SpaceWalk_TheGame

0.1

Generated by Doxygen 1.9.6

Chapter 1

SpaceWalk_TheGame

A Roleplay Space adventure game. You can move room to room, collect items and open new paths to explore the game world. Meet and interact with NPCs.

1.1 Business Requirements

1. Command line user interface
2. Load story from file
3. User choices change storyline
4. User can interact with ∗∗NPC∗∗s
5. User can interact with items
6. (8bit Graphics)

1.2 Technical Requirements

1. Create class hierarchy / architecture.
2. Define Use Cases.
3. Define Activities with diagrams.

1.3 Architecture

1.3.1 Class Architecture

1.3.2 Component Architecure

1.3.3 World Initialization Sequence

1.3.4 Functional Use Case diagram

1.3.5 Game Loop Activity Diagram

1.4 References

TinyXML2

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

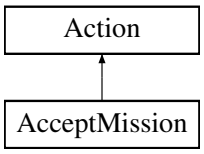
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 AcceptMission Class Reference

Inheritance diagram for AcceptMission:



Public Member Functions

- AcceptMission (const std::string &, World &)

  Construct a new Accept Mission object.
- void doAction (Mission &)

  Accept mission given in args.
- void operator() (Mission &)

  The AcceptMission object is a callable object, and will call the doAction method of the object.

Additional Inherited Members

5.1.1 Constructor & Destructor Documentation

5.1.1.1 AcceptMission()

```
AcceptMission::AcceptMission (
            const std::string & desc,
            World & gm )
```

Construct a new Accept Mission object.

Parameters

desc	(std::string): Description of the action.
gm	(World&): The game world, so the action class can access the ongoing games attributes.

5.1.2 Member Function Documentation

5.1.2.1 doAction()

```
void AcceptMission::doAction (
            Mission & m )
```

Accept mission given in args.

Parameters

m	(Mission&): Mission to accept.

5.1.2.2 operator()()

```
void AcceptMission::operator() (
            Mission & m )
```

The AcceptMission object is a callable object, and will call the doAction method of the object.

Parameters

m	(Mission&): Mission to accept.

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

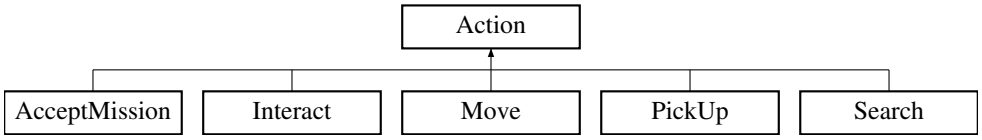
- src/interface.hpp
- src/interface.cpp

5.2 Action Class Reference

Abstract class representing a player action.

`#include <interface.hpp>`

Inheritance diagram for Action:



Public Member Functions

- Action (const std::string &, World &)

  Construct a new Action object.
- virtual void doAction ()

  Take action with the player character.
- std::string getDescription ()

  Get the Description object.
- virtual void operator() ()=0

  The object created from the Action class, can be called like a function.

Protected Attributes

- World & **game_world**

5.2.1 Detailed Description

Abstract class representing a player action.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Action()

```
Action::Action (
            const std::string & desc,
            World & gm )
```

Construct a new Action object.

Parameters

desc	(std::string): Description of the action.
gm	(World&): The game world, so the action class can access the ongoing games attributes.

5.2.3 Member Function Documentation

5.2.3.1 doAction()

```
virtual void Action::doAction ( )  [virtual]
```

Take action with the player character.

5.2.3.2 getDescription()

```
std::string Action::getDescription ( )
```

Get the Description object.

Returns

std::string

5.2.3.3 operator()()

```
virtual void Action::operator() ( )  [pure virtual]
```

The object created from the Action class, can be called like a function.

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

- src/interface.hpp
- src/interface.cpp

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

Public Member Functions

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

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

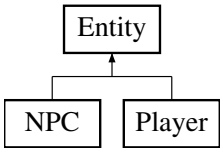
- src/tinyxml2.h

5.4 Entity Class Reference

Base class for a NPC, USER or any other Entity living in the game world.

`#include <engine.hpp>`

Inheritance diagram for Entity:



Public Member Functions

- Entity (const std::string &n)

  Construct a new Entity object.
- std::string getName () const

  Get the Name of the Entity.
- Entity & addItem (item &i)

  Add Item to Entity's inventory.
- Entity & addItems (items &i)

  Add multiple items from an item vector to the entity's inventory.
- items & getInventory ()

  Get the Inventory object.

Protected Attributes

- const std::string **name**
- items **inventory**
- int **hp**
- int **stamina**
- int **intelligence**
- int **strenght**

5.4.1 Detailed Description

Base class for a NPC, USER or any other Entity living in the game world.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Entity()

```
Entity::Entity (
            const std::string & n )  [inline]
```

Construct a new Entity object.

Parameters

n	(std::string)

5.4.3 Member Function Documentation

5.4.3.1 addItem()

```
Entity & Entity::addItem (
            item & i )  [inline]
```

Add Item to Entity's inventory.

Parameters

i	(item&): Item to move to Entity's inventory

Returns

Entity&

5.4.3.2 addItems()

```
Entity & Entity::addItems (
            items & i )  [inline]
```

Add multiple items from an item vector to the entity's inventory.

Parameters

i	

Returns

Entity&

5.4.3.3 getInventory()

```
items & Entity::getInventory ( )  [inline]
```

Get the Inventory object.

Returns

items&

5.4.3.4 getName()

```
std::string Entity::getName ( ) const  [inline]
```

Get the Name of the Entity.

Returns

std::string

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

- src/engine.hpp

5.5 tinyxml2::Entity Struct Reference

Public Attributes

- const char * **pattern**
- int **length**
- char **value**

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

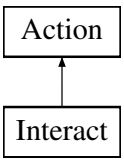
- src/tinyxml2.cpp

5.6 Interact Class Reference

Realisation of the action class, interaction with an NPC.

```
#include <interface.hpp>
```

Inheritance diagram for Interact:



Public Member Functions

- Interact (const std::string &, World &)

  Construct a new Action object.
- missions doAction (npc &)

  Interact with the npc.
- void acceptMission (npc &)

  Accept NPC's mission.
- void operator() (npc &)

  The Interact object is a callable object, and will call the doAction method of the object.

Additional Inherited Members

5.6.1 Detailed Description

Realisation of the action class, interaction with an NPC.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Interact()

```
Interact::Interact (
            const std::string & desc,
            World & gm )
```

Construct a new Action object.

Parameters

desc	(std::string): Description of the action.
gm	(World&): The game world, so the action class can access the ongoing games attributes.

5.6.3 Member Function Documentation

5.6.3.1 acceptMission()

```
void Interact::acceptMission (
            npc &  )
```

Accept NPC's mission.

Parameters

npc	(npc&): The smart pointer of the npc that will be talked to.

5.6.3.2 doAction()

```
missions Interact::doAction (
            npc & npc )
```

Interact with the npc.

Parameters

npc	(npc&): The smart pointer of the npc that will be talked to.

5.6.3.3 operator()()

```
void Interact::operator() (
            npc &  )
```

The Interact object is a callable object, and will call the doAction method of the object.

Parameters

npc	(npc&): The smart pointer of the npc that will be talked to.

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

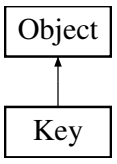
- src/interface.hpp
- src/interface.cpp

5.7 Key Class Reference

An object that can open a room.

```
#include <engine.hpp>
```

Inheritance diagram for Key:



Public Member Functions

- Key (const int kid, const std::string &n, const int id, const std::string &d)

  Construct a new Key object.
- int getKeyID ()

  Get the KeyID of the key.

5.7.1 Detailed Description

An object that can open a room.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Key()

```
Key::Key (
            const int kid,
            const std::string & n,
            const int id,
            const std::string & d )  [inline]
```

Construct a new Key object.

Parameters

kid	The key id should be identical to the room's it opens
n	The name of the key
id	The uniqe item id of the key
d	A description that describes the keys look, use, etc.

5.7.3 Member Function Documentation

5.7.3.1 getKeyID()

```
int Key::getKeyID ( )  [inline]
```

Get the KeyID of the key.

Returns

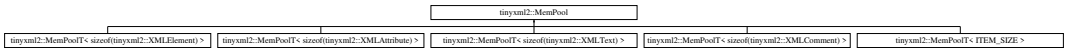
int

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

- src/engine.hpp

5.8 tinyxml2::MemPool Class Reference

Inheritance diagram for tinyxml2::MemPool:



Public Member Functions

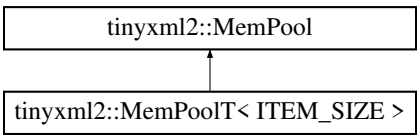
- virtual int **ItemSize** () const =0
- virtual void ∗ **Alloc** ()=0
- virtual void **Free** (void ∗)=0
- virtual void **SetTracked** ()=0

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

- src/tinyxml2.h

5.9 tinyxml2::MemPoolT< ITEM_SIZE > Class Template Reference

Inheritance diagram for tinyxml2::MemPoolT< ITEM_SIZE >:



Public Types

- enum { **ITEMS_PER_BLOCK** = (4 ∗ 1024) / ITEM_SIZE }

Public Member Functions

- void **Clear** ()
- virtual int ItemSize () const
- int **CurrentAllocs** () const
- virtual void ∗ Alloc ()
- virtual void Free (void ∗mem)
- void **Trace** (const char ∗name)
- void SetTracked ()
- int **Untracked** () const

5.9.1 Member Function Documentation

5.9.1.1 Alloc()

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

Implements tinyxml2::MemPool.

5.9.1.2 Free()

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

Implements tinyxml2::MemPool.

5.9.1.3 ItemSize()

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

Implements tinyxml2::MemPool.

5.9.1.4 SetTracked()

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

Implements tinyxml2::MemPool.

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

- src/tinyxml2.h

5.10 Mission Class Reference

class Mission helps to create a mission system, that will give objectives, to accomplish, for the player. This will give the player a direction, how to finish the story.

```
#include <engine.hpp>
```

Public Member Functions

- Mission (const std::string &desc)

  Construct a new Mission object.
- int getTargetRoom ()

  Get the Target Room ID.
- Mission & setTargetRoom (int targetRoomID)

  Set the Target Room ID.
- int getTargetItem ()

  Get the Target Item ID.
- Mission & setTargetItem (int targetItemID)

  Set the Target Item ID.
- bool checkStatus (Player &)

  Check if mission is accomplished.
- Mission & startMission ()

  Change mission status to active.

5.10.1 Detailed Description

class Mission helps to create a mission system, that will give objectives, to accomplish, for the player. This will give the player a direction, how to finish the story.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Mission()

```
Mission::Mission (
            const std::string & desc )  [inline]
```

Construct a new Mission object.

Parameters

desc	Description of the mission.

5.10.3 Member Function Documentation

5.10.3.1 checkStatus()

```
bool Mission::checkStatus (
            Player & player )
```

Check if mission is accomplished.

Parameters

player	

Returns

true

false

5.10.3.2 getTargetItem()

```
int Mission::getTargetItem ( )  [inline]
```

Get the Target Item ID.

Returns

int

5.10.3.3 getTargetRoom()

```
int Mission::getTargetRoom ( )  [inline]
```

Get the Target Room ID.

Returns

int

5.10.3.4 setTargetItem()

```
Mission & Mission::setTargetItem (
            int targetItemID )  [inline]
```

Set the Target Item ID.

Parameters

targetItemID	

Returns

Mission&

5.10.3.5 setTargetRoom()

```
Mission & Mission::setTargetRoom (
            int targetRoomID )  [inline]
```

Set the Target Room ID.

Parameters

targetRoomID	

Returns

Mission&

5.10.3.6 startMission()

```
Mission & Mission::startMission ( )  [inline]
```

Change mission status to active.

Returns

Mission&

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

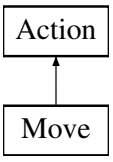
- src/engine.hpp
- src/engine.cpp

5.11 Move Class Reference

Realisation of the action class, representing movement from a room to another.

```
#include <interface.hpp>
```

Inheritance diagram for Move:



Public Member Functions

- Move (const std::string &, World &)

  Construct a new Action object.
- void doAction (node &)

  Take action with the player character and move to another room.
- void operator() (node &)

  The Move object is a callable object, and will call the doAction method of the object.

Additional Inherited Members

5.11.1 Detailed Description

Realisation of the action class, representing movement from a room to another.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Move()

```
Move::Move (
            const std::string & desc,
            World & gm )
```

Construct a new Action object.

Parameters

desc	(std::string): Description of the action.
gm	(World&): The game world, so the action class can access the ongoing games attributes.

5.11.3 Member Function Documentation

5.11.3.1 doAction()

```
void Move::doAction (
            node & r )
```

Take action with the player character and move to another room.

Parameters

r	(node&): The room that the player will be moved to.

5.11.3.2 operator()()

```
void Move::operator() (
            node & r )
```

The Move object is a callable object, and will call the doAction method of the object.

Parameters

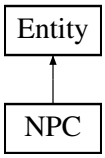
r	(node&): The room that the player will be moved to.

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

- src/interface.hpp
- src/interface.cpp

5.12 NPC Class Reference

Inheritance diagram for NPC:



Public Member Functions

- NPC (const std::string &n, const std::string &d, missions m)

  Construct a new NPC object, giving it a dialog and missions that can be accepted by the player.
- missions & **getMissions** ()
- std::string **getDialog** ()

Additional Inherited Members

5.12.1 Constructor & Destructor Documentation

5.12.1.1 NPC()

```
NPC::NPC (
            const std::string & n,
            const std::string & d,
            missions m )  [inline]
```

Construct a new NPC object, giving it a dialog and missions that can be accepted by the player.

Parameters

n	
d	
m	

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

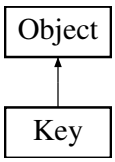
- src/engine.hpp

5.13 Object Class Reference

Base class for any object that can be owned by an Entity or Room.

`#include <engine.hpp>`

Inheritance diagram for Object:



Public Member Functions

- Object (const std::string &n, const int id, const std::string &d)

  Construct a new Object object.
- std::string getName () const

  Get the Name object.
- int const getID ()

  Get the ID of the object.
- std::string getDescription () const

  Get the Description of the object.

5.13.1 Detailed Description

Base class for any object that can be owned by an Entity or Room.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 Object()

```
Object::Object (
            const std::string & n,
            const int id,
            const std::string & d )  [inline]
```

Construct a new Object object.

Parameters

n	(std::string&) Name of the Object
d	(std::string&): Description of the object
id	(int): Creation ID of the item

5.13.3 Member Function Documentation

5.13.3.1 getDescription()

```
std::string Object::getDescription ( ) const  [inline]
```

Get the Description of the object.

Returns

objectDescription (std::string)

5.13.3.2 getID()

```
int const Object::getID ( )  [inline]
```

Get the ID of the object.

Returns

objID (int)

5.13.3.3 getName()

```
std::string Object::getName ( ) const  [inline]
```

Get the Name object.

Returns

objectName (std::string)

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

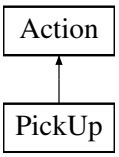
- src/engine.hpp

5.14 PickUp Class Reference

Realisation of the action class, the player picking up an item.

`#include <interface.hpp>`

Inheritance diagram for PickUp:



Public Member Functions

- PickUp (const std::string &, World &)
  Construct a new Action object.
- void doAction (item &)
  Take action with the player character and pick up the desired item and place it into the player's inventory.
- void operator() (item &)
  The PickUp object is a callable object, and will call the doAction method of the object.

Additional Inherited Members

5.14.1 Detailed Description

Realisation of the action class, the player picking up an item.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 PickUp()

```
PickUp::PickUp (
            const std::string & desc,
            World & gm )
```

Construct a new Action object.

Parameters

desc	(std::string): Description of the action.
gm	(World&): The game world, so the action class can access the ongoing games attributes.

5.14.3 Member Function Documentation

5.14.3.1 doAction()

```
void PickUp::doAction (
             item & i )
```

Take action with the player character and pick up the desired item and place it into the player's inventory.

Parameters

i	(item&): The smart pointer's reference of the item that will be picked up and placed into the player's inventory.

5.14.3.2 operator()()

```
void PickUp::operator() (
             item & i )
```

The PickUp object is a callable object, and will call the doAction method of the object.

Parameters

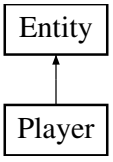
i	(item&): The smart pointer's reference of the item that will be picked up and placed into the player's inventory.

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

- src/interface.hpp
- src/interface.cpp

5.15 Player Class Reference

Inheritance diagram for Player:



Public Member Functions

- Player ()

  Construct a new Player object with Default values.
- Player (const std::string &n, Room ∗init_location)

  Construct a new Player object, giving it an initial location.
- Room ∗ getLocation ()

  Get the Location of the Entity by returning the pointer to the Room.
- Entity & setLocation (Room ∗room_ptr)

  Set the Location of the Entity by giving the pointer of the Room.

Additional Inherited Members

5.15.1 Constructor & Destructor Documentation

5.15.1.1 Player() [1/2]

```
Player::Player ( )  [inline]
```

Construct a new Player object with Default values.

5.15.1.2 Player() [2/2]

```
Player::Player (
            const std::string & n,
            Room * init_location )  [inline]
```

Construct a new Player object, giving it an initial location.

Parameters

n	
init_location	

5.15.2 Member Function Documentation

5.15.2.1 getLocation()

```
Room * Player::getLocation ( )  [inline]
```

Get the Location of the Entity by returning the pointer to the Room.

Returns

Room*

5.15.2.2 setLocation()

```
Entity & Player::setLocation (
            Room * room_ptr )  [inline]
```

Set the Location of the Entity by giving the pointer of the Room.

Parameters

room_ptr	

Returns

Entity&

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

- src/engine.hpp

5.16 Room Class Reference

Part of the World. A room that contains items, that can be collected, players or npc can move in and out of these rooms.

```
#include <engine.hpp>
```

Public Member Functions

- Room (const std::string &n, int id, const std::string &desc)
  Construct a new Room object.
- Room (const std::string &n, int id, const std::string &desc, neighbours cn)
  Construct a new Room object.
- std::string getName () const
  Get the Name of the Room.
- int getID () const
  Get the roomID of the Room.
- std::string getDescription () const
  Get the Description of the Room.
- neighbours getNeighbours () const
  Get the Neighbours object.
- Room & addNeighbour (int rid)
  Add a neighbour to the Neighours object.

- Room & addNeighbours (neighbours rids)

  Add multiple neighbours to the Neighbours object.
- items & getItems ()

  Get the Items object.
- Room & addItem (item &i)

  Add new item to the Inventory of the Room.
- Room & addItems (items &inv)

  Add new Items to the Inventory of the Room.
- Room & addEntity (npc &e)

  Add entity to the population of the room.
- Room & addEntities (npcs &ents)

  Add a bunch of entities to the population of the room.
- Room & setLock (LockStatus stat)

  Set the Lock object.
- bool isLocked ()

  Check wheter room is locked or not. Returns true if locked, false otherwise.
- const npcs & getPopulation ()

  Get the Population object.
- ~Room ()

  Destroy the Room object. Reset all nodes in the neighours vector and all items in inventory.

Static Public Member Functions

- static bool unlock (item &, node &)

  With the right key the room given in the argument can be unlocked. If the key fits the door, after it unlocked the room, it cant be used anymore.

5.16.1 Detailed Description

Part of the World. A room that contains items, that can be collected, players or npc can move in and out of these rooms.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 Room() [1/2]

```
Room::Room (
            const std::string & n,
            int id,
            const std::string & desc )  [inline]
```

Construct a new Room object.

Parameters

n	(std::string&): Name of the Room that cant be changed later.
id	(int): This have to be a unique ID to be able to connect keys with rooms.
desc	(std::string&): Description of the room. Cant be changed later.

5.16.2.2 Room() [2/2]

```
Room::Room (
            const std::string & n,
            int id,
            const std::string & desc,
            neighbours cn )  [inline]
```

Construct a new Room object.

Parameters

n	(std::string&): Name of the Room that cant be changed later.
id	(int): This have to be a unique ID to be able to connect keys with rooms.
desc	(std::string&): Description of the room. Cant be changed later.
cn	Connected neighbours

5.16.2.3 ~Room()

```
Room::~Room ( )  [inline]
```

Destroy the Room object. Reset all nodes in the neighours vector and all items in inventory.

5.16.3 Member Function Documentation

5.16.3.1 addEntities()

```
Room & Room::addEntities (
            npcs & ents )  [inline]
```

Add a bunch of entities to the population of the room.

Parameters

ents	

Returns

Room&

5.16.3.2 addEntity()

```
Room & Room::addEntity (
            npc & e )  [inline]
```

Add entity to the population of the room.

Parameters

e	

Returns

Room&

5.16.3.3 addItem()

```
Room & Room::addItem (
            item & i )  [inline]
```

Add new item to the Inventory of the Room.

Parameters

i	(item&) new Item

Returns

Room&

5.16.3.4 addItems()

```
Room & Room::addItems (
            items & inv )  [inline]
```

Add new Items to the Inventory of the Room.

Parameters

inv	(items&) Vector of Items to be added to the Inventory of the Room

Returns

Room&

5.16.3.5 addNeighbour()

```
Room & Room::addNeighbour (
            int rid )  [inline]
```

Add a neighbour to the Neighours object.

Parameters

nn	(node&) A new Room, that will be added to the Neighbours object.

Returns

Room&

5.16.3.6 addNeighbours()

```
Room & Room::addNeighbours (
            neighbours rids )  [inline]
```

Add multiple neighbours to the Neighbours object.

Parameters

ns	(nodes&) Vector of Neighbours object

Returns

Room&

5.16.3.7 getDescription()

```
std::string Room::getDescription ( ) const  [inline]
```

Get the Description of the Room.

Returns

description (std::string)

5.16.3.8 getID()

```
int Room::getID ( ) const  [inline]
```

Get the roomID of the Room.

Returns

roomID (int)

5.16.3.9 getItems()

```
items & Room::getItems ( )  [inline]
```

Get the Items object.

Returns

items const&

5.16.3.10 getName()

```
std::string Room::getName ( ) const  [inline]
```

Get the Name of the Room.

Returns

roomName (std::string)

5.16.3.11 getNeighbours()

```
neighbours Room::getNeighbours ( ) const  [inline]
```

Get the Neighbours object.

Returns

neighbours (const nodes)

5.16.3.12 getPopulation()

```
const npcs & Room::getPopulation ( )  [inline]
```

Get the Population object.

Returns

const entities&

5.16.3.13 isLocked()

```
bool Room::isLocked ( )  [inline]
```

Check wheter room is locked or not. Returns true if locked, false otherwise.

Returns

true

false

5.16.3.14 setLock()

```
Room & Room::setLock (
            LockStatus stat )  [inline]
```

Set the Lock object.

Parameters

stat	

Returns

Room&

5.16.3.15 unlock()

```
bool Room::unlock (
            item & k,
            node & r )  [static]
```

With the right key the room given in the argument can be unlocked. If the key fits the door, after it unlocked the room, it cant be used anymore.

Parameters

k	key to unlock the room
r	room to be unlocked

Returns

true

false

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

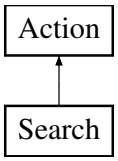
- src/engine.hpp
- src/engine.cpp

5.17 Search Class Reference

Realisation of the action class, respresenting the search of a room.

`#include <interface.hpp>`

Inheritance diagram for Search:



Public Member Functions

- Search (const std::string &, World &)

  Construct a new Action object.
- search_results doAction (Room ∗)

  Take action with player character and search through the room, that the player is in.
- search_results operator() (Room ∗)

  The Search object is a callable object, and will call the doAction method of the object.

Additional Inherited Members

5.17.1 Detailed Description

Realisation of the action class, respresenting the search of a room.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 Search()

```
Search::Search (
            const std::string & desc,
            World & gm )
```

Construct a new Action object.

Parameters

desc	(std::string): Description of the action.
gm	(World&): The game world, so the action class can access the ongoing games attributes.

5.17.3 Member Function Documentation

5.17.3.1 doAction()

```
search_results Search::doAction (
            Room * r )
```

Take action with player character and search through the room, that the player is in.

Parameters

r	(Room∗): The pointer to the room, that the player is in. This is a pointer because if the player object has a location attribute that stores the Room's pointer.

Returns

search_results

5.17.3.2 operator()()

```
search_results Search::operator() (
            Room * r )
```

The Search object is a callable object, and will call the doAction method of the object.

Parameters

r	(Room∗): The pointer to the room, that the player is in. This is a pointer because if the player object has a location attribute that stores the Room's pointer.

Returns

search_results

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

- src/interface.hpp
- src/interface.cpp

5.18 tinyxml2::StrPair Class Reference

Public Types

- enum **Mode** {
  NEEDS_ENTITY_PROCESSING = 0x01 , **NEEDS_NEWLINE_NORMALIZATION** = 0x02 , **NEEDS_↩**
  WHITESPACE_COLLAPSING = 0x04 , **TEXT_ELEMENT** = NEEDS_ENTITY_PROCESSING | NEEDS_↩
  NEWLINE_NORMALIZATION ,
  TEXT_ELEMENT_LEAVE_ENTITIES = NEEDS_NEWLINE_NORMALIZATION , **ATTRIBUTE_NAME** =
  0 , **ATTRIBUTE_VALUE** = NEEDS_ENTITY_PROCESSING | NEEDS_NEWLINE_NORMALIZATION ,
  ATTRIBUTE_VALUE_LEAVE_ENTITIES = NEEDS_NEWLINE_NORMALIZATION ,
  COMMENT = NEEDS_NEWLINE_NORMALIZATION }

Public Member Functions

- 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** ()

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

- src/tinyxml2.h
- src/tinyxml2.cpp

5.19 World Class Reference

The world that contains and manages all the rooms, entities. A World object will able to parse the story file and initialize the game and run it.

```
#include <engine.hpp>
```

Public Member Functions

- World (const char ∗path2story)

  Construct a new World object.
- nodes & getWorldRooms ()

  Get the World Rooms object.
- missions & getWorldMission ()

  Get the World Mission object.
- void RoomFactory (const std::string &, int, const std::string &, neighbours)

  Create Room with initial params and add it to worldRooms.
- tinyxml2::XMLDocument & getStory ()

  Get the Story object.

- items makeInventory (tinyxml2::XMLElement ∗)

  create inventory
- neighbours parseConnections (tinyxml2::XMLElement ∗)

  Load neigbouring rooms id's to connection map.
- void loadRooms (tinyxml2::XMLElement ∗)

  This function iterates through the room elements of the world element in the xml file and constructs the rooms of the world.
- void loadEntities (tinyxml2::XMLElement ∗, node &)

  Load entities from xml doc.
- Mission makeMission (tinyxml2::XMLElement ∗)

  Construct a new mission with a description, then set mission targets.
- void loadWorldMissions (tinyxml2::XMLElement ∗)

  Load world story missions from xml doc.
- missions loadNPCMissions (tinyxml2::XMLElement ∗)

  Load missions of an NPC.
- void initWorld (const char ∗)

  Initialize world with the xml story file.
- void enterRoom (node &r)

  move entity to room
- Player & getPlayer ()

  Get the Player object.
- void startMission (Mission &m)

  Start mission by changing mission's status to in progress and placing it to the active_missions vector.
- void destroyWorld ()

  Free resources used by world.

5.19.1 Detailed Description

The world that contains and manages all the rooms, entities. A World object will able to parse the story file and initialize the game and run it.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 World()

```
World::World (
            const char * path2story )  [inline]
```

Construct a new World object.

Parameters

path2story	(const char∗) The path to the story xml file.

5.19.3 Member Function Documentation

5.19.3.1 destroyWorld()

```
void World::destroyWorld ( )  [inline]
```

Free resources used by world.

5.19.3.2 enterRoom()

```
void World::enterRoom (
            node & r )  [inline]
```

move entity to room

Parameters

player	
room	

5.19.3.3 getPlayer()

```
Player & World::getPlayer ( )  [inline]
```

Get the Player object.

Returns

Player&

5.19.3.4 getStory()

```
tinyxml2::XMLDocument & World::getStory ( )  [inline]
```

Get the Story object.

Returns

tinyxml2::XMLDocument&

5.19.3.5 getWorldMission()

```
missions & World::getWorldMission ( )  [inline]
```

Get the World Mission object.

Returns

missions&

5.19.3.6 getWorldRooms()

```
nodes & World::getWorldRooms ( )  [inline]
```

Get the World Rooms object.

Returns

nodes

5.19.3.7 initWorld()

```
void World::initWorld (
            const char * path2story )
```

Initialize world with the xml story file.

Parameters

path2story	

5.19.3.8 loadEntities()

```
void World::loadEntities (
            tinyxml2::XMLElement * firstEle,
            node & r )
```

Load entities from xml doc.

Parameters

firstEle	

5.19.3.9 loadNPCMissions()

```
missions World::loadNPCMissions (
            tinyxml2::XMLElement * missionsEle )
```

Load missions of an NPC.

Parameters

missionsEle	

Returns

missions

5.19.3.10 loadRooms()

```
void World::loadRooms (
            tinyxml2::XMLElement * firstRoom )
```

This function iterates through the room elements of the world element in the xml file and constructs the rooms of the world.

Parameters

firstRoom	

5.19.3.11 loadWorldMissions()

```
void World::loadWorldMissions (
            tinyxml2::XMLElement * missionsEle )
```

Load world story missions from xml doc.

Parameters

firstEle	first mission element in xml doc.

5.19.3.12 makeInventory()

```
items World::makeInventory (
            tinyxml2::XMLElement * invEle )
```

create inventory

Parameters

invEle	(tinyxml2::XMLElement*)

Returns

items

5.19.3.13 makeMission()

```
Mission World::makeMission (
            tinyxml2::XMLElement * missionEle )
```

Construct a new mission with a description, then set mission targets.

Parameters

missionEle	

Returns

Mission

5.19.3.14 parseConnections()

```
neighbours World::parseConnections (
            tinyxml2::XMLElement * conns )
```

Load neigbouring rooms id's to connection map.

Parameters

conns	

Returns

std::vector<int>

5.19.3.15 RoomFactory()

```
void World::RoomFactory (
            const std::string & name,
            int id,
            const std::string & desc,
            neighbours nids )
```

Create Room with initial params and add it to worldRooms.

Parameters

title	(const std::string&): Name of the Room
id	(const std::string&): ID of the Room
desc	(const std::string&): Description of the Room

5.19.3.16 startMission()

```
void World::startMission (
            Mission & m )  [inline]
```

Start mission by changing mission's status to in progress and placing it to the active_missions vector.

Parameters

m	

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

- src/engine.hpp
- src/engine.cpp

5.20 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**

5.20.1 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.

5.20.2 Member Function Documentation

5.20.2.1 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.

5.20.2.2 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.

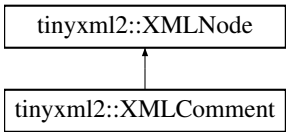
The documentation for this class was generated from the following files:

- src/tinyxml2.h
- src/tinyxml2.cpp

5.21 tinyxml2::XMLComment Class Reference

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

Inheritance diagram for tinyxml2::XMLComment:



Public Member Functions

- virtual XMLComment * ToComment ()

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

Protected Member Functions

- **XMLComment** (XMLDocument *doc)
- char * ParseDeep (char *p, StrPair *parentEndTag, int *curLineNumPtr)

Friends

- class **XMLDocument**

Additional Inherited Members

5.21.1 Detailed Description

An XML Comment.

5.21.2 Member Function Documentation

5.21.2.1 Accept()

```
bool tinyxml2::XMLComment::Accept (
            XMLVisitor * visitor ) const  [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.

5.21.2.2 ParseDeep()

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

Reimplemented from tinyxml2::XMLNode.

5.21.2.3 ShallowClone()

```
XMLNode * tinyxml2::XMLComment::ShallowClone (
            XMLDocument * document ) const  [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.

5.21.2.4 ShallowEqual()

```
bool tinyxml2::XMLComment::ShallowEqual (
            const XMLNode * compare ) const  [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.

5.21.2.5 ToComment() [1/2]

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

Safely cast to a Comment, or null.

Reimplemented from tinyxml2::XMLNode.

5.21.2.6 ToComment() [2/2]

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

Reimplemented from tinyxml2::XMLNode.

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

- src/tinyxml2.h
- src/tinyxml2.cpp

5.22 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

5.22.1 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.

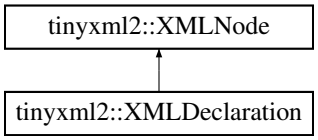
The documentation for this class was generated from the following file:

- src/tinyxml2.h

5.23 tinyxml2::XMLDeclaration Class Reference

`#include <tinyxml2.h>`

Inheritance diagram for tinyxml2::XMLDeclaration:



Public Member Functions

- virtual XMLDeclaration * ToDeclaration ()

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

Protected Member Functions

- **XMLDeclaration** (XMLDocument *doc)
- char * ParseDeep (char *p, StrPair *parentEndTag, int *curLineNumPtr)

Friends

- class **XMLDocument**

Additional Inherited Members

5.23.1 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.

5.23.2 Member Function Documentation

5.23.2.1 Accept()

```
bool tinyxml2::XMLDeclaration::Accept (
            XMLVisitor * visitor ) const  [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.

5.23.2.2 ParseDeep()

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

Reimplemented from tinyxml2::XMLNode.

5.23.2.3 ShallowClone()

```
XMLNode * tinyxml2::XMLDeclaration::ShallowClone (
            XMLDocument * document ) const  [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.

5.23.2.4 ShallowEqual()

```
bool tinyxml2::XMLDeclaration::ShallowEqual (
            const XMLNode * compare ) const  [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.

5.23.2.5 ToDeclaration() [1/2]

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

Safely cast to a Declaration, or null.

Reimplemented from tinyxml2::XMLNode.

5.23.2.6 ToDeclaration() [2/2]

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

Reimplemented from tinyxml2::XMLNode.

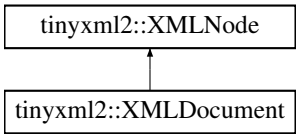
The documentation for this class was generated from the following files:

- src/tinyxml2.h
- src/tinyxml2.cpp

5.24 tinyxml2::XMLDocument Class Reference

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

Inheritance diagram for tinyxml2::XMLDocument:



Public Member Functions

- **XMLDocument** (bool processEntities=true, Whitespace whitespaceMode=PRESERVE_WHITESPACE)

  constructor
- virtual XMLDocument * ToDocument ()

  Safely cast to a Document, or null.
- virtual const XMLDocument * ToDocument () const
- 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
- 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)
- void **MarkInUse** (const XMLNode *const)
- virtual XMLNode * ShallowClone (XMLDocument *) const
- virtual bool ShallowEqual (const XMLNode *) 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

5.24.1 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.

5.24.2 Member Function Documentation

5.24.2.1 Accept()

```
bool tinyxml2::XMLDocument::Accept (
            XMLVisitor * visitor ) const  [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.

5.24.2.2 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.

5.24.2.3 DeleteNode()

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

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

5.24.2.4 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.

5.24.2.5 HasBOM()

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

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

5.24.2.6 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.

5.24.2.7 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.

5.24.2.8 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.

5.24.2.9 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"?>
```

5.24.2.10 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.

5.24.2.11 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.

5.24.2.12 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.

5.24.2.13 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.

5.24.2.14 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
```

5.24.2.15 RootElement()

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

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

5.24.2.16 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.

5.24.2.17 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.

5.24.2.18 SetBOM()

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

Sets whether to write the BOM when writing the file.

5.24.2.19 ShallowClone()

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

5.24.2.20 ShallowEqual()

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

5.24.2.21 ToDocument() [1/2]

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

Safely cast to a Document, or null.

Reimplemented from tinyxml2::XMLNode.

5.24.2.22 ToDocument() [2/2]

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

Reimplemented from tinyxml2::XMLNode.

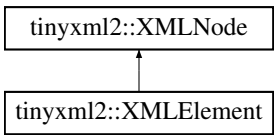
The documentation for this class was generated from the following files:

- src/tinyxml2.h
- src/tinyxml2.cpp

5.25 tinyxml2::XMLElement Class Reference

`#include <tinyxml2.h>`

Inheritance 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 ()

  Safely cast to an Element, or null.
- virtual const XMLElement * ToElement () const
- virtual bool Accept (XMLVisitor *visitor) const
- 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
- virtual bool ShallowEqual (const XMLNode *compare) const

Protected Member Functions

- char * ParseDeep (char *p, StrPair *parentEndTag, int *curLineNumPtr)

Friends

- class **XMLDocument**

Additional Inherited Members

5.25.1 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.

5.25.2 Member Function Documentation

5.25.2.1 Accept()

```
bool tinyxml2::XMLElement::Accept (
            XMLVisitor * visitor ) const  [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.

5.25.2.2 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();
}
```

5.25.2.3 DeleteAttribute()

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

Delete an attribute.

5.25.2.4 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 ".

5.25.2.5 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.

5.25.2.6 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()).

5.25.2.7 ParseDeep()

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

Reimplemented from tinyxml2::XMLNode.

5.25.2.8 QueryAttribute()

```
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
```

5.25.2.9 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
```

5.25.2.10 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.

5.25.2.11 SetText()

```
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>
```

5.25.2.12 ShallowClone()

```
XMLNode * tinyxml2::XMLElement::ShallowClone (
            XMLDocument * document ) const  [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.

5.25.2.13 ShallowEqual()

```
bool tinyxml2::XMLElement::ShallowEqual (
            const XMLNode * compare ) const  [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.

5.25.2.14 ToElement() [1/2]

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

Safely cast to an Element, or null.

Reimplemented from tinyxml2::XMLNode.

5.25.2.15 ToElement() [2/2]

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

Reimplemented from tinyxml2::XMLNode.

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

- src/tinyxml2.h
- src/tinyxml2.cpp

5.26 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.

5.26.1 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" ).FirstChildElemen
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.

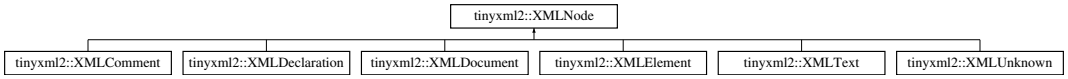
The documentation for this class was generated from the following file:

- src/tinyxml2.h

5.27 tinyxml2::XMLNode Class Reference

`#include <tinyxml2.h>`

Inheritance 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
- 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 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**

5.27.1 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)
```

5.27.2 Member Function Documentation

5.27.2.1 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::XMLText, tinyxml2::XMLComment, tinyxml2::XMLDeclaration, tinyxml2::XMLUnknown, tinyxml2::XMLElement, and tinyxml2::XMLDocument.

5.27.2.2 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 is 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()

5.27.2.3 DeleteChild()

```
void tinyxml2::XMLNode::DeleteChild (
            XMLNode * node )
```

Delete a child of this node.

5.27.2.4 DeleteChildren()

```
void tinyxml2::XMLNode::DeleteChildren ( )
```

Delete all the children of this node.

5.27.2.5 FirstChildElement()

```
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.

5.27.2.6 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.

5.27.2.7 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.

5.27.2.8 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.

5.27.2.9 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.

5.27.2.10 LastChildElement()

```
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.

5.27.2.11 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.

5.27.2.12 SetValue()

```
void tinyxml2::XMLNode::SetValue (
            const char * val,
            bool staticMem = false )
```

Set the Value of an XML node.

See also

Value()

5.27.2.13 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::XMLDocument, tinyxml2::XMLText, tinyxml2::XMLComment, tinyxml2::XMLDeclaration, tinyxml2::XMLUnknown, and tinyxml2::XMLElement.

5.27.2.14 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::XMLDocument, tinyxml2::XMLText, tinyxml2::XMLComment, tinyxml2::XMLDeclaration, tinyxml2::XMLUnknown, and tinyxml2::XMLElement.

5.27.2.15 ToComment()

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

Safely cast to a Comment, or null.

Reimplemented in tinyxml2::XMLComment.

5.27.2.16 ToDeclaration()

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

Safely cast to a Declaration, or null.

Reimplemented in tinyxml2::XMLDeclaration.

5.27.2.17 ToDocument()

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

Safely cast to a Document, or null.

Reimplemented in tinyxml2::XMLDocument.

5.27.2.18 ToElement()

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

Safely cast to an Element, or null.

Reimplemented in tinyxml2::XMLElement.

5.27.2.19 ToText()

```
virtual XMLText * tinyxml2::XMLNode::ToText ( )  [inline], [virtual]
```

Safely cast to Text, or null.

Reimplemented in tinyxml2::XMLText.

5.27.2.20 ToUnknown()

```
virtual XMLUnknown * tinyxml2::XMLNode::ToUnknown ( )  [inline], [virtual]
```

Safely cast to an Unknown, or null.

Reimplemented in tinyxml2::XMLUnknown.

5.27.2.21 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
```

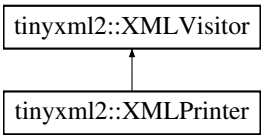
The documentation for this class was generated from the following files:

- src/tinyxml2.h
- src/tinyxml2.cpp

5.28 tinyxml2::XMLPrinter Class Reference

`#include <tinyxml2.h>`

Inheritance diagram for tinyxml2::XMLPrinter:



Public Member Functions

- XMLPrinter (FILE *file=0, bool compact=false, int depth=0)
- 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 &)

  Visit a document.
- virtual bool VisitExit (const XMLDocument &)

  Visit a document.

- virtual bool VisitEnter (const XMLElement &element, const XMLAttribute *attribute)

  Visit an element.
- virtual bool VisitExit (const XMLElement &element)

  Visit an element.
- virtual bool Visit (const XMLText &text)

  Visit a text node.
- virtual bool Visit (const XMLComment &comment)

  Visit a comment node.
- virtual bool Visit (const XMLDeclaration &declaration)

  Visit a declaration.
- virtual bool Visit (const XMLUnknown &unknown)

  Visit an unknown node.
- const char * CStr () const
- int CStrSize () const
- void ClearBuffer (bool resetToFirstElement=true)

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**

5.28.1 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();
```

5.28.2 Constructor & Destructor Documentation

5.28.2.1 XMLPrinter()

```
tinyxml2::XMLPrinter::XMLPrinter (
             FILE * file = 0,
             bool compact = false,
             int depth = 0 )
```

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.

5.28.3 Member Function Documentation

5.28.3.1 ClearBuffer()

```
void tinyxml2::XMLPrinter::ClearBuffer (
             bool resetToFirstElement = true )  [inline]
```

If in print to memory mode, reset the buffer to the beginning.

5.28.3.2 CStr()

```
const char * tinyxml2::XMLPrinter::CStr ( ) const  [inline]
```

If in print to memory mode, return a pointer to the XML file in memory.

5.28.3.3 CStrSize()

```
int 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.)

5.28.3.4 OpenElement()

```
void tinyxml2::XMLPrinter::OpenElement (
            const char * name,
            bool compactMode = false )
```

If streaming, start writing an element. The element must be closed with CloseElement()

5.28.3.5 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().

5.28.3.6 PushHeader()

```
void tinyxml2::XMLPrinter::PushHeader (
            bool writeBOM,
            bool writeDeclaration )
```

If streaming, write the BOM and declaration.

5.28.3.7 Visit() [1/4]

```
bool tinyxml2::XMLPrinter::Visit (
            const XMLComment &  )  [virtual]
```

Visit a comment node.

Reimplemented from tinyxml2::XMLVisitor.

5.28.3.8 Visit() [2/4]

```
bool tinyxml2::XMLPrinter::Visit (
            const XMLDeclaration &  )  [virtual]
```

Visit a declaration.

Reimplemented from tinyxml2::XMLVisitor.

5.28.3.9 Visit() [3/4]

```
bool tinyxml2::XMLPrinter::Visit (
            const XMLText &  )  [virtual]
```

Visit a text node.

Reimplemented from tinyxml2::XMLVisitor.

5.28.3.10 Visit() [4/4]

```
bool tinyxml2::XMLPrinter::Visit (
            const XMLUnknown &  )  [virtual]
```

Visit an unknown node.

Reimplemented from tinyxml2::XMLVisitor.

5.28.3.11 VisitEnter() [1/2]

```
bool tinyxml2::XMLPrinter::VisitEnter (
            const XMLDocument &  )  [virtual]
```

Visit a document.

Reimplemented from tinyxml2::XMLVisitor.

5.28.3.12 VisitEnter() [2/2]

```
bool tinyxml2::XMLPrinter::VisitEnter (
            const XMLElement & ,
            const XMLAttribute *  )  [virtual]
```

Visit an element.

Reimplemented from tinyxml2::XMLVisitor.

5.28.3.13 VisitExit() [1/2]

```
virtual bool tinyxml2::XMLPrinter::VisitExit (
            const XMLDocument &  )  [inline], [virtual]
```

Visit a document.

Reimplemented from tinyxml2::XMLVisitor.

5.28.3.14 VisitExit() [2/2]

```
bool tinyxml2::XMLPrinter::VisitExit (
            const XMLElement &  )  [virtual]
```

Visit an element.

Reimplemented from tinyxml2::XMLVisitor.

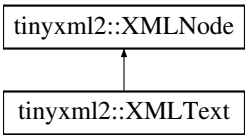
The documentation for this class was generated from the following files:

- src/tinyxml2.h
- src/tinyxml2.cpp

5.29 tinyxml2::XMLText Class Reference

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

Inheritance diagram for tinyxml2::XMLText:



Public Member Functions

- virtual bool Accept (XMLVisitor *visitor) const
- virtual XMLText * ToText ()

  Safely cast to Text, or null.
- virtual const XMLText * ToText () const
- 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
- virtual bool ShallowEqual (const XMLNode *compare) const

Protected Member Functions

- **XMLText** (XMLDocument *doc)
- char * ParseDeep (char *p, StrPair *parentEndTag, int *curLineNumPtr)

Friends

- class **XMLDocument**

Additional Inherited Members

5.29.1 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().

5.29.2 Member Function Documentation

5.29.2.1 Accept()

```
bool tinyxml2::XMLText::Accept (
            XMLVisitor * visitor ) const  [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.

5.29.2.2 ParseDeep()

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

Reimplemented from tinyxml2::XMLNode.

5.29.2.3 ShallowClone()

```
XMLNode * tinyxml2::XMLText::ShallowClone (
            XMLDocument * document ) const  [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.

5.29.2.4 ShallowEqual()

```
bool tinyxml2::XMLText::ShallowEqual (
            const XMLNode * compare ) const  [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.

5.29.2.5 ToText() [1/2]

```
virtual XMLText * tinyxml2::XMLText::ToText ( )  [inline], [virtual]
```

Safely cast to Text, or null.

Reimplemented from tinyxml2::XMLNode.

5.29.2.6 ToText() [2/2]

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

Reimplemented from tinyxml2::XMLNode.

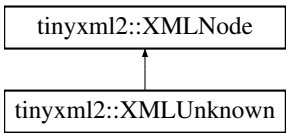
The documentation for this class was generated from the following files:

- src/tinyxml2.h
- src/tinyxml2.cpp

5.30 tinyxml2::XMLUnknown Class Reference

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

Inheritance diagram for tinyxml2::XMLUnknown:



Public Member Functions

- virtual XMLUnknown * ToUnknown ()

  Safely cast to an Unknown, or null.
- virtual const XMLUnknown * ToUnknown () const
- virtual bool Accept (XMLVisitor *visitor) const
- virtual XMLNode * ShallowClone (XMLDocument *document) const
- virtual bool ShallowEqual (const XMLNode *compare) const

Protected Member Functions

- **XMLUnknown** (XMLDocument *doc)
- char * ParseDeep (char *p, StrPair *parentEndTag, int *curLineNumPtr)

Friends

- class **XMLDocument**

Additional Inherited Members

5.30.1 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.

5.30.2 Member Function Documentation

5.30.2.1 Accept()

```
bool tinyxml2::XMLUnknown::Accept (
            XMLVisitor * visitor ) const  [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.

5.30.2.2 ParseDeep()

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

Reimplemented from tinyxml2::XMLNode.

5.30.2.3 ShallowClone()

```
XMLNode * tinyxml2::XMLUnknown::ShallowClone (
            XMLDocument * document ) const  [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.

5.30.2.4 ShallowEqual()

```
bool tinyxml2::XMLUnknown::ShallowEqual (
            const XMLNode * compare ) const  [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.

5.30.2.5 ToUnknown() [1/2]

```
virtual XMLUnknown * tinyxml2::XMLUnknown::ToUnknown ( )  [inline], [virtual]
```

Safely cast to an Unknown, or null.

Reimplemented from tinyxml2::XMLNode.

5.30.2.6 ToUnknown() [2/2]

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

Reimplemented from tinyxml2::XMLNode.

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

- src/tinyxml2.h
- src/tinyxml2.cpp

5.31 tinyxml2::XMLUtil Class Reference

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)

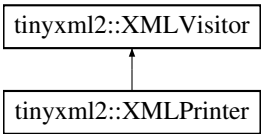
The documentation for this class was generated from the following files:

- src/tinyxml2.h
- src/tinyxml2.cpp

5.32 tinyxml2::XMLVisitor Class Reference

`#include <tinyxml2.h>`

Inheritance diagram for tinyxml2::XMLVisitor:



Public Member Functions

- 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.

5.32.1 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()

5.32.2 Member Function Documentation

5.32.2.1 Visit() [1/4]

```
virtual bool tinyxml2::XMLVisitor::Visit (
            const XMLComment &  )  [inline], [virtual]
```

Visit a comment node.

Reimplemented in tinyxml2::XMLPrinter.

5.32.2.2 Visit() [2/4]

```
virtual bool tinyxml2::XMLVisitor::Visit (
            const XMLDeclaration &  )  [inline], [virtual]
```

Visit a declaration.

Reimplemented in tinyxml2::XMLPrinter.

5.32.2.3 Visit() [3/4]

```
virtual bool tinyxml2::XMLVisitor::Visit (
            const XMLText &  )  [inline], [virtual]
```

Visit a text node.

Reimplemented in tinyxml2::XMLPrinter.

5.32.2.4 Visit() [4/4]

```
virtual bool tinyxml2::XMLVisitor::Visit (
            const XMLUnknown &  )  [inline], [virtual]
```

Visit an unknown node.

Reimplemented in tinyxml2::XMLPrinter.

5.32.2.5 VisitEnter() [1/2]

```
virtual bool tinyxml2::XMLVisitor::VisitEnter (
            const XMLDocument &  )  [inline], [virtual]
```

Visit a document.

Reimplemented in tinyxml2::XMLPrinter.

5.32.2.6 VisitEnter() [2/2]

```
virtual bool tinyxml2::XMLVisitor::VisitEnter (
            const XMLElement & ,
            const XMLAttribute *  )  [inline], [virtual]
```

Visit an element.

Reimplemented in tinyxml2::XMLPrinter.

5.32.2.7 VisitExit() [1/2]

```
virtual bool tinyxml2::XMLVisitor::VisitExit (
            const XMLDocument &  )  [inline], [virtual]
```

Visit a document.

Reimplemented in tinyxml2::XMLPrinter.

5.32.2.8 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:

- src/tinyxml2.h

Chapter 6

File Documentation

6.1 src/engine.cpp File Reference

Implementation of the game logic.

```
#include "engine.hpp"
```

6.1.1 Detailed Description

Implementation of the game logic.

Author

Peter Bence (ecneb2000@gmail.com)

Version

0.1

Date

2022-10-29

Copyright

6.2 src/engine.hpp File Reference

Definition and part implementation of game logic.

```
#include <vector>
#include <string>
#include <map>
#include <iostream>
#include <memory>
#include "tinyxml2.h"
```

Classes

- class Object

  Base class for any object that can be owned by an Entity or Room.
- class Key

  An object that can open a room.
- class Entity

  Base class for a NPC, USER or any other Entity living in the game world.
- class Player
- class NPC
- class Room

  Part of the World. A room that contains items, that can be collected, players or npc can move in and out of these rooms.
- class Mission

  class Mission helps to create a mission system, that will give objectives, to accomplish, for the player. This will give the player a direction, how to finish the story.
- class World

  The world that contains and manages all the rooms, entities. A World object will able to parse the story file and initialize the game and run it.

Typedefs

- typedef std::unique_ptr< Room > **node**
- typedef std::unique_ptr< Object > **item**
- typedef std::unique_ptr< NPC > **npc**
- typedef std::vector< node > **nodes**
- typedef std::vector< int > **neighbours**
- typedef std::vector< item > **items**
- typedef std::vector< npc > **npcs**
- typedef std::pair< int, std::vector< int > > **roomConnection**
- typedef std::vector< Mission > **missions**

Enumerations

- enum **missionStatus** { **finished** , **active** , **unactive** }
- enum LockStatus { **locked** , **unlocked** }

  Enum class to make tracking room lockstatus easier.

6.2.1 Detailed Description

Definition and part implementation of game logic.

Author

Peter Bence (ecneb2000@gmail.com)

Version

0.1

Date

2022-10-29

Copyright

Copyright (c) 2022

6.2.2 Enumeration Type Documentation

6.2.2.1 LockStatus

```
enum LockStatus
```

Enum class to make tracking room lockstatus easier.

6.3 engine.hpp

Go to the documentation of this file.

```
1
11 #ifndef ENGINE
12 #define ENGINE
13 #include <vector>
14 #include <string>
15 #include <map>
16 #include <iostream>
17 #include <memory>
18 #include "tinyxml2.h"
19
20 class World;
21 class Object;
22 class Room;
23 class Entity;
24 class Player;
25 class NPC;
26 class Mission;
27 class Choice;
28
33 typedef std::unique_ptr<Room> node;
38 typedef std::unique_ptr<Object> item;
43 typedef std::unique_ptr<NPC> npc;
48 typedef std::vector<node> nodes;
53 typedef std::vector<int> neighbours;
58 typedef std::vector<item> items;
63 typedef std::vector<npc> npcs;
68 typedef std::pair<int, std::vector<int» roomConnection;
73 typedef std::vector<Mission> missions;
74
75 enum missionStatus{finished, active, unactive};
76
81 class Object {
82     const std::string objectName;
83     const std::string objectDescriptor;
84     const int objID;
85
86 public:
94     Object(const std::string& n, const int id, const std::string& d) :  objectName(n),
      objectDescriptor(d), objID(id)  {}
100     std::string getName()const {return objectName;}
106     int const getID() {return objID;}
112     std::string getDescription()const {return objectDescriptor;}
113     virtual ~Object() {}
114 };
115
120 class Key :  public Object {
121     const int keyID;
122 public:
131     Key(const int kid, const std::string& n, const int id, const std::string& d) :  Object(n, id, d),
      keyID(kid) {}
137     int getKeyID() {
138         return keyID;
139     }
140 };
141
146 class Entity {
147 protected:
148     const std::string name;
149     items inventory;
150     int hp;
151     int stamina;
```

```
152     int intelligence;
153     int strenght;
154 public:
160     Entity(const std::string& n) :  name(n) {}
166     std::string getName()const {return name;}
173     Entity& addItem(item &i) {
174         inventory.push_back(std::move(i));
175         return *this;
176     }
183     Entity& addItems(items &i) {
184         for (auto it = i.begin(); it != i.end(); it++) {
185             this->addItem(*it);
186         }
187         return *this;
188     }
194     items& getInventory() {
195         return inventory;
196     }
197
198     virtual ~Entity() {
199         for (items::iterator it = inventory.begin(); it != inventory.end(); it++) {
200             it->reset();
201         }
202     }
203 };
204
205 class Player :  public Entity {
206     Room* location;
207 public:
212     Player() :  Entity("Default") {}
219     Player(const std::string& n, Room* init_location) :  Entity(n), location(init_location) {}
225     Room* getLocation() {
226         return location;
227     }
234     Entity& setLocation(Room* room_ptr) {
235         this->location = room_ptr;
236         return *this;
237     }
238 };
239
240 class NPC :  public Entity {
241     std::string dialog;
242     missions missions_to_give;
243 public:
251     NPC(const std::string& n,  const std::string& d, missions m) :  Entity(n), dialog(d),
    missions_to_give(m) {}
252     missions& getMissions() {
253         return missions_to_give;
254     }
255     std::string getDialog() {return dialog;}
256 };
257
262 enum LockStatus{locked, unlocked};
263
269 class Room {
270     const std::string roomName; // Name of the room.
271     const int roomID; // ID of the room, that connects a key to this room.
272     const std::string description; // Description of the room.
273     LockStatus lock;
274     npcs roomPopulation;
275     neighbours connectedRooms; // Neighbouring rooms.
276     items inventory; // Items, that can be found in the room.
277 public:
285     Room(const std::string& n, int id, const std::string& desc) :  roomName(n), roomID(id),
    description(desc), lock(locked) {}
294     Room(const std::string& n, int id, const std::string& desc, neighbours cn) :  roomName(n),
    roomID(id), description(desc), lock(locked), connectedRooms(cn) {}
300     std::string getName()const {return roomName;}
306     int getID()const {return roomID;}
312     std::string getDescription()const {return description;}
318     neighbours getNeighbours()const {return connectedRooms;}
325     Room& addNeighbour(int rid) {
326         connectedRooms.push_back(rid);
327         return *this;
328     }
335     Room& addNeighbours(neighbours rids) {
336         for (auto rid :  rids) {
337             this->addNeighbour(rid);
338         }
339         return *this;
340     }
346     items& getItems() {return inventory;}
353     Room& addItem(item& i) {
354         inventory.emplace_back(std::move(i));
355         return *this;
356     }
363     Room& addItems(items& inv) {
```

```
364         for (items::iterator it = inv.begin(); it != inv.end(); it++) {
365             this->addItem(*it);
366         }
367         return *this;
368     }
375     Room& addEntity(npc& e) {
376         roomPopulation.emplace_back(std::move(e));
377         return *this;
378     }
385     Room& addEntities(npcs& ents) {
386         for (npcs::iterator it = ents.begin(); it != ents.end(); it++) {
387             this->addEntity(*it);
388         }
389         return *this;
390     }
397     Room& setLock(LockStatus stat) {
398         lock = stat;
399         return *this;
400     }
409     bool static unlock(item&, node&);
416     bool isLocked() {
417         if ( lock == locked ) {
418             return true;
419         }
420         return false;
421     }
427     const npcs& getPopulation() {return roomPopulation;}
433     ~Room() {
434         for (items::iterator iit = inventory.begin(); iit != inventory.end(); iit++) {
435             iit->reset();
436         }
437         for (npcs::iterator eit = roomPopulation.begin(); eit != roomPopulation.end(); eit++) {
438             eit->reset();
439         }
440     }
441 };
442
448 class Mission {
449     std::string description;
450     int targetRoom;
451     int targetItem;
452     missionStatus status;
457     void complete() {
458         status = finished;
459     }
460 public:
466     Mission(const std::string& desc) :  description(desc) {}
472     int getTargetRoom() {return targetRoom;}
479     Mission& setTargetRoom(int targetRoomID) {
480         targetRoom = targetRoomID;
481         return *this;
482     }
488     int getTargetItem() {return targetItem;}
495     Mission& setTargetItem(int targetItemID) {
496         targetItem = targetItemID;
497         return *this;
498     }
506     bool checkStatus(Player&);
512     Mission& startMission() {
513         status = active;
514         return *this;
515     }
516 };
517
523 class World {
524     std::string title;
525     nodes worldRooms;
526     tinyxml2::XMLDocument story;
527     Player player;
528     missions active_missions;
529 public:
530     World() {}
536     World(const char* path2story) {
537         story.LoadFile(path2story);
538     }
544     nodes& getWorldRooms() {
545         return worldRooms;
546     }
552     missions& getWorldMission() {
553         return active_missions;
554     }
562     void RoomFactory(const std::string&, int, const std::string&, neighbours);
568     tinyxml2::XMLDocument& getStory() {return story;}
575     items makeInventory(tinyxml2::XMLElement*);
582     neighbours parseConnections(tinyxml2::XMLElement*);
588     void loadRooms(tinyxml2::XMLElement*);
594     void loadEntities(tinyxml2::XMLElement*, node&);
```

```
601     Mission makeMission(tinyxml2::XMLElement*);
607     void loadWorldMissions(tinyxml2::XMLElement*);
614     missions loadNPCMissions(tinyxml2::XMLElement*);
620     void initWorld(const char*);
627     void enterRoom(node& r) {
628         player.setLocation(r.get());
629     }
635     Player& getPlayer() {
636         return player;
637     }
643     void startMission(Mission& m) {
644         active_missions.push_back(m.startMission());
645     }
650     void destroyWorld() {
651         for (nodes::iterator it = worldRooms.begin(); it != worldRooms.end(); it++) {
652             it->reset();
653         }
654     }
655     ~World() {
656         this->destroyWorld();
657     }
658 };
659 #endif
```

6.4 src/interface.cpp File Reference

Implementation of the interface that connects the game and the game eninge.

```
#include "engine.hpp"
#include "interface.hpp"
```

6.4.1 Detailed Description

Implementation of the interface that connects the game and the game eninge.

Author

Peter Bence (ecneb2000@gmail.com)

Version

0.1

Date

2022-10-29

Copyright

Copyright (c) 2022

6.5 src/interface.hpp File Reference

Implementation of interface that connects the game with game eingine.

Classes

- class Action

  Abstract class representing a player action.
- class Move

  Realisation of the action class, representing movement from a room to another.
- class Search

  Realisation of the action class, respresenting the search of a room.
- class PickUp

  Realisation of the action class, the player picking up an item.
- class Interact

  Realisation of the action class, interaction with an NPC.
- class AcceptMission

Typedefs

- typedef std::pair< items &, const npcs & > **search_results**

6.5.1 Detailed Description

Implementation of interface that connects the game with game eingine.

Author

Peter Bence (ecneb2000@gmail.com)

Version

0.1

Date

2022-10-29

Copyright

6.6 interface.hpp

Go to the documentation of this file.

```
1
11 #ifndef INTERFACE
12 #define INTERFACE
13
14 typedef std::pair<items&, const npcs&> search_results;
15
20 class Action {
21     std::string description;
22 protected:
23     World& game_world;
24 public:
32     Action(const std::string&, World&);
37     virtual void doAction();
43     std::string getDescription();
48     virtual void operator ()() = 0;
49 };
50
55 class Move :  public Action {
56 public:
64     Move(const std::string&, World&);
71     void doAction(node&) ;
78     void operator ()(node&) ;
79 };
80
85 class Search :  public Action {
86 public:
94     Search(const std::string&, World&);
102     search_results doAction(Room*) ;
110     search_results operator ()(Room*) ;
111 };
112
117 class PickUp :  public Action {
118 public:
126     PickUp(const std::string&, World&);
133     void doAction(item&) ;
140     void operator ()(item&) ;
141 };
142
147 class Interact :  public Action {
148 public:
156     Interact(const std::string&, World&);
163     missions doAction(npc&) ;
170     void acceptMission(npc&);
177     void operator ()(npc&) ;
178 };
179
180 class AcceptMission :  public Action {
181 public:
189     AcceptMission(const std::string&, World&);
196     void doAction(Mission&) ;
203     void operator ()(Mission&) ;
204 };
205
206 #endif
```

6.7 src/main.cpp File Reference

Main loop implementation of "SpaceWalk TheGame".

```
#include "interface.hpp"
#include "Config.h"
```

Functions

- int **main** ()

6.7.1 Detailed Description

Main loop implementation of "SpaceWalk TheGame".

Author

Peter Bence (ecneb2000@gmail.com)

Version

0.1

Date

2022-10-29

Copyright

6.8 tinyxml2.h

```
1 /*
2 Original code by Lee Thomason (www.grinninglizard.com)
3
4 This software is provided 'as-is', without any express or implied
5 warranty.  In no event will the authors be held liable for any
6 damages arising from the use of this software.
7
8 Permission is granted to anyone to use this software for any
9 purpose, including commercial applications, and to alter it and
10 redistribute it freely, subject to the following restrictions:
11
12 1.  The origin of this software must not be misrepresented; you must
13 not claim that you wrote the original software.  If you use this
14 software in a product, an acknowledgment in the product documentation
15 would be appreciated but is not required.
16
17 2.  Altered source versions must be plainly marked as such, and
18 must not be misrepresented as being the original software.
19
20 3.  This notice may not be removed or altered from any source
21 distribution.
22 */
23
24 #ifndef TINYXML2_INCLUDED
25 #define TINYXML2_INCLUDED
26
27 #if defined(ANDROID_NDK) || defined(__BORLANDC__) || defined(__QNXNTO__)
28 #   include <ctype.h>
29 #   include <limits.h>
30 #   include <stdio.h>
31 #   include <stdlib.h>
32 #   include <string.h>
33 #   if defined(__PS3__)
34 #       include <stddef.h>
35 #   endif
36 #else
37 #   include <cctype>
38 #   include <climits>
39 #   include <cstdio>
40 #   include <cstdlib>
41 #   include <cstring>
42 #endif
43 #include <stdint.h>
44
45 /*
46 TODO: intern strings instead of allocation.
47 */
48 /*
```

```
49 gcc:
50 g++ -Wall -DTINYXML2_DEBUG tinyxml2.cpp xmltest.cpp -o gccxmltest.exe
51
52 Formatting, Artistic Style:
53 AStyle.exe --style=1tbs --indent-switches --break-closing-brackets --indent-preprocessor tinyxml2.cpp
      tinyxml2.h
54 */
55
56 #if defined( _DEBUG ) || defined (__DEBUG__)
57 #   ifndef TINYXML2_DEBUG
58 #       define TINYXML2_DEBUG
59 #   endif
60 #endif
61
62 #ifdef _MSC_VER
63 #   pragma warning(push)
64 #   pragma warning(disable:  4251)
65 #endif
66
67 #ifdef _WIN32
68 #   ifdef TINYXML2_EXPORT
69 #       define TINYXML2_LIB __declspec(dllexport)
70 #   elif defined(TINYXML2_IMPORT)
71 #       define TINYXML2_LIB __declspec(dllimport)
72 #   else
73 #       define TINYXML2_LIB
74 #   endif
75 #elif __GNUC__ >= 4
76 #   define TINYXML2_LIB __attribute__((visibility("default")))
77 #else
78 #   define TINYXML2_LIB
79 #endif
80
81
82 #if !defined(TIXMLASSERT)
83 #if defined(TINYXML2_DEBUG)
84 #   if defined(_MSC_VER)
85 #       // "(void)0," is for suppressing C4127 warning in "assert(false)", "assert(true)" and the like
86 #       define TIXMLASSERT( x )           do { if ( !((void)0,(x))) { __debugbreak(); } } while(false)
87 #   elif defined (ANDROID_NDK)
88 #       include <android/log.h>
89 #       define TIXMLASSERT( x )           do { if ( !(x)) { __android_log_assert( "assert", "grinliz",
      "ASSERT in '%s' at %d.", __FILE__, __LINE__ ); } } while(false)
90 #   else
91 #       include <assert.h>
92 #       define TIXMLASSERT                assert
93 #   endif
94 #else
95 #   define TIXMLASSERT( x )               do {} while(false)
96 #endif
97 #endif
98
99 /* Versioning, past 1.0.14:
100 http://semver.org/
101 */
102 static const int TIXML2_MAJOR_VERSION = 9;
103 static const int TIXML2_MINOR_VERSION = 0;
104 static const int TIXML2_PATCH_VERSION = 0;
105
106 #define TINYXML2_MAJOR_VERSION 9
107 #define TINYXML2_MINOR_VERSION 0
108 #define TINYXML2_PATCH_VERSION 0
109
110 // A fixed element depth limit is problematic.  There needs to be a
111 // limit to avoid a stack overflow.  However, that limit varies per
112 // system, and the capacity of the stack.  On the other hand, it's a trivial
113 // attack that can result from ill, malicious, or even correctly formed XML,
114 // so there needs to be a limit in place.
115 static const int TINYXML2_MAX_ELEMENT_DEPTH = 100;
116
117 namespace tinyxml2
118 {
119 class XMLDocument;
120 class XMLElement;
121 class XMLAttribute;
122 class XMLComment;
123 class XMLText;
124 class XMLDeclaration;
125 class XMLUnknown;
126 class XMLPrinter;
127
128 /*
129 A class that wraps strings.  Normally stores the start and end
130 pointers into the XML file itself, and will apply normalization
131 and entity translation if actually read.  Can also store (and memory
132 manage) a traditional char[]
133
```

```
134 Isn't clear why TINYXML2_LIB is needed; but seems to fix #719
135 */
136 class TINYXML2_LIB StrPair
137 {
138 public:
139     enum Mode {
140         NEEDS_ENTITY_PROCESSING         = 0x01,
141         NEEDS_NEWLINE_NORMALIZATION     = 0x02,
142         NEEDS_WHITESPACE_COLLAPSING     = 0x04,
143 
144         TEXT_ELEMENT                    = NEEDS_ENTITY_PROCESSING | NEEDS_NEWLINE_NORMALIZATION,
145         TEXT_ELEMENT_LEAVE_ENTITIES     = NEEDS_NEWLINE_NORMALIZATION,
146         ATTRIBUTE_NAME                  = 0,
147         ATTRIBUTE_VALUE                 = NEEDS_ENTITY_PROCESSING | NEEDS_NEWLINE_NORMALIZATION,
148         ATTRIBUTE_VALUE_LEAVE_ENTITIES  = NEEDS_NEWLINE_NORMALIZATION,
149         COMMENT                         = NEEDS_NEWLINE_NORMALIZATION
150     };
151 
152     StrPair() :  _flags( 0 ), _start( 0 ), _end( 0 ) {}
153     ~StrPair();
154 
155     void Set( char* start, char* end, int flags ) {
156         TIXMLASSERT( start );
157         TIXMLASSERT( end );
158         Reset();
159         _start  = start;
160         _end    = end;
161         _flags  = flags | NEEDS_FLUSH;
162     }
163 
164     const char* GetStr();
165 
166     bool Empty()const {
167         return _start == _end;
168     }
169 
170     void SetInternedStr( const char* str ) {
171         Reset();
172         _start = const_cast<char*>(str);
173     }
174 
175     void SetStr( const char* str, int flags=0 );
176 
177     char* ParseText( char* in, const char* endTag, int strFlags, int* curLineNumPtr );
178     char* ParseName( char* in );
179 
180     void TransferTo( StrPair* other );
181     void Reset();
182 
183 private:
184     void CollapseWhitespace();
185 
186     enum {
187         NEEDS_FLUSH = 0x100,
188         NEEDS_DELETE = 0x200
189     };
190 
191     int     _flags;
192     char*   _start;
193     char*   _end;
194 
195     StrPair( const StrPair& other );  // not supported
196     void operator=( const StrPair& other );  // not supported, use TransferTo()
197 };
198 
199 
200 /*
201 A dynamic array of Plain Old Data.  Doesn't support constructors, etc.
202 Has a small initial memory pool, so that low or no usage will not
203 cause a call to new/delete
204 */
205 template <class T, int INITIAL_SIZE>
206 class DynArray
207 {
208 public:
209     DynArray() :
210         _mem( _pool ),
211         _allocated( INITIAL_SIZE ),
212         _size( 0 )
213     {
214     }
215 
216     ~DynArray() {
217         if ( _mem != _pool ) {
218             delete [] _mem;
219         }
220     }
```

```
221
222     void Clear() {
223         _size = 0;
224     }
225
226     void Push( T t ) {
227         TIXMLASSERT( _size < INT_MAX );
228         EnsureCapacity( _size+1 );
229         _mem[_size] = t;
230         ++_size;
231     }
232
233     T* PushArr( int count ) {
234         TIXMLASSERT( count >= 0 );
235         TIXMLASSERT( _size <= INT_MAX - count );
236         EnsureCapacity( _size+count );
237         T* ret = &_mem[_size];
238         _size += count;
239         return ret;
240     }
241
242     T Pop() {
243         TIXMLASSERT( _size > 0 );
244         --_size;
245         return _mem[_size];
246     }
247
248     void PopArr( int count ) {
249         TIXMLASSERT( _size >= count );
250         _size -= count;
251     }
252
253     bool Empty()const                   {
254         return _size == 0;
255     }
256
257     T& operator[](int i)                 {
258         TIXMLASSERT( i>= 0 && i < _size );
259         return _mem[i];
260     }
261
262     const T& operator[](int i)const     {
263         TIXMLASSERT( i>= 0 && i < _size );
264         return _mem[i];
265     }
266
267     const T& PeekTop()const             {
268         TIXMLASSERT( _size > 0 );
269         return _mem[ _size - 1];
270     }
271
272     int Size()const                     {
273         TIXMLASSERT( _size >= 0 );
274         return _size;
275     }
276
277     int Capacity()const                 {
278         TIXMLASSERT( _allocated >= INITIAL_SIZE );
279         return _allocated;
280     }
281
282     void SwapRemove(int i) {
283         TIXMLASSERT(i >= 0 && i < _size);
284         TIXMLASSERT(_size > 0);
285         _mem[i] = _mem[_size - 1];
286         --_size;
287     }
288
289     const T* Mem()const                 {
290         TIXMLASSERT( _mem );
291         return _mem;
292     }
293
294     T* Mem() {
295         TIXMLASSERT( _mem );
296         return _mem;
297     }
298
299 private:
300     DynArray( const DynArray& ); // not supported
301     void operator=( const DynArray& ); // not supported
302
303     void EnsureCapacity( int cap ) {
304         TIXMLASSERT( cap > 0 );
305         if ( cap > _allocated ) {
306             TIXMLASSERT( cap <= INT_MAX / 2 );
307             const int newAllocated = cap * 2;
```

```
308             T* newMem = new T[newAllocated];
309             TIXMLASSERT( newAllocated >= _size );
310             memcpy( newMem, _mem, sizeof(T)*_size );    // warning:  not using constructors, only works
    for PODs
311             if ( _mem != _pool ) {
312                 delete [] _mem;
313             }
314             _mem = newMem;
315             _allocated = newAllocated;
316         }
317     }
318
319     T*  _mem;
320     T   _pool[INITIAL_SIZE];
321     int _allocated;     // objects allocated
322     int _size;          // number objects in use
323 };
324
325
326 /*
327 Parent virtual class of a pool for fast allocation
328 and deallocation of objects.
329 */
330 class MemPool
331 {
332 public:
333     MemPool() {}
334     virtual ~MemPool() {}
335
336     virtual int ItemSize() const = 0;
337     virtual void* Alloc() = 0;
338     virtual void Free( void* ) = 0;
339     virtual void SetTracked() = 0;
340 };
341
342
343 /*
344 Template child class to create pools of the correct type.
345 */
346 template< int ITEM_SIZE >
347 class MemPoolT :  public MemPool
348 {
349 public:
350     MemPoolT() :  _blockPtrs(), _root(0), _currentAllocs(0), _nAllocs(0), _maxAllocs(0), _nUntracked(0)
    {}
351     ~MemPoolT() {
352         MemPoolT< ITEM_SIZE >::Clear();
353     }
354
355     void Clear() {
356         // Delete the blocks.
357         while( !_blockPtrs.Empty()) {
358             Block* lastBlock = _blockPtrs.Pop();
359             delete lastBlock;
360         }
361         _root = 0;
362         _currentAllocs = 0;
363         _nAllocs = 0;
364         _maxAllocs = 0;
365         _nUntracked = 0;
366     }
367
368     virtual int ItemSize()const    {
369         return ITEM_SIZE;
370     }
371     int CurrentAllocs()const       {
372         return _currentAllocs;
373     }
374
375     virtual void* Alloc() {
376         if ( !_root ) {
377             // Need a new block.
378             Block* block = new Block();
379             _blockPtrs.Push( block );
380
381             Item* blockItems = block->items;
382             for( int i = 0; i < ITEMS_PER_BLOCK - 1; ++i ) {
383                 blockItems[i].next = &(blockItems[i + 1]);
384             }
385             blockItems[ITEMS_PER_BLOCK - 1].next = 0;
386             _root = blockItems;
387         }
388         Item* const result = _root;
389         TIXMLASSERT( result != 0 );
390         _root = _root->next;
391
392         ++_currentAllocs;
```

```
393         if ( _currentAllocs > _maxAllocs ) {
394             _maxAllocs = _currentAllocs;
395         }
396         ++_nAllocs;
397         ++_nUntracked;
398         return result;
399     }
400
401     virtual void Free( void* mem ) {
402         if ( !mem ) {
403             return;
404         }
405         --_currentAllocs;
406         Item* item = static_cast<Item*>( mem );
407 #ifdef TINYXML2_DEBUG
408         memset( item, 0xfe, sizeof( *item ) );
409 #endif
410         item->next = _root;
411         _root = item;
412     }
413     void Trace( const char* name ) {
414         printf( "Mempool %s watermark=%d [%dk] current=%d size=%d nAlloc=%d blocks=%d\n",
415                 name, _maxAllocs, _maxAllocs * ITEM_SIZE / 1024, _currentAllocs,
416                 ITEM_SIZE, _nAllocs, _blockPtrs.Size() );
417     }
418
419     void SetTracked() {
420         --_nUntracked;
421     }
422
423     int Untracked()const {
424         return _nUntracked;
425     }
426
427     // This number is perf sensitive.  4k seems like a good tradeoff on my machine.
428     // The test file is large, 170k.
429     // Release:          VS2010 gcc(no opt)
430     //      1k:          4000
431     //      2k:          4000
432     //      4k:          3900    21000
433     //      16k:        5200
434     //      32k:        4300
435     //      64k:        4000    21000
436     // Declared public because some compilers do not accept to use ITEMS_PER_BLOCK
437     // in private part if ITEMS_PER_BLOCK is private
438     enum { ITEMS_PER_BLOCK = (4 * 1024) / ITEM_SIZE };
439
440 private:
441     MemPoolT( const MemPoolT& ); // not supported
442     void operator=( const MemPoolT& ); // not supported
443
444     union Item {
445         Item*   next;
446         char    itemData[ITEM_SIZE];
447     };
448     struct Block {
449         Item items[ITEMS_PER_BLOCK];
450     };
451     DynArray< Block*, 10 > _blockPtrs;
452     Item* _root;
453
454     int _currentAllocs;
455     int _nAllocs;
456     int _maxAllocs;
457     int _nUntracked;
458 };
459
460
461
481 class TINYXML2_LIB XMLVisitor
482 {
483 public:
484     virtual ~XMLVisitor() {}
485
487     virtual bool VisitEnter( const XMLDocument& /*doc*/ )          {
488         return true;
489     }
491     virtual bool VisitExit( const XMLDocument& /*doc*/ )            {
492         return true;
493     }
494
496     virtual bool VisitEnter( const XMLElement& /*element*/, const XMLAttribute* /*firstAttribute*/ )
      {
497         return true;
498     }
500     virtual bool VisitExit( const XMLElement& /*element*/ )          {
501         return true;
```

```
502     }
503
505     virtual bool Visit( const XMLDeclaration& /*declaration*/ )       {
506         return true;
507     }
509     virtual bool Visit( const XMLText& /*text*/ )                    {
510         return true;
511     }
513     virtual bool Visit( const XMLComment& /*comment*/ )               {
514         return true;
515     }
517     virtual bool Visit( const XMLUnknown& /*unknown*/ )               {
518         return true;
519     }
520 };
521
522 // WARNING: must match XMLDocument::_errorNames[]
523 enum XMLError {
524     XML_SUCCESS = 0,
525     XML_NO_ATTRIBUTE,
526     XML_WRONG_ATTRIBUTE_TYPE,
527     XML_ERROR_FILE_NOT_FOUND,
528     XML_ERROR_FILE_COULD_NOT_BE_OPENED,
529     XML_ERROR_FILE_READ_ERROR,
530     XML_ERROR_PARSING_ELEMENT,
531     XML_ERROR_PARSING_ATTRIBUTE,
532     XML_ERROR_PARSING_TEXT,
533     XML_ERROR_PARSING_CDATA,
534     XML_ERROR_PARSING_COMMENT,
535     XML_ERROR_PARSING_DECLARATION,
536     XML_ERROR_PARSING_UNKNOWN,
537     XML_ERROR_EMPTY_DOCUMENT,
538     XML_ERROR_MISMATCHED_ELEMENT,
539     XML_ERROR_PARSING,
540     XML_CAN_NOT_CONVERT_TEXT,
541     XML_NO_TEXT_NODE,
542     XML_ELEMENT_DEPTH_EXCEEDED,
543
544     XML_ERROR_COUNT
545 };
546
547
548 /*
549 Utility functionality.
550 */
551 class TINYXML2_LIB XMLUtil
552 {
553 public:
554     static const char* SkipWhiteSpace( const char* p, int* curLineNumPtr )  {
555         TIXMLASSERT( p );
556
557         while( IsWhiteSpace(*p) ) {
558             if (curLineNumPtr && *p == '\n') {
559                 ++(*curLineNumPtr);
560             }
561             ++p;
562         }
563         TIXMLASSERT( p );
564         return p;
565     }
566     static char* SkipWhiteSpace( char* const p, int* curLineNumPtr ) {
567         return const_cast<char*>( SkipWhiteSpace( const_cast<const char*>(p), curLineNumPtr ) );
568     }
569
570     // Anything in the high order range of UTF-8 is assumed to not be whitespace.  This isn't
571     // correct, but simple, and usually works.
572     static bool IsWhiteSpace( char p )                  {
573         return !IsUTF8Continuation(p) && isspace( static_cast<unsigned char>(p) );
574     }
575
576     inline static bool IsNameStartChar( unsigned char ch ) {
577         if ( ch >= 128 ) {
578             // This is a heuristic guess in attempt to not implement Unicode-aware isalpha()
579             return true;
580         }
581         if ( isalpha( ch ) ) {
582             return true;
583         }
584         return ch == ':'  || ch == '_';
585     }
586
587     inline static bool IsNameChar( unsigned char ch ) {
588         return IsNameStartChar( ch )
589                || isdigit( ch )
590                || ch == '.'
591                || ch == '-';
592     }
```

```
593
594     inline static bool IsPrefixHex( const char* p) {
595         p = SkipWhiteSpace(p, 0);
596         return p && *p == '0' && ( *(p + 1) == 'x' || *(p + 1) == 'X');
597     }
598
599     inline static bool StringEqual( const char* p, const char* q, int nChar=INT_MAX )  {
600         if ( p == q ) {
601             return true;
602         }
603         TIXMLASSERT( p );
604         TIXMLASSERT( q );
605         TIXMLASSERT( nChar >= 0 );
606         return strncmp( p, q, nChar ) == 0;
607     }
608
609     inline static bool IsUTF8Continuation( const char p ) {
610         return ( p & 0x80 ) != 0;
611     }
612
613     static const char* ReadBOM( const char* p, bool* hasBOM );
614     // p is the starting location,
615     // the UTF-8 value of the entity will be placed in value, and length filled in.
616     static const char* GetCharacterRef( const char* p, char* value, int* length );
617     static void ConvertUTF32ToUTF8( unsigned long input, char* output, int* length );
618
619     // converts primitive types to strings
620     static void ToStr( int v, char* buffer, int bufferSize );
621     static void ToStr( unsigned v, char* buffer, int bufferSize );
622     static void ToStr( bool v, char* buffer, int bufferSize );
623     static void ToStr( float v, char* buffer, int bufferSize );
624     static void ToStr( double v, char* buffer, int bufferSize );
625     static void ToStr(int64_t v, char* buffer, int bufferSize);
626     static void ToStr(uint64_t v, char* buffer, int bufferSize);
627
628     // converts strings to primitive types
629     static bool ToInt( const char* str, int* value );
630     static bool ToUnsigned( const char* str, unsigned* value );
631     static bool ToBool( const char* str, bool* value );
632     static bool ToFloat( const char* str, float* value );
633     static bool ToDouble( const char* str, double* value );
634     static bool ToInt64(const char* str, int64_t* value);
635     static bool ToUnsigned64(const char* str, uint64_t* value);
636     // Changes what is serialized for a boolean value.
637     // Default to "true" and "false".  Shouldn't be changed
638     // unless you have a special testing or compatibility need.
639     // Be careful:  static, global, & not thread safe.
640     // Be sure to set static const memory as parameters.
641     static void SetBoolSerialization(const char* writeTrue, const char* writeFalse);
642
643 private:
644     static const char* writeBoolTrue;
645     static const char* writeBoolFalse;
646 };
647
648
674 class TINYXML2_LIB XMLNode
675 {
676     friend class XMLDocument;
677     friend class XMLElement;
678 public:
679
681     const XMLDocument* GetDocument()const    {
682         TIXMLASSERT( _document );
683         return _document;
684     }
686     XMLDocument* GetDocument()               {
687         TIXMLASSERT( _document );
688         return _document;
689     }
690
692     virtual XMLElement*       ToElement()        {
693         return 0;
694     }
696     virtual XMLText*      ToText()       {
697         return 0;
698     }
700     virtual XMLComment*       ToComment()        {
701         return 0;
702     }
704     virtual XMLDocument* ToDocument()  {
705         return 0;
706     }
708     virtual XMLDeclaration*   ToDeclaration()    {
709         return 0;
710     }
712     virtual XMLUnknown*       ToUnknown()        {
```

```
713         return 0;
714     }
715
716     virtual const XMLElement*     ToElement()const       {
717         return 0;
718     }
719     virtual const XMLText*          ToText()const          {
720         return 0;
721     }
722     virtual const XMLComment*       ToComment()const       {
723         return 0;
724     }
725     virtual const XMLDocument*      ToDocument()const      {
726         return 0;
727     }
728     virtual const XMLDeclaration*   ToDeclaration()const   {
729         return 0;
730     }
731     virtual const XMLUnknown*       ToUnknown()const       {
732         return 0;
733     }
734
744     const char* Value() const;
745
749     void SetValue( const char* val, bool staticMem=false );
750
752     int GetLineNum()const { return _parseLineNum; }
753
755     const XMLNode*   Parent()const            {
756         return _parent;
757     }
758
759     XMLNode* Parent()                         {
760         return _parent;
761     }
762
764     bool NoChildren()const                    {
765         return !_firstChild;
766     }
767
769     const XMLNode*  FirstChild()const     {
770         return _firstChild;
771     }
772
773     XMLNode*     FirstChild()            {
774         return _firstChild;
775     }
776
780     const XMLElement* FirstChildElement( const char* name = 0 ) const;
781
782     XMLElement* FirstChildElement( const char* name = 0 )    {
783         return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->FirstChildElement( name ));
784     }
785
787     const XMLNode*   LastChild()const                       {
788         return _lastChild;
789     }
790
791     XMLNode*     LastChild()                                {
792         return _lastChild;
793     }
794
798     const XMLElement* LastChildElement( const char* name = 0 ) const;
799
800     XMLElement* LastChildElement( const char* name = 0 )     {
801         return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->LastChildElement(name) );
802     }
803
805     const XMLNode*   PreviousSibling()const                   {
806         return _prev;
807     }
808
809     XMLNode* PreviousSibling()                                {
810         return _prev;
811     }
812
814     const XMLElement*   PreviousSiblingElement( const char* name = 0 ) const ;
815
816     XMLElement* PreviousSiblingElement( const char* name = 0 ) {
817         return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->PreviousSiblingElement( name )
    );
818     }
819
821     const XMLNode*   NextSibling()const                       {
822         return _next;
823     }
824
```

```
825     XMLNode* NextSibling()                                  {
826         return _next;
827     }
828
830     const XMLElement*   NextSiblingElement( const char* name = 0 ) const;
831
832     XMLElement* NextSiblingElement( const char* name = 0 )  {
833         return const_cast<XMLElement*>(const_cast<const XMLNode*>(this)->NextSiblingElement( name ) );
834     }
835
843     XMLNode* InsertEndChild( XMLNode* addThis );
844
845     XMLNode* LinkEndChild( XMLNode* addThis )   {
846         return InsertEndChild( addThis );
847     }
855     XMLNode* InsertFirstChild( XMLNode* addThis );
864     XMLNode* InsertAfterChild( XMLNode* afterThis, XMLNode* addThis );
865
869     void DeleteChildren();
870
874     void DeleteChild( XMLNode* node );
875
885     virtual XMLNode* ShallowClone( XMLDocument* document ) const = 0;
886
900     XMLNode* DeepClone( XMLDocument* target ) const;
901
908     virtual bool ShallowEqual( const XMLNode* compare ) const = 0;
909
932     virtual bool Accept( XMLVisitor* visitor ) const = 0;
933
939     void SetUserData(void* userData) { _userData = userData; }
940
946     void* GetUserData()const            { return _userData; }
947
948 protected:
949     explicit XMLNode( XMLDocument* );
950     virtual ~XMLNode();
951
952     virtual char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr);
953
954     XMLDocument* _document;
955     XMLNode*     _parent;
956     mutable StrPair  _value;
957     int              _parseLineNum;
958
959     XMLNode*     _firstChild;
960     XMLNode*     _lastChild;
961
962     XMLNode*     _prev;
963     XMLNode*     _next;
964
965     void*           _userData;
966
967 private:
968     MemPool*     _memPool;
969     void Unlink( XMLNode* child );
970     static void DeleteNode( XMLNode* node );
971     void InsertChildPreamble( XMLNode* insertThis ) const;
972     const XMLElement* ToElementWithName( const char* name ) const;
973
974     XMLNode( const XMLNode& );    // not supported
975     XMLNode& operator=( const XMLNode& ); // not supported
976 };
977
978
991 class TINYXML2_LIB XMLText :  public XMLNode
992 {
993     friend class XMLDocument;
994 public:
995     virtual bool Accept( XMLVisitor* visitor ) const;
996
997     virtual XMLText* ToText()           {
998         return this;
999     }
1000     virtual const XMLText* ToText()const    {
1001         return this;
1002     }
1003
1005     void SetCData( bool isCData )           {
1006         _isCData = isCData;
1007     }
1009     bool CData()const                   {
1010         return _isCData;
1011     }
1012
1013     virtual XMLNode* ShallowClone( XMLDocument* document ) const;
1014     virtual bool ShallowEqual( const XMLNode* compare ) const;
```

```
1015
1016 protected:
1017     explicit XMLText( XMLDocument* doc )   :  XMLNode( doc ), _isCData( false )  {}
1018     virtual ~XMLText()                                                          {}
1019
1020     char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr );
1021
1022 private:
1023     bool _isCData;
1024
1025     XMLText( const XMLText& );  // not supported
1026     XMLText& operator=( const XMLText& );   // not supported
1027 };
1028
1029
1031 class TINYXML2_LIB XMLComment :  public XMLNode
1032 {
1033     friend class XMLDocument;
1034 public:
1035     virtual XMLComment*   ToComment()                     {
1036         return this;
1037     }
1038     virtual const XMLComment* ToComment()const       {
1039         return this;
1040     }
1041
1042     virtual bool Accept( XMLVisitor* visitor ) const;
1043
1044     virtual XMLNode* ShallowClone( XMLDocument* document ) const;
1045     virtual bool ShallowEqual( const XMLNode* compare ) const;
1046
1047 protected:
1048     explicit XMLComment( XMLDocument* doc );
1049     virtual ~XMLComment();
1050
1051     char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr);
1052
1053 private:
1054     XMLComment( const XMLComment& );    // not supported
1055     XMLComment& operator=( const XMLComment& ); // not supported
1056 };
1057
1058
1070 class TINYXML2_LIB XMLDeclaration :  public XMLNode
1071 {
1072     friend class XMLDocument;
1073 public:
1074     virtual XMLDeclaration*   ToDeclaration()                     {
1075         return this;
1076     }
1077     virtual const XMLDeclaration* ToDeclaration()const       {
1078         return this;
1079     }
1080
1081     virtual bool Accept( XMLVisitor* visitor ) const;
1082
1083     virtual XMLNode* ShallowClone( XMLDocument* document ) const;
1084     virtual bool ShallowEqual( const XMLNode* compare ) const;
1085
1086 protected:
1087     explicit XMLDeclaration( XMLDocument* doc );
1088     virtual ~XMLDeclaration();
1089
1090     char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr );
1091
1092 private:
1093     XMLDeclaration( const XMLDeclaration& );    // not supported
1094     XMLDeclaration& operator=( const XMLDeclaration& ); // not supported
1095 };
1096
1097
1105 class TINYXML2_LIB XMLUnknown :  public XMLNode
1106 {
1107     friend class XMLDocument;
1108 public:
1109     virtual XMLUnknown*   ToUnknown()                     {
1110         return this;
1111     }
1112     virtual const XMLUnknown* ToUnknown()const       {
1113         return this;
1114     }
1115
1116     virtual bool Accept( XMLVisitor* visitor ) const;
1117
1118     virtual XMLNode* ShallowClone( XMLDocument* document ) const;
1119     virtual bool ShallowEqual( const XMLNode* compare ) const;
1120
```

```
1121 protected:
1122     explicit XMLUnknown( XMLDocument* doc );
1123     virtual ~XMLUnknown();
1124 
1125     char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr );
1126 
1127 private:
1128     XMLUnknown( const XMLUnknown& );    // not supported
1129     XMLUnknown& operator=( const XMLUnknown& ); // not supported
1130 };
1131 
1132 
1133 
1140 class TINYXML2_LIB XMLAttribute
1141 {
1142     friend class XMLElement;
1143 public:
1145     const char* Name() const;
1146 
1148     const char* Value() const;
1149 
1151     int GetLineNum()const { return _parseLineNum; }
1152 
1154     const XMLAttribute* Next()const {
1155         return _next;
1156     }
1157 
1162     int IntValue()const {
1163         int i = 0;
1164         QueryIntValue(&i);
1165         return i;
1166     }
1167 
1168     int64_t Int64Value()const {
1169         int64_t i = 0;
1170         QueryInt64Value(&i);
1171         return i;
1172     }
1173 
1174     uint64_t Unsigned64Value()const {
1175         uint64_t i = 0;
1176         QueryUnsigned64Value(&i);
1177         return i;
1178     }
1179 
1181     unsigned UnsignedValue()const         {
1182         unsigned i=0;
1183         QueryUnsignedValue( &i );
1184         return i;
1185     }
1187     bool     BoolValue()const             {
1188         bool b=false;
1189         QueryBoolValue( &b );
1190         return b;
1191     }
1193     double   DoubleValue()const         {
1194         double d=0;
1195         QueryDoubleValue( &d );
1196         return d;
1197     }
1199     float    FloatValue()const               {
1200         float f=0;
1201         QueryFloatValue( &f );
1202         return f;
1203     }
1204 
1209     XMLError QueryIntValue( int* value ) const;
1211     XMLError QueryUnsignedValue( unsigned int* value ) const;
1213     XMLError QueryInt64Value(int64_t* value) const;
1215     XMLError QueryUnsigned64Value(uint64_t* value) const;
1217     XMLError QueryBoolValue( bool* value ) const;
1219     XMLError QueryDoubleValue( double* value ) const;
1221     XMLError QueryFloatValue( float* value ) const;
1222 
1224     void SetAttribute( const char* value );
1226     void SetAttribute( int value );
1228     void SetAttribute( unsigned value );
1230     void SetAttribute(int64_t value);
1232     void SetAttribute(uint64_t value);
1234     void SetAttribute( bool value );
1236     void SetAttribute( double value );
1238     void SetAttribute( float value );
1239 
1240 private:
1241     enum { BUF_SIZE = 200 };
1242 
1243     XMLAttribute() :  _name(), _value(),_parseLineNum( 0 ), _next( 0 ), _memPool( 0 ) {}
```

```
1244     virtual ~XMLAttribute() {}
1245
1246     XMLAttribute( const XMLAttribute& );    // not supported
1247     void operator=( const XMLAttribute& );  // not supported
1248     void SetName( const char* name );
1249
1250     char* ParseDeep( char* p, bool processEntities, int* curLineNumPtr );
1251
1252     mutable StrPair _name;
1253     mutable StrPair _value;
1254     int             _parseLineNum;
1255     XMLAttribute*   _next;
1256     MemPool*        _memPool;
1257 };
1258
1259
1264 class TINYXML2_LIB XMLElement :  public XMLNode
1265 {
1266     friend class XMLDocument;
1267 public:
1269     const char* Name()const        {
1270         return Value();
1271     }
1273     void SetName( const char* str, bool staticMem=false )    {
1274         SetValue( str, staticMem );
1275     }
1276
1277     virtual XMLElement* ToElement()             {
1278         return this;
1279     }
1280     virtual const XMLElement* ToElement()const {
1281         return this;
1282     }
1283     virtual bool Accept( XMLVisitor* visitor ) const;
1284
1308     const char* Attribute( const char* name, const char* value=0 ) const;
1309
1316     int IntAttribute(const char* name, int defaultValue = 0) const;
1318     unsigned UnsignedAttribute(const char* name, unsigned defaultValue = 0) const;
1320     int64_t Int64Attribute(const char* name, int64_t defaultValue = 0) const;
1322     uint64_t Unsigned64Attribute(const char* name, uint64_t defaultValue = 0) const;
1324     bool BoolAttribute(const char* name, bool defaultValue = false) const;
1326     double DoubleAttribute(const char* name, double defaultValue = 0) const;
1328     float FloatAttribute(const char* name, float defaultValue = 0) const;
1329
1343     XMLError QueryIntAttribute( const char* name, int* value )const               {
1344         const XMLAttribute* a = FindAttribute( name );
1345         if ( !a ) {
1346             return XML_NO_ATTRIBUTE;
1347         }
1348         return a->QueryIntValue( value );
1349     }
1350
1352     XMLError QueryUnsignedAttribute( const char* name, unsigned int* value )const    {
1353         const XMLAttribute* a = FindAttribute( name );
1354         if ( !a ) {
1355             return XML_NO_ATTRIBUTE;
1356         }
1357         return a->QueryUnsignedValue( value );
1358     }
1359
1361     XMLError QueryInt64Attribute(const char* name, int64_t* value)const {
1362         const XMLAttribute* a = FindAttribute(name);
1363         if (!a) {
1364             return XML_NO_ATTRIBUTE;
1365         }
1366         return a->QueryInt64Value(value);
1367     }
1368
1370     XMLError QueryUnsigned64Attribute(const char* name, uint64_t* value)const {
1371         const XMLAttribute* a = FindAttribute(name);
1372         if(!a) {
1373             return XML_NO_ATTRIBUTE;
1374         }
1375         return a->QueryUnsigned64Value(value);
1376     }
1377
1379     XMLError QueryBoolAttribute( const char* name, bool* value )const                {
1380         const XMLAttribute* a = FindAttribute( name );
1381         if ( !a ) {
1382             return XML_NO_ATTRIBUTE;
1383         }
1384         return a->QueryBoolValue( value );
1385     }
1387     XMLError QueryDoubleAttribute( const char* name, double* value )const          {
1388         const XMLAttribute* a = FindAttribute( name );
1389         if ( !a ) {
```

```
1390            return XML_NO_ATTRIBUTE;
1391        }
1392        return a->QueryDoubleValue( value );
1393    }
1395    XMLError QueryFloatAttribute( const char* name, float* value )const         {
1396        const XMLAttribute* a = FindAttribute( name );
1397        if ( !a ) {
1398            return XML_NO_ATTRIBUTE;
1399        }
1400        return a->QueryFloatValue( value );
1401    }
1402
1404    XMLError QueryStringAttribute(const char* name, const char** value)const {
1405        const XMLAttribute* a = FindAttribute(name);
1406        if (!a) {
1407            return XML_NO_ATTRIBUTE;
1408        }
1409        *value = a->Value();
1410        return XML_SUCCESS;
1411    }
1412
1413
1414
1432    XMLError QueryAttribute( const char* name, int* value )const {
1433        return QueryIntAttribute( name, value );
1434    }
1435
1436    XMLError QueryAttribute( const char* name, unsigned int* value )const {
1437        return QueryUnsignedAttribute( name, value );
1438    }
1439
1440    XMLError QueryAttribute(const char* name, int64_t* value)const {
1441        return QueryInt64Attribute(name, value);
1442    }
1443
1444    XMLError QueryAttribute(const char* name, uint64_t* value)const {
1445        return QueryUnsigned64Attribute(name, value);
1446    }
1447
1448    XMLError QueryAttribute( const char* name, bool* value )const {
1449        return QueryBoolAttribute( name, value );
1450    }
1451
1452    XMLError QueryAttribute( const char* name, double* value )const {
1453        return QueryDoubleAttribute( name, value );
1454    }
1455
1456    XMLError QueryAttribute( const char* name, float* value )const {
1457        return QueryFloatAttribute( name, value );
1458    }
1459
1460    XMLError QueryAttribute(const char* name, const char** value)const {
1461        return QueryStringAttribute(name, value);
1462    }
1463
1465    void SetAttribute( const char* name, const char* value )    {
1466        XMLAttribute* a = FindOrCreateAttribute( name );
1467        a->SetAttribute( value );
1468    }
1470    void SetAttribute( const char* name, int value )            {
1471        XMLAttribute* a = FindOrCreateAttribute( name );
1472        a->SetAttribute( value );
1473    }
1475    void SetAttribute( const char* name, unsigned value )       {
1476        XMLAttribute* a = FindOrCreateAttribute( name );
1477        a->SetAttribute( value );
1478    }
1479
1481    void SetAttribute(const char* name, int64_t value) {
1482        XMLAttribute* a = FindOrCreateAttribute(name);
1483        a->SetAttribute(value);
1484    }
1485
1487    void SetAttribute(const char* name, uint64_t value) {
1488        XMLAttribute* a = FindOrCreateAttribute(name);
1489        a->SetAttribute(value);
1490    }
1491
1493    void SetAttribute( const char* name, bool value )           {
1494        XMLAttribute* a = FindOrCreateAttribute( name );
1495        a->SetAttribute( value );
1496    }
1498    void SetAttribute( const char* name, double value )     {
1499        XMLAttribute* a = FindOrCreateAttribute( name );
1500        a->SetAttribute( value );
1501    }
1503    void SetAttribute( const char* name, float value )      {
```

```
1504         XMLAttribute* a = FindOrCreateAttribute( name );
1505         a->SetAttribute( value );
1506     }
1507
1511     void DeleteAttribute( const char* name );
1512
1514     const XMLAttribute* FirstAttribute()const {
1515         return _rootAttribute;
1516     }
1518     const XMLAttribute* FindAttribute( const char* name ) const;
1519
1548     const char* GetText() const;
1549
1584     void SetText( const char* inText );
1586     void SetText( int value );
1588     void SetText( unsigned value );
1590     void SetText(int64_t value);
1592     void SetText(uint64_t value);
1594     void SetText( bool value );
1596     void SetText( double value );
1598     void SetText( float value );
1599
1626     XMLError QueryIntText( int* ival ) const;
1628     XMLError QueryUnsignedText( unsigned* uval ) const;
1630     XMLError QueryInt64Text(int64_t* uval) const;
1632     XMLError QueryUnsigned64Text(uint64_t* uval) const;
1634     XMLError QueryBoolText( bool* bval ) const;
1636     XMLError QueryDoubleText( double* dval ) const;
1638     XMLError QueryFloatText( float* fval ) const;
1639
1640     int IntText(int defaultValue = 0) const;
1641
1643     unsigned UnsignedText(unsigned defaultValue = 0) const;
1645     int64_t Int64Text(int64_t defaultValue = 0) const;
1647     uint64_t Unsigned64Text(uint64_t defaultValue = 0) const;
1649     bool BoolText(bool defaultValue = false) const;
1651     double DoubleText(double defaultValue = 0) const;
1653     float FloatText(float defaultValue = 0) const;
1654
1659     XMLElement* InsertNewChildElement(const char* name);
1661     XMLComment* InsertNewComment(const char* comment);
1663     XMLText* InsertNewText(const char* text);
1665     XMLDeclaration* InsertNewDeclaration(const char* text);
1667     XMLUnknown* InsertNewUnknown(const char* text);
1668
1669
1670     // internal:
1671     enum ElementClosingType {
1672         OPEN,       // <foo>
1673         CLOSED,     // <foo/>
1674         CLOSING     // </foo>
1675     };
1676     ElementClosingType ClosingType()const {
1677         return _closingType;
1678     }
1679     virtual XMLNode* ShallowClone( XMLDocument* document ) const;
1680     virtual bool ShallowEqual( const XMLNode* compare ) const;
1681
1682 protected:
1683     char* ParseDeep( char* p, StrPair* parentEndTag, int* curLineNumPtr );
1684
1685 private:
1686     XMLElement( XMLDocument* doc );
1687     virtual ~XMLElement();
1688     XMLElement( const XMLElement& );    // not supported
1689     void operator=( const XMLElement& );    // not supported
1690
1691     XMLAttribute* FindOrCreateAttribute( const char* name );
1692     char* ParseAttributes( char* p, int* curLineNumPtr );
1693     static void DeleteAttribute( XMLAttribute* attribute );
1694     XMLAttribute* CreateAttribute();
1695
1696     enum { BUF_SIZE = 200 };
1697     ElementClosingType _closingType;
1698     // The attribute list is ordered; there is no 'lastAttribute'
1699     // because the list needs to be scanned for dupes before adding
1700     // a new attribute.
1701     XMLAttribute* _rootAttribute;
1702 };
1703
1704
1705 enum Whitespace {
1706     PRESERVE_WHITESPACE,
1707     COLLAPSE_WHITESPACE
1708 };
1709
1710
```

```
1716 class TINYXML2_LIB XMLDocument :  public XMLNode
1717 {
1718     friend class XMLElement;
1719     // Gives access to SetError and Push/PopDepth, but over-access for everything else.
1720     // Wishing C++ had "internal" scope.
1721     friend class XMLNode;
1722     friend class XMLText;
1723     friend class XMLComment;
1724     friend class XMLDeclaration;
1725     friend class XMLUnknown;
1726 public:
1728     XMLDocument( bool processEntities = true, Whitespace whitespaceMode = PRESERVE_WHITESPACE );
1729     ~XMLDocument();
1730
1731     virtual XMLDocument* ToDocument()             {
1732         TIXMLASSERT( this == _document );
1733         return this;
1734     }
1735     virtual const XMLDocument* ToDocument()const    {
1736         TIXMLASSERT( this == _document );
1737         return this;
1738     }
1739
1750     XMLError Parse( const char* xml, size_t nBytes=static_cast<size_t>(-1) );
1751
1757     XMLError LoadFile( const char* filename );
1758
1770     XMLError LoadFile( FILE* );
1771
1777     XMLError SaveFile( const char* filename, bool compact = false );
1778
1786     XMLError SaveFile( FILE* fp, bool compact = false );
1787
1788     bool ProcessEntities()const       {
1789         return _processEntities;
1790     }
1791     Whitespace WhitespaceMode()const   {
1792         return _whitespaceMode;
1793     }
1794
1798     bool HasBOM()const {
1799         return _writeBOM;
1800     }
1803     void SetBOM( bool useBOM ) {
1804         _writeBOM = useBOM;
1805     }
1806
1810     XMLElement* RootElement()             {
1811         return FirstChildElement();
1812     }
1813     const XMLElement* RootElement()const {
1814         return FirstChildElement();
1815     }
1816
1831     void Print( XMLPrinter* streamer=0 ) const;
1832     virtual bool Accept( XMLVisitor* visitor ) const;
1833
1839     XMLElement* NewElement( const char* name );
1845     XMLComment* NewComment( const char* comment );
1851     XMLText* NewText( const char* text );
1863     XMLDeclaration* NewDeclaration( const char* text=0 );
1869     XMLUnknown* NewUnknown( const char* text );
1870
1875     void DeleteNode( XMLNode* node );
1876
1878     void ClearError();
1879
1881     bool Error()const {
1882         return _errorID != XML_SUCCESS;
1883     }
1885     XMLError  ErrorID()const {
1886         return _errorID;
1887     }
1888     const char* ErrorName() const;
1889     static const char* ErrorIDToName(XMLError errorID);
1890
1894     const char* ErrorStr() const;
1895
1897     void PrintError() const;
1898
1900     int ErrorLineNum()const
1901 {
1902         return _errorLineNum;
1903     }
1904
1906     void Clear();
1907
```

```
1915     void DeepCopy(XMLDocument* target) const;
1916
1917     // internal
1918     char* Identify( char* p, XMLNode** node );
1919
1920     // internal
1921     void MarkInUse(const XMLNode* const);
1922
1923     virtual XMLNode* ShallowClone( XMLDocument* /*document*/ )const  {
1924         return 0;
1925     }
1926     virtual bool ShallowEqual( const XMLNode* /*compare*/ )const    {
1927         return false;
1928     }
1929
1930 private:
1931     XMLDocument( const XMLDocument& );    // not supported
1932     void operator=( const XMLDocument& );    // not supported
1933
1934     bool            _writeBOM;
1935     bool            _processEntities;
1936     XMLError        _errorID;
1937     Whitespace      _whitespaceMode;
1938     mutable StrPair  _errorStr;
1939     int             _errorLineNum;
1940     char*           _charBuffer;
1941     int             _parseCurLineNum;
1942     int             _parsingDepth;
1943     // Memory tracking does add some overhead.
1944     // However, the code assumes that you don't
1945     // have a bunch of unlinked nodes around.
1946     // Therefore it takes less memory to track
1947     // in the document vs.  a linked list in the XMLNode,
1948     // and the performance is the same.
1949     DynArray<XMLNode*, 10> _unlinked;
1950
1951     MemPoolT< sizeof(XMLElement) >     _elementPool;
1952     MemPoolT< sizeof(XMLAttribute) > _attributePool;
1953     MemPoolT< sizeof(XMLText) >       _textPool;
1954     MemPoolT< sizeof(XMLComment) >     _commentPool;
1955
1956     static const char* _errorNames[XML_ERROR_COUNT];
1957
1958     void Parse();
1959
1960     void SetError( XMLError error, int lineNum, const char* format, ...  );
1961
1962     // Something of an obvious security hole, once it was discovered.
1963     // Either an ill-formed XML or an excessively deep one can overflow
1964     // the stack.  Track stack depth, and error out if needed.
1965     class DepthTracker {
1966     public:
1967         explicit DepthTracker(XMLDocument * document) {
1968             this->_document = document;
1969             document->PushDepth();
1970         }
1971         ~DepthTracker() {
1972             _document->PopDepth();
1973         }
1974     private:
1975         XMLDocument * _document;
1976     };
1977     void PushDepth();
1978     void PopDepth();
1979
1980     template<class NodeType, int PoolElementSize>
1981     NodeType* CreateUnlinkedNode( MemPoolT<PoolElementSize>& pool );
1982 };
1983
1984 template<class NodeType, int PoolElementSize>
1985 inline NodeType* XMLDocument::CreateUnlinkedNode( MemPoolT<PoolElementSize>& pool )
1986 {
1987     TIXMLASSERT( sizeof( NodeType ) == PoolElementSize );
1988     TIXMLASSERT( sizeof( NodeType ) == pool.ItemSize() );
1989     NodeType* returnNode = new (pool.Alloc()) NodeType( this );
1990     TIXMLASSERT( returnNode );
1991     returnNode->_memPool = &pool;
1992
1993     _unlinked.Push(returnNode);
1994     return returnNode;
1995 }
1996
2052 class TINYXML2_LIB XMLHandle
2053 {
2054 public:
2056     explicit XMLHandle( XMLNode* node ) :  _node( node ) {
2057     }
```

```
2059     explicit XMLHandle( XMLNode& node ) :  _node( &node ) {
2060     }
2062     XMLHandle( const XMLHandle& ref ) :  _node( ref._node ) {
2063     }
2065     XMLHandle& operator=( const XMLHandle& ref )                          {
2066         _node = ref._node;
2067         return *this;
2068     }
2069
2071     XMLHandle FirstChild()                                                  {
2072         return XMLHandle( _node ?  _node->FirstChild() :  0 );
2073     }
2075     XMLHandle FirstChildElement( const char* name = 0 )                      {
2076         return XMLHandle( _node ?  _node->FirstChildElement( name ) :  0 );
2077     }
2079     XMLHandle LastChild()                                                   {
2080         return XMLHandle( _node ?  _node->LastChild() :  0 );
2081     }
2083     XMLHandle LastChildElement( const char* name = 0 )                     {
2084         return XMLHandle( _node ?  _node->LastChildElement( name ) :  0 );
2085     }
2087     XMLHandle PreviousSibling()                                             {
2088         return XMLHandle( _node ?  _node->PreviousSibling() :  0 );
2089     }
2091     XMLHandle PreviousSiblingElement( const char* name = 0 )              {
2092         return XMLHandle( _node ?  _node->PreviousSiblingElement( name ) :  0 );
2093     }
2095     XMLHandle NextSibling()                                                 {
2096         return XMLHandle( _node ?  _node->NextSibling() :  0 );
2097     }
2099     XMLHandle NextSiblingElement( const char* name = 0 )                  {
2100         return XMLHandle( _node ?  _node->NextSiblingElement( name ) :  0 );
2101     }
2102
2104     XMLNode* ToNode()                           {
2105         return _node;
2106     }
2108     XMLElement* ToElement()                   {
2109         return ( _node ?  _node->ToElement() :  0 );
2110     }
2112     XMLText* ToText()                           {
2113         return ( _node ?  _node->ToText() :  0 );
2114     }
2116     XMLUnknown* ToUnknown()                   {
2117         return ( _node ?  _node->ToUnknown() :  0 );
2118     }
2120     XMLDeclaration* ToDeclaration()           {
2121         return ( _node ?  _node->ToDeclaration() :  0 );
2122     }
2123
2124 private:
2125     XMLNode* _node;
2126 };
2127
2128
2133 class TINYXML2_LIB XMLConstHandle
2134 {
2135 public:
2136     explicit XMLConstHandle( const XMLNode* node ) :  _node( node ) {
2137     }
2138     explicit XMLConstHandle( const XMLNode& node ) :  _node( &node ) {
2139     }
2140     XMLConstHandle( const XMLConstHandle& ref ) :  _node( ref._node ) {
2141     }
2142
2143     XMLConstHandle& operator=( const XMLConstHandle& ref )                          {
2144         _node = ref._node;
2145         return *this;
2146     }
2147
2148     const XMLConstHandle FirstChild()const                                           {
2149         return XMLConstHandle( _node ?  _node->FirstChild() :  0 );
2150     }
2151     const XMLConstHandle FirstChildElement( const char* name = 0 )const              {
2152         return XMLConstHandle( _node ?  _node->FirstChildElement( name ) :  0 );
2153     }
2154     const XMLConstHandle LastChild()const                                          {
2155         return XMLConstHandle( _node ?  _node->LastChild() :  0 );
2156     }
2157     const XMLConstHandle LastChildElement( const char* name = 0 )const               {
2158         return XMLConstHandle( _node ?  _node->LastChildElement( name ) :  0 );
2159     }
2160     const XMLConstHandle PreviousSibling()const                                    {
2161         return XMLConstHandle( _node ?  _node->PreviousSibling() :  0 );
2162     }
2163     const XMLConstHandle PreviousSiblingElement( const char* name = 0 )const     {
2164         return XMLConstHandle( _node ?  _node->PreviousSiblingElement( name ) :  0 );
```

```
2165     }
2166     const XMLConstHandle NextSibling()const                                    {
2167         return XMLConstHandle( _node ?  _node->NextSibling() :  0 );
2168     }
2169     const XMLConstHandle NextSiblingElement( const char* name = 0 )const       {
2170         return XMLConstHandle( _node ?  _node->NextSiblingElement( name ) :  0 );
2171     }
2172
2173
2174     const XMLNode* ToNode()const               {
2175         return _node;
2176     }
2177     const XMLElement* ToElement()const          {
2178         return ( _node ?  _node->ToElement() :  0 );
2179     }
2180     const XMLText* ToText()const               {
2181         return ( _node ?  _node->ToText() :  0 );
2182     }
2183     const XMLUnknown* ToUnknown()const          {
2184         return ( _node ?  _node->ToUnknown() :  0 );
2185     }
2186     const XMLDeclaration* ToDeclaration()const  {
2187         return ( _node ?  _node->ToDeclaration() :  0 );
2188     }
2189
2190 private:
2191     const XMLNode* _node;
2192 };
2193
2194
2237 class TINYXML2_LIB XMLPrinter :  public XMLVisitor
2238 {
2239 public:
2246     XMLPrinter( FILE* file=0, bool compact = false, int depth = 0 );
2247     virtual ~XMLPrinter()    {}
2248
2250     void PushHeader( bool writeBOM, bool writeDeclaration );
2254     void OpenElement( const char* name, bool compactMode=false );
2256     void PushAttribute( const char* name, const char* value );
2257     void PushAttribute( const char* name, int value );
2258     void PushAttribute( const char* name, unsigned value );
2259     void PushAttribute( const char* name, int64_t value );
2260     void PushAttribute( const char* name, uint64_t value );
2261     void PushAttribute( const char* name, bool value );
2262     void PushAttribute( const char* name, double value );
2264     virtual void CloseElement( bool compactMode=false );
2265
2267     void PushText( const char* text, bool cdata=false );
2269     void PushText( int value );
2271     void PushText( unsigned value );
2273     void PushText( int64_t value );
2275     void PushText( uint64_t value );
2277     void PushText( bool value );
2279     void PushText( float value );
2281     void PushText( double value );
2282
2284     void PushComment( const char* comment );
2285
2286     void PushDeclaration( const char* value );
2287     void PushUnknown( const char* value );
2288
2289     virtual bool VisitEnter( const XMLDocument& /*doc*/ );
2290     virtual bool VisitExit( const XMLDocument& /*doc*/ )            {
2291         return true;
2292     }
2293
2294     virtual bool VisitEnter( const XMLElement& element, const XMLAttribute* attribute );
2295     virtual bool VisitExit( const XMLElement& element );
2296
2297     virtual bool Visit( const XMLText& text );
2298     virtual bool Visit( const XMLComment& comment );
2299     virtual bool Visit( const XMLDeclaration& declaration );
2300     virtual bool Visit( const XMLUnknown& unknown );
2301
2306     const char* CStr()const {
2307         return _buffer.Mem();
2308     }
2314     int CStrSize()const {
2315         return _buffer.Size();
2316     }
2321     void ClearBuffer( bool resetToFirstElement = true ) {
2322         _buffer.Clear();
2323         _buffer.Push(0);
2324         _firstElement = resetToFirstElement;
2325     }
2326
2327 protected:
```

```
2328     virtual bool CompactMode( const XMLElement& ) { return _compactMode; }
2329
2333     virtual void PrintSpace( int depth );
2334     virtual void Print( const char* format, ... );
2335     virtual void Write( const char* data, size_t size );
2336     virtual void Putc( char ch );
2337
2338     inline void Write(const char* data) { Write(data, strlen(data)); }
2339
2340     void SealElementIfJustOpened();
2341     bool _elementJustOpened;
2342     DynArray< const char*, 10 > _stack;
2343
2344 private:
2349     void PrepareForNewNode( bool compactMode );
2350     void PrintString( const char*, bool restrictedEntitySet );  // prints out, after detecting
      entities.
2351
2352     bool _firstElement;
2353     FILE* _fp;
2354     int _depth;
2355     int _textDepth;
2356     bool _processEntities;
2357     bool _compactMode;
2358
2359     enum {
2360         ENTITY_RANGE = 64,
2361         BUF_SIZE = 200
2362     };
2363     bool _entityFlag[ENTITY_RANGE];
2364     bool _restrictedEntityFlag[ENTITY_RANGE];
2365
2366     DynArray< char, 20 > _buffer;
2367
2368     // Prohibit cloning, intentionally not implemented
2369     XMLPrinter( const XMLPrinter& );
2370     XMLPrinter& operator=( const XMLPrinter& );
2371 };
2372
2373
2374 }   // tinyxml2
2375
2376 #if defined(_MSC_VER)
2377 #   pragma warning(pop)
2378 #endif
2379
2380 #endif // TINYXML2_INCLUDED
```

Index