MindEngine

Generated by Doxygen 1.8.11

Contents

Chapter 1

Todo List

Member RuleBased::DataGroupMatch::matchesNode (const DataNode *node, void *bindings)

Need adding bindings mechanism here

Member RuleBased::IdType

Use identifier with strings may decrease performance with large systems due to string-matching operations. Will change this to integer or unsigned integer as soon as I find a way to match identifier with human-readable strings.

Member RuleBased::NumberDatumMatch< T >::matchesNode (const DataNode *node, void *bindings)

Need adding bindings mechanism here

Member RuleBased::Rule::action ()=0

Examine the performance and reusability of using a method. If it is hard to expand, find a way to use a struct/class instead.

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Namespace Documentation

5.1 RuleBased Namespace Reference

Contains classes to represent Rule-based system's database, which stores knowledge available to the AI agent as well as the implementation of rules and the mechanism to match the rules and the data in the database.

Classes

- class DataGroup

  Represents a non-leaf node, which contains children. Its children can be any DataNode object: either another DataGroup, or a Datum only.
- struct DataGroupMatch

  Matches a group of data in the database. This is done by building a match data structure that mirrors the data structure that is being searched for in the database.
- class DataNode

  Base class of each node in the database tree. Since every node needs an identifier, but non-leaf nodes contain their children, while leaf nodes store values.
- struct DataNodeMatch

  A struct derived from Match, it is responsible for matching a single DataNode in the database.
- class Datum

  A Datum consists of an identifier and and value. In Database's tree structure, a leaf node is a Datum.
- class IdCheck

  Provides methods to check the IDs.
- struct Match

  Provides the mechanism to match the data item from the rule with any item inside the database.
- struct NumberDatumMatch

  A struct provides mechanism for matching two Datum whose values is number values (int, float,...).
- class Rule

  Represent a rule in a Rule-based system. A rule has two components: an if clause is going to be used to match against the database and a function to perform any action required.

Typedefs

- typedef std::string IdType

  Currently set the type of the ID of data nodes as string.

5.1.1 Detailed Description

Contains classes to represent Rule-based system's database, which stores knowledge available to the AI agent as well as the implementation of rules and the mechanism to match the rules and the data in the database.

5.1.2 Typedef Documentation

5.1.2.1 typedef std::string RuleBased::IdType

Currently set the type of the ID of data nodes as string.

Todo Use identifier with strings may decrease performance with large systems due to string-matching operations. Will change this to integer or unsigned integer as soon as I find a way to match identifier with human-readable strings.

Chapter 6

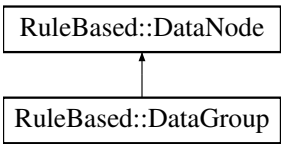
Class Documentation

6.1 RuleBased::DataGroup Class Reference

Represents a non-leaf node, which contains children. Its children can be any DataNode object: either another DataGroup, or a Datum only.

`#include <DataGroup.h>`

Inheritance diagram for RuleBased::DataGroup:



Public Member Functions

- DataGroup ()

  DataGroup default constructor.
- DataGroup (const std::string &identifier, DataNode *rightSibling, DataNode *leftMostChild)

  DataGroup constructor.
- virtual ~DataGroup ()

  ~DataGroup destructor.
- DataNode * getLeftMostChild () const

  Data nodes are put into a left-most child, right sibling tree. This function returns the pointer to the left most child of this node.
- bool isGroup ()

  Allows user to check whether this node is a DataGroup or not.

Additional Inherited Members

6.1.1 Detailed Description

Represents a non-leaf node, which contains children. Its children can be any DataNode object: either another DataGroup, or a Datum only.

See also

DataNode
Datum

6.1.2 Constructor & Destructor Documentation

6.1.2.1 RuleBased::DataGroup::DataGroup (const std::string & *identifier*, DataNode * *rightSibling*, DataNode * *leftMostChild*)

DataGroup constructor.

Parameters

identifier	a string parameter.
rightSibling	a DataNode pointer representing this node's right sibling in tree.
leftMostChild	a DataNode pointer representing this node's left most child in tree.

6.1.3 Member Function Documentation

6.1.3.1 DataNode * RuleBased::DataGroup::getLeftMostChild ( ) const

Data nodes are put into a left-most child, right sibling tree. This function returns the pointer to the left most child of this node.

Returns

The left most child of this data group node.

6.1.3.2 bool RuleBased::DataGroup::isGroup ( )

Allows user to check whether this node is a DataGroup or not.

Returns

true if this node is a DataGroup, otherwise returns false.

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

- Engine/Database/DataGroup.h
- Engine/Database/DataGroup.cpp

6.2 RuleBased::DataGroupMatch Struct Reference

Matches a group of data in the database. This is done by building a match data structure that mirrors the data structure that is being searched for in the database.

`#include <DataGroupMatch.h>`

Public Member Functions

- virtual bool matchesNode (const DataNode *node, void *bindings)

  Tries to match the given data node from the database against the criteria. Method used: a recursive matching algorithm that travels down the given node, trying to match it against the structure of this match and its children.

Public Attributes

- IdType identifier

  The identifier to match.
- DataNodeMatch * leftMostChild

  The first sub-match in this group.

6.2.1 Detailed Description

Matches a group of data in the database. This is done by building a match data structure that mirrors the data structure that is being searched for in the database.

6.2.2 Member Function Documentation

6.2.2.1 bool RuleBased::DataGroupMatch::matchesNode (const DataNode * *node*, void * *bindings*) [virtual]

Tries to match the given data node from the database against the criteria. Method used: a recursive matching algorithm that travels down the given node, trying to match it against the structure of this match and its children.

Parameters

node	The database node to match on.
bindings	When part of the if clause matches a wild card, it is added to the bindings. This parameter is both input and output parameter.

Returns

true if matches, otherwise return false.

Todo Need adding bindings mechanism here

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

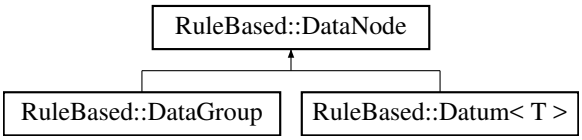
- Engine/Rules/DataGroupMatch.h
- Engine/Rules/DataGroupMatch.cpp

6.3 RuleBased::DataNode Class Reference

Base class of each node in the database tree. Since every node needs an identifier, but non-leaf nodes contain their children, while leaf nodes store values.

`#include <DataNode.h>`

Inheritance diagram for RuleBased::DataNode:



Public Member Functions

- DataNode ()
  DataNode default constructor.
- DataNode (const IdType &identifier, DataNode *rightSibling)
  DataNode constructor with parameters.
- virtual ~DataNode ()
  ~DataNode destructor.
- IdType getIdentifier () const
  The data nodes have unique identifiers.
- DataNode * getRightSibling () const
  Data nodes are put into a left-most child, right sibling tree. This function is used to get the sibling next to current node.
- virtual bool isGroup () const
  Allows user to check whether this node is a DataGroup or not.
- virtual bool isDatum () const
  Allows user to check whether this node is a Datum or not.

Protected Attributes

- IdType identifier
  Each item of data should be identified to know what the value means.
- DataNode * rightSibling
  The right sibling node of this node, or NULL if this node is the right most child.

6.3.1 Detailed Description

Base class of each node in the database tree. Since every node needs an identifier, but non-leaf nodes contain their children, while leaf nodes store values.

See also

DataGroup
Datum

6.3.2 Constructor & Destructor Documentation

6.3.2.1 RuleBased::DataNode::DataNode (const IdType & *identifier*, DataNode * *rightSibling*)

DataNode constructor with parameters.

Parameters

identifier	an ID parameter.
rightSibling	a DataNode pointer representing this node's right sibling in tree. structure

6.3.3 Member Function Documentation

6.3.3.1 IdType RuleBased::DataNode::getIdentifier ( ) const

The data nodes have unique identifiers.

Returns

The identifier of this node.

6.3.3.2 DataNode * RuleBased::DataNode::getRightSibling ( ) const

Data nodes are put into a left-most child, right sibling tree. This function is used to get the sibling next to current node.

Returns

The right sibling node of this node, or NULL if this node is the right most child.

6.3.3.3 bool RuleBased::DataNode::isDatum ( ) const [virtual]

Allows user to check whether this node is a Datum or not.

Returns

true if this node is a Datum, otherwise returns false.

6.3.3.4 bool RuleBased::DataNode::isGroup ( ) const [virtual]

Allows user to check whether this node is a DataGroup or not.

Returns

true if this node is a DataGroup, otherwise returns false.

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

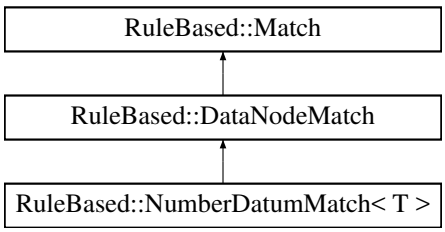
- Engine/Database/DataNode.h
- Engine/Database/DataNode.cpp

6.4 RuleBased::DataNodeMatch Struct Reference

A struct derived from Match, it is responsible for matching a single DataNode in the database.

`#include <DataNodeMatch.h>`

Inheritance diagram for RuleBased::DataNodeMatch:



Public Member Functions

- virtual bool matches (const DataNode *database, void *bindings)

  Matches the given database by checking each element in the database against the matchesNode method.
- bool matchesChildren (const DataGroup *group, void *bindings)

  Matches all the children of the given group to see if any of them pass the matchesNode test. This is used in the implementation of the matches method to iterate through items in the database.
- virtual bool matchesNode (const DataNode *node, void *bindings)=0

  Matches the data node from the database against the criteria in this match.

Public Attributes

- DataNodeMatch * rightSibling

  The right sibling of this node in match tree.

6.4.1 Detailed Description

A struct derived from Match, it is responsible for matching a single DataNode in the database.

Conceptually, the match class could match the whole database in a single operation: it is only ever fed the whole database, so it can do with that what it likes. However in practical, the vast majority of matching requirements involve trying to find a single item in the database. This struct's match method iterates through the items in the database, and calls matchesNode on each one. Matches items can be implemented in sub-classes to check if the item fulfils the mtach criteria.

Data node matches are arranged into tree, just like how data nodes are.

See also

Match

6.4.2 Member Function Documentation

6.4.2.1 bool RuleBased::DataNodeMatch::matches (const DataNode * *database*, void * *bindings*) [virtual]

Matches the given database by checking each element in the database against the matchesNode method.

Parameters

database	The database node to match on.
bindings	When part of the if clause matches a wild card, it is added to the bindings. This parameter is both input and output parameter.

Returns

true if matches, else returns false.

Implements RuleBased::Match.

6.4.2.2 bool RuleBased::DataNodeMatch::matchesChildren (const DataGroup * *group*, void * *bindings*)

Matches all the children of the given group to see if any of them pass the matchesNode test. This is used in the implementation of the matches method to iterate through items in the database.

Parameters

group	The group to match on.
bindings	When part of the if clause matches a wild card, it is added to the bindings. This parameter is both input and output parameter.

Returns

true if matches, else return false.

6.4.2.3 virtual bool RuleBased::DataNodeMatch::matchesNode (const DataNode * *node*, void * *bindings*) [pure virtual]

Matches the data node from the database against the criteria in this match.

Parameters

node	The database node to match on.
bindings	When part of the if clause matches a wild card, it is added to the bindings. This parameter is both input and output parameter.

Returns

true if matches, else return false.

Implemented in RuleBased::NumberDatumMatch< T >.

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

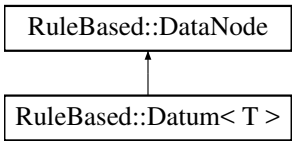
- Engine/Rules/DataNodeMatch.h
- Engine/Rules/DataNodeMatch.cpp

6.5 RuleBased::Datum< T > Class Template Reference

A Datum consists of an identifier and and value. In Database's tree structure, a leaf node is a Datum.

`#include <Datum.h>`

Inheritance diagram for RuleBased::Datum< T >:



Public Member Functions

- Datum (T value)
  Datum constructor.
- Datum (const std::string &identifier, DataNode *rightSibling, T value)
  Datum constructor.
- virtual ~Datum ()
  ~Datum destructor
- void setValue (T newValue)
  To change the value of the Datum.
- T getValue () const
  To get the value of the Datum.
- bool isDatum ()
  Allows user to check whether this node is a Datum or not.

Additional Inherited Members

6.5.1 Detailed Description

template<class T>
class RuleBased::Datum< T >

A Datum consists of an identifier and and value. In Database's tree structure, a leaf node is a Datum.

See also

DataNode
DataGroup

6.5.2 Constructor & Destructor Documentation

6.5.2.1 template<class T > RuleBased::Datum< T >::Datum (T *value*)

Datum constructor.

Parameters

value	the value that the Datum holds.

6.5.2.2 template<class T > RuleBased::Datum< T >::Datum (const std::string & *identifier*, DataNode * *rightSibling*, T *value*)

Datum constructor.

Parameters

identifier	a string parameter.
rightSibling	a DataNode pointer representing this node's right sibling in tree.
value	the value that the Datum holds.

6.5.3 Member Function Documentation

6.5.3.1 template<class T > T RuleBased::Datum< T >::getValue ( ) const

To get the value of the Datum.

Returns

The value that the datum is currently holding.

6.5.3.2 template<class T > bool RuleBased::Datum< T >::isDatum ( )

Allows user to check whether this node is a Datum or not.

Returns

true if this node is a Datum, otherwise returns false.

6.5.3.3 template<class T > void RuleBased::Datum< T >::setValue (T *newValue*)

To change the value of the Datum.

Parameters

newValue	The new value that is going to be assigned to the Datum.

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

- Engine/Database/Datum.h
- Engine/Database/Datum.cpp

6.6 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
- const T * **Mem** () const
- T * **Mem** ()

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

- Libs/tinyxml2.h

6.7 tinyxml2::Entity Struct Reference

Public Attributes

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

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

- Libs/tinyxml2.cpp

6.8 RuleBased::IdCheck Class Reference

Provides methods to check the IDs.

`#include <IdCheck.h>`

Static Public Member Functions

- static bool isWildcard (IdType identifier)

  Check if the given database item's identifier is a wildcard or not.

6.8.1 Detailed Description

Provides methods to check the IDs.

6.8.2 Member Function Documentation

6.8.2.1 bool RuleBased::IdCheck::isWildcard (IdType *identifier*) [static]

Check if the given database item's identifier is a wildcard or not.

See also

IdType

Returns

True if it is a wildcard.

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

- Engine/Rules/IdCheck.h
- Engine/Rules/IdCheck.cpp

6.9 tinyxml2::LongFitsIntoSizeTMinusOne< bool > Struct Template Reference

Public Member Functions

- template<>
  bool **Fits** (unsigned long)

Static Public Member Functions

- static bool **Fits** (unsigned long value)

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

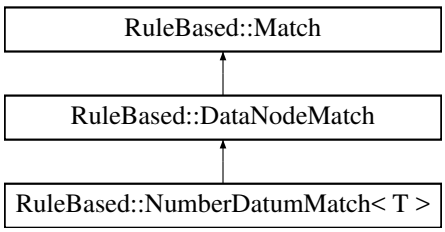
- Libs/tinyxml2.cpp

6.10 RuleBased::Match Struct Reference

Provides the mechanism to match the data item from the rule with any item inside the database.

`#include <Match.h>`

Inheritance diagram for RuleBased::Match:



Public Member Functions

- virtual bool matches (const DataNode *database, void *bindings)=0
  Check the match on the database.

6.10.1 Detailed Description

Provides the mechanism to match the data item from the rule with any item inside the database.

6.10.2 Member Function Documentation

6.10.2.1 virtual bool RuleBased::Match::matches (const DataNode * *database*, void * *bindings*) [pure virtual]

Check the match on the database.

Parameters

database	The database to match on.
bindings	When part of the if clause matches a wild card, it is added to the bindings. This parameter is both input and output parameter.

Returns

true if matches, else returns false.

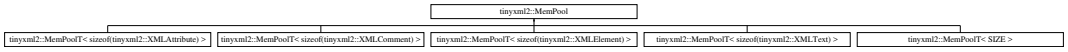
Implemented in RuleBased::DataNodeMatch.

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

- Engine/Rules/Match.h

6.11 tinyxml2::MemPool Class Reference

Inheritance diagram for tinyxml2::MemPool:



Public Member Functions

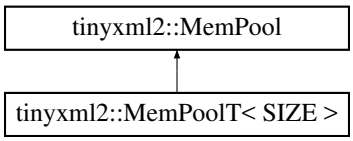
- virtual int **ItemSize** () const =0
- virtual void ∗ **Alloc** ()=0
- virtual void **Free** (void ∗)=0
- virtual void **SetTracked** ()=0
- virtual void **Clear** ()=0

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

- Libs/tinyxml2.h

6.12 tinyxml2::MemPoolT< SIZE > Class Template Reference

Inheritance diagram for tinyxml2::MemPoolT< SIZE >:



Public Types

- enum { **COUNT** = (4∗1024)/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

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

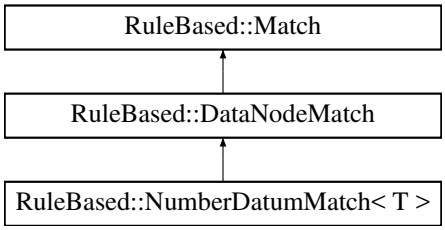
- Libs/tinyxml2.h

6.13 RuleBased::NumberDatumMatch< T > Struct Template Reference

A struct provides mechanism for matching two Datum whose values is number values (int, float,...).

`#include <NumberDatumMatch.h>`

Inheritance diagram for RuleBased::NumberDatumMatch< T >:



Public Member Functions

- virtual bool matchesNode (const DataNode *node, void *bindings)

  Matches the given database node.
- NumberDatumMatch (IdType identifier, T min, T max)

  Create a match object with the given identifier and range. This range-based approach allows you to match a range with more flexibility. For example, if you want to check the HP of the agent is lower than 80, you just create a matching range of [0, 79] (since both limits are inclusive).

Public Attributes

- T min

  The minimum value of the matching range (inclusive).
- T max

  The maximum value of the matching range (inclusive).
- IdType identifier

  The identifier to match.

6.13.1 Detailed Description

template<typename T>
struct RuleBased::NumberDatumMatch< T >

A struct provides mechanism for matching two Datum whose values is number values (int, float,...).

It uses the range-based approach, since every number values in game is limited in a specific range, for example: HP is from 0 to 100, ammo is from 0 to 50, and so on. That means, the comparision operators greater than, greater than or equal, equal, less than, less than or equal can be combined into a single match struct.

Note

This struct acts not only as the mechanism of matching two number Datum nodes, it also is the example of how to implement your own type of matching with other data structures.

See also

Match
DataNodeMatch

6.13.2 Constructor & Destructor Documentation

6.13.2.1 template<typename T > RuleBased::NumberDatumMatch< T >::NumberDatumMatch (IdType *identifier,* T *min,* T *max*)

Create a match object with the given identifier and range. This range-based approach allows you to match a range with more flexibility. For example, if you want to check the HP of the agent is lower than 80, you just create a matching range of [0, 79] (since both limits are inclusive).

Note

Remember to have max value greater than or equal to the min value.

6.13.3 Member Function Documentation

6.13.3.1 template<typename T > bool RuleBased::NumberDatumMatch< T >::matchesNode (const DataNode * *node,* void * *bindings*) [virtual]

Matches the given database node.

Parameters

node	The database node to match on.
bindings	When part of the if clause matches a wild card, it is added to the bindings. This parameter is both input and output parameter.

Returns

true if matches, else return false.

Todo Need adding bindings mechanism here

Implements RuleBased::DataNodeMatch.

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

- Engine/Rules/NumberDatumMatch.h
- Engine/Rules/NumberDatumMatch.cpp

6.14 RuleBased::Rule Class Reference

Represent a rule in a Rule-based system. A rule has two components: an if clause is going to be used to match against the database and a function to perform any action required.

`#include <Rule.h>`

Public Member Functions

- virtual void action ()=0

  The action is going to be carried out when the rule matches.

Public Attributes

- Match * ifClause

  Consist of a set of data items, in a similar format to those in the database.

6.14.1 Detailed Description

Represent a rule in a Rule-based system. A rule has two components: an if clause is going to be used to match against the database and a function to perform any action required.

6.14.2 Member Function Documentation

6.14.2.1 virtual void RuleBased::Rule::action ( ) [pure virtual]

The action is going to be carried out when the rule matches.

Todo Examine the performance and reusability of using a method. If it is hard to expand, find a way to use a struct/class instead.

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

- Engine/Rules/Rule.h

6.15 tinyxml2::StrPair Class Reference

Public Types

- enum {
  NEEDS_ENTITY_PROCESSING = 0x01, **NEEDS_NEWLINE_NORMALIZATION** = 0x02, **NEEDS_WHIT**↩
  ESPACE_COLLAPSING = 0x04, **TEXT_ELEMENT** = NEEDS_ENTITY_PROCESSING | NEEDS_NEWL↩
  INE_NORMALIZATION,
  TEXT_ELEMENT_LEAVE_ENTITIES = NEEDS_NEWLINE_NORMALIZATION, **ATTRIBUTE_NAME** = 0,
  ATTRIBUTE_VALUE = NEEDS_ENTITY_PROCESSING | NEEDS_NEWLINE_NORMALIZATION, **ATTR**↩
  IBUTE_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)
- char ∗ **ParseName** (char ∗in)
- void **TransferTo** (StrPair ∗other)

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

- Libs/tinyxml2.h
- Libs/tinyxml2.cpp

6.16 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.
- const XMLAttribute ∗ Next () const

  The next attribute in the list.
- int IntValue () 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 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 (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**

6.16.1 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.

6.16.2 Member Function Documentation

6.16.2.1 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.

6.16.2.2 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_NO_ERROR on success, and XML_WRONG_ATTRIBUTE_TYPE if the conversion is not successful.

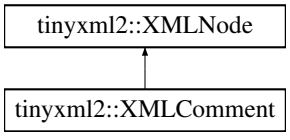
The documentation for this class was generated from the following files:

- Libs/tinyxml2.h
- Libs/tinyxml2.cpp

6.17 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 *, StrPair *endTag)

Friends

- class **XMLDocument**

Additional Inherited Members

6.17.1 Detailed Description

An XML Comment.

6.17.2 Member Function Documentation

6.17.2.1 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.

6.17.2.2 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.

6.17.2.3 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.

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

- Libs/tinyxml2.h
- Libs/tinyxml2.cpp

6.18 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

6.18.1 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.

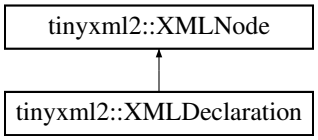
The documentation for this class was generated from the following file:

- Libs/tinyxml2.h

6.19 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 *, StrPair *endTag)

Friends

- class **XMLDocument**

Additional Inherited Members

6.19.1 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.

6.19.2 Member Function Documentation

6.19.2.1 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.

6.19.2.2 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.

6.19.2.3 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.

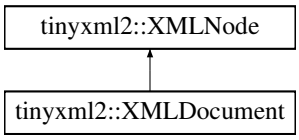
The documentation for this class was generated from the following files:

- Libs/tinyxml2.h
- Libs/tinyxml2.cpp

6.20 tinyxml2::XMLDocument Class Reference

`#include <tinyxml2.h>`

Inheritance diagram for tinyxml2::XMLDocument:



Public Member Functions

- XMLDocument (bool processEntities=true, Whitespace=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=(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 **SetError** (XMLError error, const char *str1, const char *str2)
- 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 * GetErrorStr1 () const
  Return a possibly helpful diagnostic location or string.
- const char * GetErrorStr2 () const
  Return a possibly helpful secondary diagnostic location or string.
- void PrintError () const
  If there is an error, print it to stdout.
- void Clear ()
  Clear the document, resetting it to the initial state.
- char * **Identify** (char *p, XMLNode **node)
- virtual XMLNode * ShallowClone (XMLDocument *) const
- virtual bool ShallowEqual (const XMLNode *) const

Friends

- class **XMLElement**

Additional Inherited Members

6.20.1 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.

6.20.2 Member Function Documentation

6.20.2.1 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.

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

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

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

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

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

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

6.20.2.5 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_NO_ERROR (0) on success, or an errorID.

6.20.2.6 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.

6.20.2.7 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"?>
```

6.20.2.8 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.

6.20.2.9 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.

6.20.2.10 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.

6.20.2.11 XMLError tinyxml2::XMLDocument::Parse (const char * *xml*, size_t *nBytes* = (size_t)(-1))

Parse an XML file from a character string. Returns XML_NO_ERROR (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, TinyX↩ML-2 will assume 'xml' points to a null terminated string.

6.20.2.12 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
```

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

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

6.20.2.14 XMLError tinyxml2::XMLDocument::SaveFile (const char * *filename*, bool *compact* = false)

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

6.20.2.15 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_NO_ERROR (0) on success, or an errorID.

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

Sets whether to write the BOM when writing the file.

6.20.2.17 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.

6.20.2.18 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.

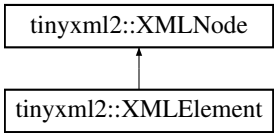
The documentation for this class was generated from the following files:

- Libs/tinyxml2.h
- Libs/tinyxml2.cpp

6.21 tinyxml2::XMLElement Class Reference

`#include <tinyxml2.h>`

Inheritance diagram for tinyxml2::XMLElement:



Public Types

- enum { **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) const
- unsigned UnsignedAttribute (const char *name) const

  See IntAttribute()
- bool BoolAttribute (const char *name) const

  See IntAttribute()
- double DoubleAttribute (const char *name) const

  See IntAttribute()
- float FloatAttribute (const char *name) const

See IntAttribute()

- XMLError QueryIntAttribute (const char *name, int *value) const
- XMLError QueryUnsignedAttribute (const char *name, unsigned int *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()
- int QueryAttribute (const char *name, int *value) const
- int **QueryAttribute** (const char *name, unsigned int *value) const
- int **QueryAttribute** (const char *name, bool *value) const
- int **QueryAttribute** (const char *name, double *value) const
- int **QueryAttribute** (const char *name, float *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, 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 (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 QueryBoolText (bool *bval) const

  See QueryIntText()
- XMLError QueryDoubleText (double *dval) const

  See QueryIntText()

- XMLError QueryFloatText (float *fval) const

  See QueryIntText()
- int **ClosingType** () const
- virtual XMLNode * ShallowClone (XMLDocument *document) const
- virtual bool ShallowEqual (const XMLNode *compare) const

Protected Member Functions

- char * **ParseDeep** (char *p, StrPair *endTag)

Friends

- class **XMLDocument**

Additional Inherited Members

6.21.1 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.

6.21.2 Member Function Documentation

6.21.2.1 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.

6.21.2.2 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();
}
```

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

Delete an attribute.

6.21.2.4 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 ".

6.21.2.5 int tinyxml2::XMLElement::IntAttribute (const char * *name*) const [inline]

Given an attribute name, IntAttribute() returns the value of the attribute interpreted as an integer. 0 will be returned if there is an error. For a method with error checking, see QueryIntAttribute()

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

Given an attribute name, QueryAttribute() returns XML_NO_ERROR, 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
```

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

Given an attribute name, QueryIntAttribute() returns XML_NO_ERROR, 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
```

6.21.2.8 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.

6.21.2.9 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>
```

6.21.2.10 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.

6.21.2.11 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.

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

- Libs/tinyxml2.h
- Libs/tinyxml2.cpp

6.22 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.

6.22.1 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.

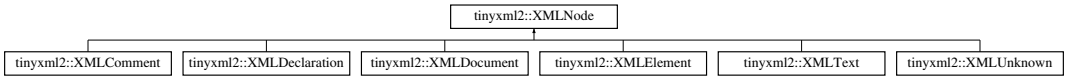
The documentation for this class was generated from the following file:

- Libs/tinyxml2.h

6.23 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)
- 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
- virtual bool ShallowEqual (const XMLNode *compare) const =0
- virtual bool Accept (XMLVisitor *visitor) const =0

Protected Member Functions

- **XMLNode** (XMLDocument *)
- virtual char * **ParseDeep** (char *, StrPair *)

Protected Attributes

- XMLDocument * **_document**
- XMLNode * **_parent**
- StrPair **_value**
- XMLNode * **_firstChild**
- XMLNode * **_lastChild**
- XMLNode * **_prev**
- XMLNode * **_next**

Friends

- class **XMLDocument**
- class **XMLElement**

6.23.1 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)
```

6.23.2 Member Function Documentation

6.23.2.1 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::XMLDocument, tinyxml2::XMLElement, tinyxml2::XMLUnknown, tinyxml2::XML↩Declaration, tinyxml2::XMLComment, and tinyxml2::XMLText.

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

Delete a child of this node.

6.23.2.3 void tinyxml2::XMLNode::DeleteChildren ( )

Delete all the children of this node.

6.23.2.4 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.

6.23.2.5 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.

6.23.2.6 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.

6.23.2.7 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.

6.23.2.8 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.

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

Set the Value of an XML node.

See also

Value()

6.23.2.10 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::XMLElement, tinyxml2::XMLUnknown, tinyxml2::XMLDeclaration, tinyxml2::XMLComment, and tinyxml2::XMLText.

6.23.2.11 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::XMLElement, tinyxml2::XMLUnknown, tinyxml2::XMLDeclaration, tinyxml2::XMLComment, and tinyxml2::XMLText.

6.23.2.12 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
```

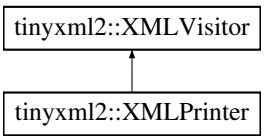
The documentation for this class was generated from the following files:

- Libs/tinyxml2.h
- Libs/tinyxml2.cpp

6.24 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, 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 (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 ()

Protected Member Functions

- virtual bool **CompactMode** (const XMLElement &)
- virtual void PrintSpace (int depth)
- void **Print** (const char *format,...)
- void **SealElementIfJustOpened** ()

Protected Attributes

- bool **_elementJustOpened**
- DynArray< const char *, 10 > **_stack**

6.24.1 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();
```

6.24.2 Constructor & Destructor Documentation

6.24.2.1 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.

6.24.3 Member Function Documentation

6.24.3.1 void tinyxml2::XMLPrinter::ClearBuffer ( ) [inline]

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

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

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

6.24.3.3 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.)

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

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

6.24.3.5 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().

6.24.3.6 void tinyxml2::XMLPrinter::PushHeader (bool *writeBOM,* bool *writeDeclaration*)

If streaming, write the BOM and declaration.

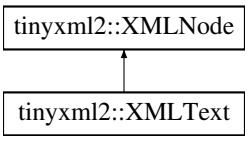
The documentation for this class was generated from the following files:

- Libs/tinyxml2.h
- Libs/tinyxml2.cpp

6.25 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 ∗, StrPair ∗endTag)

Friends

- class **XMLDocument**

Additional Inherited Members

6.25.1 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().

6.25.2 Member Function Documentation

6.25.2.1 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.

6.25.2.2 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.

6.25.2.3 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.

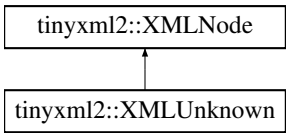
The documentation for this class was generated from the following files:

- Libs/tinyxml2.h
- Libs/tinyxml2.cpp

6.26 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 *, StrPair *endTag)

Friends

- class **XMLDocument**

Additional Inherited Members

6.26.1 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.

6.26.2 Member Function Documentation

6.26.2.1 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.

6.26.2.2 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.

6.26.2.3 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.

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

- Libs/tinyxml2.h
- Libs/tinyxml2.cpp

6.27 tinyxml2::XMLUtil Class Reference

Static Public Member Functions

- static const char * **SkipWhiteSpace** (const char *p)
- static char * **SkipWhiteSpace** (char *p)
- static bool **IsWhiteSpace** (char p)
- static bool **IsNameStartChar** (unsigned char ch)
- static bool **IsNameChar** (unsigned char ch)
- static bool **StringEqual** (const char *p, const char *q, int nChar=INT_MAX)
- static bool **IsUTF8Continuation** (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 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)

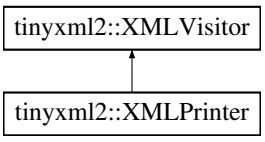
The documentation for this class was generated from the following files:

- Libs/tinyxml2.h
- Libs/tinyxml2.cpp

6.28 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.

6.28.1 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()

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

- Libs/tinyxml2.h

Index