );
        return f;
    }

    XMLError QueryIntValue( int* value ) const;
    XMLError QueryUnsignedValue( unsigned int* value ) const;
    XMLError QueryInt64Value(int64_t* value) const;
    XMLError QueryUnsigned64Value(uint64_t* value) const;
    XMLError QueryBoolValue( bool* value ) const;
    XMLError QueryDoubleValue( double* value ) const;
    XMLError QueryFloatValue( float* value ) const;

    void SetAttribute( const char* value );
    void SetAttribute( int value );
    void SetAttribute( unsigned value );
    void SetAttribute(int64_t value);
    void SetAttribute(uint64_t value);
    void SetAttribute( bool value );
    void SetAttribute( double value );
    void SetAttribute( float value );

private:
    enum { BUF_SIZE = 200 };

    XMLAttribute() : _name(), _value(),_parseLineNum( 0 ), _next( 0 ), _memPool( 0 ) {}
    virtual ~XMLAttribute() {}

    XMLAttribute( const XMLAttribute& );    // not supported
    void operator=( const XMLAttribute& );  // not supported
    void SetName( const char* name );

    char* ParseDeep( char* p, bool processEntities, int* curLineNumPtr );

    mutable StrPair _name;
    mutable StrPair _value;
    int             _parseLineNum;
    XMLAttribute*   _next;
    MemPool*        _memPool;
};


class TINYXML2_LIB XMLElement : public XMLNode
{
    friend class XMLDocument;
public:
    const char* Name() const        {
        return Value();
    }
    void SetName( const char* str, bool staticMem=false )   {
        SetValue( str, staticMem );
    }

    virtual XMLElement* ToElement() override    {
        return this;
    }
    virtual const XMLElement* ToElement() const override {
        return this;
    }
    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;
    int64_t Int64Attribute(const char* name, int64_t defaultValue = 0) const;
    uint64_t Unsigned64Attribute(const char* name, uint64_t defaultValue = 0) const;
    bool BoolAttribute(const char* name, bool defaultValue = false) const;
    double DoubleAttribute(const char* name, double defaultValue = 0) const;
    float FloatAttribute(const char* name, float defaultValue = 0) const;

    XMLError QueryIntAttribute( const char* name, int* value ) const                {
        const XMLAttribute* a = FindAttribute( name );
        if ( !a ) {
            return XML_NO_ATTRIBUTE;
        }
        return a->QueryIntValue( value );
    }

    XMLError QueryUnsignedAttribute( const char* name, unsigned int* value ) const  {
        const XMLAttribute* a = FindAttribute( name );
        if ( !a ) {
            return XML_NO_ATTRIBUTE;
        }
        return a->QueryUnsignedValue( value );
    }

    XMLError QueryInt64Attribute(const char* name, int64_t* value) const {
        const XMLAttribute* a = FindAttribute(name);
        if (!a) {
            return XML_NO_ATTRIBUTE;
        }
        return a->QueryInt64Value(value);
    }

    XMLError QueryUnsigned64Attribute(const char* name, uint64_t* value) const {
        const XMLAttribute* a = FindAttribute(name);
        if(!a) {
            return XML_NO_ATTRIBUTE;
        }
        return a->QueryUnsigned64Value(value);
    }

    XMLError QueryBoolAttribute( const char* name, bool* value ) const              {
        const XMLAttribute* a = FindAttribute( name );
        if ( !a ) {
            return XML_NO_ATTRIBUTE;
        }
        return a->QueryBoolValue( value );
    }
    XMLError QueryDoubleAttribute( const char* name, double* value ) const          {
        const XMLAttribute* a = FindAttribute( name );
        if ( !a ) {
            return XML_NO_ATTRIBUTE;
        }
        return a->QueryDoubleValue( value );
    }
    XMLError QueryFloatAttribute( const char* name, float* value ) const            {
        const XMLAttribute* a = FindAttribute( name );
        if ( !a ) {
            return XML_NO_ATTRIBUTE;
        }
        return a->QueryFloatValue( value );
    }

    XMLError QueryStringAttribute(const char* name, const char** value) const {
        const XMLAttribute* a = FindAttribute(name);
        if (!a) {
            return XML_NO_ATTRIBUTE;
        }
        *value = a->Value();
        return XML_SUCCESS;
    }



    XMLError QueryAttribute( const char* name, int* value ) const {
        return QueryIntAttribute( name, value );
    }

    XMLError QueryAttribute( const char* name, unsigned int* value ) const {
        return QueryUnsignedAttribute( name, value );
    }

    XMLError QueryAttribute(const char* name, int64_t* value) const {
        return QueryInt64Attribute(name, value);
    }

    XMLError QueryAttribute(const char* name, uint64_t* value) const {
        return QueryUnsigned64Attribute(name, value);
    }

    XMLError QueryAttribute( const char* name, bool* value ) const {
        return QueryBoolAttribute( name, value );
    }

    XMLError QueryAttribute( const char* name, double* value ) const {
        return QueryDoubleAttribute( name, value );
    }

    XMLError QueryAttribute( const char* name, float* value ) const {
        return QueryFloatAttribute( name, value );
    }

    XMLError QueryAttribute(const char* name, const char** value) const {
        return QueryStringAttribute(name, value);
    }

    void SetAttribute( const char* name, const char* value )    {
        XMLAttribute* a = FindOrCreateAttribute( name );
        a->SetAttribute( value );
    }
    void SetAttribute( const char* name, int value )            {
        XMLAttribute* a = FindOrCreateAttribute( name );
        a->SetAttribute( value );
    }
    void SetAttribute( const char* name, unsigned value )       {
        XMLAttribute* a = FindOrCreateAttribute( name );
        a->SetAttribute( value );
    }

    void SetAttribute(const char* name, int64_t value) {
        XMLAttribute* a = FindOrCreateAttribute(name);
        a->SetAttribute(value);
    }

    void SetAttribute(const char* name, uint64_t value) {
        XMLAttribute* a = FindOrCreateAttribute(name);
        a->SetAttribute(value);
    }

    void SetAttribute( const char* name, bool value )           {
        XMLAttribute* a = FindOrCreateAttribute( name );
        a->SetAttribute( value );
    }
    void SetAttribute( const char* name, double value )     {
        XMLAttribute* a = FindOrCreateAttribute( name );
        a->SetAttribute( value );
    }
    void SetAttribute( const char* name, float value )      {
        XMLAttribute* a = FindOrCreateAttribute( name );
        a->SetAttribute( value );
    }

    void DeleteAttribute( const char* name );

    const XMLAttribute* FirstAttribute() const {
        return _rootAttribute;
    }
    const XMLAttribute* FindAttribute( const char* name ) const;

    const char* GetText() const;

    void SetText( const char* inText );
    void SetText( int value );
    void SetText( unsigned value );
    void SetText(int64_t value);
    void SetText(uint64_t value);
    void SetText( bool value );
    void SetText( double value );
    void SetText( float value );

    XMLError QueryIntText( int* ival ) const;
    XMLError QueryUnsignedText( unsigned* uval ) const;
    XMLError QueryInt64Text(int64_t* uval) const;
    XMLError QueryUnsigned64Text(uint64_t* uval) const;
    XMLError QueryBoolText( bool* bval ) const;
    XMLError QueryDoubleText( double* dval ) const;
    XMLError QueryFloatText( float* fval ) const;

    int IntText(int defaultValue = 0) const;

    unsigned UnsignedText(unsigned defaultValue = 0) const;
    int64_t Int64Text(int64_t defaultValue = 0) const;
    uint64_t Unsigned64Text(uint64_t defaultValue = 0) const;
    bool BoolText(bool defaultValue = false) const;
    double DoubleText(double defaultValue = 0) const;
    float FloatText(float defaultValue = 0) const;

    XMLElement* InsertNewChildElement(const char* name);
    XMLComment* InsertNewComment(const char* comment);
    XMLText* InsertNewText(const char* text);
    XMLDeclaration* InsertNewDeclaration(const char* text);
    XMLUnknown* InsertNewUnknown(const char* text);


    // internal:
    enum ElementClosingType {
        OPEN,       // <foo>
        CLOSED,     // <foo/>
        CLOSING     // </foo>
    };
    ElementClosingType ClosingType() const {
        return _closingType;
    }
    virtual XMLNode* ShallowClone( XMLDocument* document ) const override;
    virtual bool ShallowEqual( const XMLNode* compare ) const override;

protected:
    char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr ) override;

private:
    XMLElement( XMLDocument* doc );
    virtual ~XMLElement();
    XMLElement( const XMLElement& );    // not supported
    void operator=( const XMLElement& );    // not supported

    XMLAttribute* FindOrCreateAttribute( const char* name );
    char* ParseAttributes( char* p, int* curLineNumPtr );
    static void DeleteAttribute( XMLAttribute* attribute );
    XMLAttribute* CreateAttribute();

    enum { BUF_SIZE = 200 };
    ElementClosingType _closingType;
    // The attribute list is ordered; there is no 'lastAttribute'
    // because the list needs to be scanned for dupes before adding
    // a new attribute.
    XMLAttribute* _rootAttribute;
};


enum Whitespace {
    PRESERVE_WHITESPACE,
    COLLAPSE_WHITESPACE,
    PEDANTIC_WHITESPACE
};


class TINYXML2_LIB XMLDocument : public XMLNode
{
    friend class XMLElement;
    // Gives access to SetError and Push/PopDepth, but over-access for everything else.
    // Wishing C++ had "internal" scope.
    friend class XMLNode;
    friend class XMLText;
    friend class XMLComment;
    friend class XMLDeclaration;
    friend class XMLUnknown;
public:
    XMLDocument( bool processEntities = true, Whitespace whitespaceMode = PRESERVE_WHITESPACE );
    ~XMLDocument();

    virtual XMLDocument* ToDocument() override      {
        TIXMLASSERT( this == _document );
        return this;
    }
    virtual const XMLDocument* ToDocument() const override {
        TIXMLASSERT( this == _document );
        return this;
    }

    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        {
        return _processEntities;
    }
    Whitespace WhitespaceMode() const   {
        return _whitespaceMode;
    }

    bool HasBOM() const {
        return _writeBOM;
    }
    void SetBOM( bool useBOM ) {
        _writeBOM = useBOM;
    }

    XMLElement* RootElement()               {
        return FirstChildElement();
    }
    const XMLElement* RootElement() const   {
        return FirstChildElement();
    }

    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();

    bool Error() const {
        return _errorID != XML_SUCCESS;
    }
    XMLError  ErrorID() const {
        return _errorID;
    }
    const char* ErrorName() const;
    static const char* ErrorIDToName(XMLError errorID);

    const char* ErrorStr() const;

    void PrintError() const;

    int ErrorLineNum() const
    {
        return _errorLineNum;
    }

    void Clear();

    void DeepCopy(XMLDocument* target) const;

    // internal
    char* Identify( char* p, XMLNode** node, bool first );

    // internal
    void MarkInUse(const XMLNode* const);

    virtual XMLNode* ShallowClone( XMLDocument* /*document*/ ) const override{
        return 0;
    }
    virtual bool ShallowEqual( const XMLNode* /*compare*/ ) const override{
        return false;
    }

private:
    XMLDocument( const XMLDocument& );  // not supported
    void operator=( const XMLDocument& );   // not supported

    bool            _writeBOM;
    bool            _processEntities;
    XMLError        _errorID;
    Whitespace      _whitespaceMode;
    mutable StrPair _errorStr;
    int             _errorLineNum;
    char*           _charBuffer;
    int             _parseCurLineNum;
    int             _parsingDepth;
    // Memory tracking does add some overhead.
    // However, the code assumes that you don't
    // have a bunch of unlinked nodes around.
    // Therefore it takes less memory to track
    // in the document vs. a linked list in the XMLNode,
    // and the performance is the same.
    DynArray<XMLNode*, 10> _unlinked;

    MemPoolT< sizeof(XMLElement) >   _elementPool;
    MemPoolT< sizeof(XMLAttribute) > _attributePool;
    MemPoolT< sizeof(XMLText) >      _textPool;
    MemPoolT< sizeof(XMLComment) >   _commentPool;

    static const char* _errorNames[XML_ERROR_COUNT];

    void Parse();

    void SetError( XMLError error, int lineNum, const char* format, ... );

    // Something of an obvious security hole, once it was discovered.
    // Either an ill-formed XML or an excessively deep one can overflow
    // the stack. Track stack depth, and error out if needed.
    class DepthTracker {
    public:
        explicit DepthTracker(XMLDocument * document) {
            this->_document = document;
            document->PushDepth();
        }
        ~DepthTracker() {
            _document->PopDepth();
        }
    private:
        XMLDocument * _document;
    };
    void PushDepth();
    void PopDepth();

    template<class NodeType, size_t PoolElementSize>
    NodeType* CreateUnlinkedNode( MemPoolT<PoolElementSize>& pool );
};

template<class NodeType, size_t PoolElementSize>
inline NodeType* XMLDocument::CreateUnlinkedNode( MemPoolT<PoolElementSize>& pool )
{
    TIXMLASSERT( sizeof( NodeType ) == PoolElementSize );
    TIXMLASSERT( sizeof( NodeType ) == pool.ItemSize() );
    NodeType* returnNode = new (pool.Alloc()) NodeType( this );
    TIXMLASSERT( returnNode );
    returnNode->_memPool = &pool;

    _unlinked.Push(returnNode);
    return returnNode;
}

class TINYXML2_LIB XMLHandle
{
public:
    explicit XMLHandle( XMLNode* node ) : _node( node ) {
    }
    explicit XMLHandle( XMLNode& node ) : _node( &node ) {
    }
    XMLHandle( const XMLHandle& ref ) : _node( ref._node ) {
    }
    XMLHandle& operator=( const XMLHandle& ref )                            {
        _node = ref._node;
        return *this;
    }

    XMLHandle FirstChild()                                                  {
        return XMLHandle( _node ? _node->FirstChild() : 0 );
    }
    XMLHandle FirstChildElement( const char* name = 0 )                     {
        return XMLHandle( _node ? _node->FirstChildElement( name ) : 0 );
    }
    XMLHandle LastChild()                                                   {
        return XMLHandle( _node ? _node->LastChild() : 0 );
    }
    XMLHandle LastChildElement( const char* name = 0 )                      {
        return XMLHandle( _node ? _node->LastChildElement( name ) : 0 );
    }
    XMLHandle PreviousSibling()                                             {
        return XMLHandle( _node ? _node->PreviousSibling() : 0 );
    }
    XMLHandle PreviousSiblingElement( const char* name = 0 )                {
        return XMLHandle( _node ? _node->PreviousSiblingElement( name ) : 0 );
    }
    XMLHandle NextSibling()                                                 {
        return XMLHandle( _node ? _node->NextSibling() : 0 );
    }
    XMLHandle NextSiblingElement( const char* name = 0 )                    {
        return XMLHandle( _node ? _node->NextSiblingElement( name ) : 0 );
    }

    XMLNode* ToNode()                            {
        return _node;
    }
    XMLElement* ToElement()                      {
        return ( _node ? _node->ToElement() : 0 );
    }
    XMLText* ToText()                            {
        return ( _node ? _node->ToText() : 0 );
    }
    XMLUnknown* ToUnknown()                      {
        return ( _node ? _node->ToUnknown() : 0 );
    }
    XMLDeclaration* ToDeclaration()              {
        return ( _node ? _node->ToDeclaration() : 0 );
    }

private:
    XMLNode* _node;
};


class TINYXML2_LIB XMLConstHandle
{
public:
    explicit XMLConstHandle( const XMLNode* node ) : _node( node ) {
    }
    explicit XMLConstHandle( const XMLNode& node ) : _node( &node ) {
    }
    XMLConstHandle( const XMLConstHandle& ref ) : _node( ref._node ) {
    }

    XMLConstHandle& operator=( const XMLConstHandle& ref )                          {
        _node = ref._node;
        return *this;
    }

    const XMLConstHandle FirstChild() const                                         {
        return XMLConstHandle( _node ? _node->FirstChild() : 0 );
    }
    const XMLConstHandle FirstChildElement( const char* name = 0 ) const                {
        return XMLConstHandle( _node ? _node->FirstChildElement( name ) : 0 );
    }
    const XMLConstHandle LastChild()    const                                       {
        return XMLConstHandle( _node ? _node->LastChild() : 0 );
    }
    const XMLConstHandle LastChildElement( const char* name = 0 ) const             {
        return XMLConstHandle( _node ? _node->LastChildElement( name ) : 0 );
    }
    const XMLConstHandle PreviousSibling() const                                    {
        return XMLConstHandle( _node ? _node->PreviousSibling() : 0 );
    }
    const XMLConstHandle PreviousSiblingElement( const char* name = 0 ) const       {
        return XMLConstHandle( _node ? _node->PreviousSiblingElement( name ) : 0 );
    }
    const XMLConstHandle NextSibling() const                                        {
        return XMLConstHandle( _node ? _node->NextSibling() : 0 );
    }
    const XMLConstHandle NextSiblingElement( const char* name = 0 ) const           {
        return XMLConstHandle( _node ? _node->NextSiblingElement( name ) : 0 );
    }


    const XMLNode* ToNode() const               {
        return _node;
    }
    const XMLElement* ToElement() const         {
        return ( _node ? _node->ToElement() : 0 );
    }
    const XMLText* ToText() const               {
        return ( _node ? _node->ToText() : 0 );
    }
    const XMLUnknown* ToUnknown() const         {
        return ( _node ? _node->ToUnknown() : 0 );
    }
    const XMLDeclaration* ToDeclaration() const {
        return ( _node ? _node->ToDeclaration() : 0 );
    }

private:
    const XMLNode* _node;
};


class TINYXML2_LIB XMLPrinter : public XMLVisitor
{
public:
    enum EscapeAposCharsInAttributes {
        ESCAPE_APOS_CHARS_IN_ATTRIBUTES,
        DONT_ESCAPE_APOS_CHARS_IN_ATTRIBUTES
    };

    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 );
    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 );

    void PushText( const char* text, bool cdata=false );
    void PushText( int value );
    void PushText( unsigned value );
    void PushText( int64_t value );
    void PushText( uint64_t value );
    void PushText( bool value );
    void PushText( float value );
    void PushText( double value );

    void PushComment( const char* comment );

    void PushDeclaration( const char* value );
    void PushUnknown( const char* value );

    virtual bool VisitEnter( const XMLDocument& /*doc*/ ) override;
    virtual bool VisitExit( const XMLDocument& /*doc*/ ) override   {
        return true;
    }

    virtual bool VisitEnter( const XMLElement& element, const XMLAttribute* attribute ) override;
    virtual bool VisitExit( const XMLElement& element ) override;

    virtual bool Visit( const XMLText& text ) override;
    virtual bool Visit( const XMLComment& comment ) override;
    virtual bool Visit( const XMLDeclaration& declaration ) override;
    virtual bool Visit( const XMLUnknown& unknown ) override;

    const char* CStr() const {
        return _buffer.Mem();
    }
    size_t CStrSize() const {
        return _buffer.Size();
    }
    void ClearBuffer( bool resetToFirstElement = true ) {
        _buffer.Clear();
        _buffer.Push(0);
        _firstElement = resetToFirstElement;
    }

protected:
    virtual bool CompactMode( const XMLElement& )   { return _compactMode; }

    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 );

    inline void Write(const char* data) { Write(data, strlen(data)); }

    void SealElementIfJustOpened();
    bool _elementJustOpened;
    DynArray< const char*, 10 > _stack;

private:
    void PrepareForNewNode( bool compactMode );
    void PrintString( const char*, bool restrictedEntitySet );  // prints out, after detecting entities.

    bool _firstElement;
    FILE* _fp;
    int _depth;
    int _textDepth;
    bool _processEntities;
    bool _compactMode;

    enum {
        ENTITY_RANGE = 64,
        BUF_SIZE = 200
    };
    bool _entityFlag[ENTITY_RANGE];
    bool _restrictedEntityFlag[ENTITY_RANGE];

    DynArray< char, 20 > _buffer;

    // Prohibit cloning, intentionally not implemented
    XMLPrinter( const XMLPrinter& );
    XMLPrinter& operator=( const XMLPrinter& );
};


} // namespace tinyxml2

#if defined(_MSC_VER)
#   pragma warning(pop)
#endif

#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.

```
#pragma once

#include <iostream>
#include <string>


struct Utils {
    static std::string zeroPadding(uint16_t value, std::size_t width) {
        std::string s = std::to_string(value);
        if (s.length() < width) s = std::string(width - s.length(), '0') + s;
        return s;
    };

    static std::string extractAttribute(const std::string& line, const std::string& attr) {
        std::string key = attr + "=\"";
        size_t start = line.find(key);

        if (start == std::string::npos) return "";

        start += key.size();
        size_t end = line.find('"', start);

        return line.substr(start, end - start);
    }

    static bool hasPattern(const std::string &str, const std::string &pattern) {
        return str.find(pattern) != std::string::npos;
    }

    static void clear() {
        #ifdef _WIN32
            system("cls");
        #else
            system("clear");
        #endif
    }
};

```

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 <cstddef>
#include <cstdarg>
```

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
```