LibMusicXML Reference Manual

0.92

Generated by Doxygen 1.3.3

Tue Mar 23 09:49:43 2004

Contents

Chapter 1

LibMusicXML Overview

1.1 Introduction

The MusicXML library provides a set of classes that covers the elements defined by the MusicXML 0.8 dtds. The library has been developped in C++, with a great care of preserving platform independence.

Connection between the classes and the xml elements is not a one-to-one relation: for simplification, a class may contain several MusicXML elements, provided that these elements are not reused anywhere else. For example, MusicXML defines `dynamics` as separate elements (see common.dtd): using a single object to represent them clarifies the representation. Apart a few exceptions detailled below, all of the MusicXML elements have their counterpart in the library representation.

A good knowledge of MusicXML is required for a good comprehension of the library.

1.2 Conventions

1.2.1 Memory management

Each class that describes a MusicXML element is handled using ***smart pointers*** . A smart pointer maintains a reference count for an object and takes in charge the automatic deletion of this object when its reference count drops to zero.

Objects that are handled using smart pointers derive from the `smartable` class. Their implementation prevents direct call to the constructor. Instead of constructor call, they provide a function to create a new object directly embedded within a smart pointer.

Smart pointers don't work when a loop in the reference occurs. However, it doesn't matter in our case since MusicXML is strictly hierarchical.

1.2.2 Class and type names.

All the classes that corresponds to a MusicXML element have a name in the form of `Txxx` where `xxx` relates to the MusicXML element name.

For each `smartable` class, a typedef defines the corresponding smart pointer as follow:

`typedef Txxx Sxxx` where `xxx` is the class name.

More generally:

- a type name that starts with **T** denotes a class name,
- a type name that starts with **S** denotes a class smart pointer.

1.2.3 Constructors.

As mentioned above, *smartable* classes prevent direct call to their constructor but a *new* function is provided which conforms to the following rule:

for a class named `Txxx` where `xxx` is the class name, the *new* function name is `newxxx` .

A *new* function interface is similar to the class constructor. Each MusicXML object provides a unique constructor. Arguments of a class constructor are limited to the **required** MusicXML element attributes.

1.2.4 Fields access.

Fields of the defined objects are generally private or at least protected. Semantic of a field may be deduced from the field name and from the object corresponding MusicXML description, which is provided as the class documentation. Access methods are designed according to the following rules:

- methods starting with `set` are write only methods,
- methods starting with `get` are read only methods,
- methods that designates a field name (like `transpose()` to access a `fTranspose` field) are read/write methods,
- `add` methods are polymorphic methods used to push new elements into internal lists.

Warning:
Read/write methods are always used with embedded smart pointers. These pointers are initialized by default and therefore points to *null*. This design is intended to denote undefined elements.

1.2.5 Files organization.

The files organization is similar to the MusicXML dtd files organization: for example, most of the elements or entities defined in common.dtd are handled by **common.h**(p. **??**). Elements described within MusicXML attributes.dtd or direction.dtd are respectively located in **TAttributes.h**(p. **??**) or **TDirection.h**(p. **??**). When no correspondence exists between a header file and a MusicXML dtd, the header file name relates to a MusicXML element and declares this element and possibly other related elements. Except for **common.h**(p. **??**), all the files concerned by the above rules have a name starting with **T**.

1.3 Common entities representation

MusicXML defines entities that are common across multiple component DTDs. Many of them are intended to describe graphic layout like position, placement or orientation.

```
<!ENTITY % position
"default-x     %tenths;    #IMPLIED
 default-y     %tenths;    #IMPLIED
 relative-x    %tenths;    #IMPLIED
 relative-y    %tenths;    #IMPLIED">

<!ENTITY % placement
"placement (above | below) #IMPLIED">

<!ENTITY % orientation
"orientation (over | under) #IMPLIED">
```

These entities are defined as separate objects. A class is defined to aggregate the corresponding properties to objects that require them. For example, a `TOrientation` class is defined to describe the `orientation` entity; next an `Orientable` class is intended to provide the corresponding properties to derived classes. MusicXML elements that carry the orientation entity have to derive the `Orientable` class.

MusicXML also makes use of entities to enumerate values like the `orientation` defined above or the `start-stop` or `yes-no` examples below.

```
<!ENTITY % start-stop "(start | stop)">

<!ENTITY % yes-no "(yes | no)">
```

For all these entities, the library provides conversion classes (like the `YesNo` class) that:

- define a type for the corresponding enumeration
- provides conversion methods from/to integer and textual representation. These methods are generally named *xml* and overloaded to access both the integer and string representation.

```
class EXP YesNo {
    public:
    enum type { undefined, yes, no, last=no };

    static const string xml (type d);
    static       type xml (const string str);
};
```

1.4 Common types definition

Many of the MusicXML elements are defined as an alternative bewteen a set of elements. These alternative may be viewed as an inheritance relationship. Each time it has been convenient to do so, this relationship has been made explicit using a specific type, defined to cover the corresponding elements:

- `TMusicData` is an abstract class which purpose is to define a common type for all the elements covered by the `music-data` entity. It also provides the *smartability* and *visitability* properties to the derived objects.

```
class TMusicData : public virtual visitable, public virtual smartable {
     protected:
        TMusicData () {}
        virtual ~TMusicData() {}
};
```

Similarly:

- `TDirectionTypeElement` is an abstract class which purpose is to define a common type for all the elements of a `direction-type` element, `-TNotationElement` is an abstract class which purpose is to define a common type for all the elements of a `notation` element,
- `TPartListElement` is an abstract class which purpose is to define a common type for all the elements of a `part-list` element.

1.5 Browsing the music tree

The music representation is a tree which root is a *timewise* or *partwise* score. All the elements of the score support the *visitor* design pattern and accept a `TScoreVisitor` as the base class of the visitor design.

To preserve the choice of different strategies for the traversing the music representation structure, traversing has not been implemented in the tree components: it is assumed that it's the visitor responsability. A visitor that implements a basic traversing of the music structure is included in the library: the `TRoutedVisitor`.

A `TRoutedVisitor` provides a path for traversing of a score tree. Each implemented *visite* method calls *accept()* for its subclasses. Order of these calls is the following:

- subclasses representing attributes are called first,
- subclasses representing elements are called in the MusicXML defined order.

1.6 Reading and writing MusicXML files

The `TMusicXMLFile` object provides a simple API to read and write MusicXML files.

```
class EXP TMusicXMLFile {
    public:
                TMusicXMLFile() {}
        virtual ~TMusicXMLFile() {}

        SScore read (const string& file);

        bool write (SScore score, const string& file);
        bool write (SScore score, ostream& os);
};
```

It provides a `read` method that takes a file name as argument and returns smart pointer to a music score as result. The returned pointer is null in case of failure.

It provides `write` methods that take a music score and a file name or an `ostream` as arguments and returns a boolean value as result. The returned value is false in case of failure.

1.7 MusicXML Library specific elements

The library design is very close to the MusicXML definition. A music score memory representation tree matches the corresponding MusicXML tree with just a few exceptions:

- there is no chord element in the MusicXML specification but a chord object is part of the MusicXML library.
- not all the MusicXML elements are represented as individual objects: it has often been more convenient to group a collection of similar elements into a single object. The TArticulation object is an example of such a grouping.

1.8 MusicXML Elements to be implemented

The main part of the MusicXML elements is already covered by the current library. A small set of elements and attributes are however not yet supported. Below is the list of the to be implemented elements.

From attributes.dtd:

- staff-details
- measure-style

From direction.dtd:

- rehearsal
- pedal
- damp
- damp-all
- eyeglasses
- other-direction
- harmony
- grouping
- print

From identity.dtd:

- miscellaneous

From note.dtd:

- glissando
- slide
- technical
- arpeggiate
- non-arpeggiate
- other-notation
- other-articulation

- lyric
- figured-bass

From link.dtd:

- link
- bookmark

From opus.dtd:

- opus (opus.dtd)

Some common attributes are also ignored by the current implementation. They are represented by entities in common.dtd.

- editorial
- font
- printout
- trill-sound
- bend-sound

Chapter 2

LibMusicXML Module Index

2.1 LibMusicXML Modules

Here is a list of all modules:

Chapter 3

LibMusicXML Hierarchical Index

3.1 LibMusicXML Class Hierarchy

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

Chapter 4

LibMusicXML Compound Index

4.1 LibMusicXML Compound List

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

Chapter 5

LibMusicXML Page Index

5.1 LibMusicXML Related Pages

Here is a list of all related documentation pages:

Chapter 6

LibMusicXML Module Documentation

6.1 Guido

Compounds

- class **guidochord**
  The guido chord element.
- class **guidoelement**
  A generic guido element representation.
- class **guidonote**
  A guido note representation.
- class **guidonoteduration**
  A guido note duration representation.
- class **guidonotestatus**
  Represents the current status of notes duration and octave.
- class **guidoparam**
  A guidotag parameter representation.
- class **guidoseq**
  The guido sequence element.
- class **guidotag**
  A guido tag representation.
- class **guidovisitable**
  interface for visitable guido objects.
- class **guidovisitor**

an abstract generic guido visitor

- class **TXML2GuidoVisitor**

  A score visitor to produce a generic Guido representation.

Typedefs

- typedef **SMARTP**< **guidonote** > **Sguidonote**
- typedef **SMARTP**< **guidoseq** > **Sguidoseq**
- typedef **SMARTP**< **guidochord** > **Sguidochord**
- typedef **SMARTP**< **guidotag** > **Sguidotag**

6.2 Visitors

Compounds

- class **TNodeVisitor**
  A node Visitor of MusicXML scores.
- class **TPartSummary**
  summary of a MusicXML part.
- class **TRolledVisitor**
  A visitor that transport a **TNodeVisitor**(p. 90) *along a score.*
- class **TRoutedVisitor**
  A visitor that implements a traversing of a score.
- class **TScanVisitor**
  A visitor that only scans a score to gather information.
- class **TScoreVisitor**
  A Visitor of MusicXML scores.
- class **TXML2GuidoVisitor**
  A score visitor to produce a generic Guido representation.
- class **visitable**
  interface for visitable objects.
- class **vvector**
  a visitable vector

Typedefs

- typedef **SMARTP**< TNodeVisitor > **SNodeVisitor**
- typedef **SMARTP**< TRolledVisitor > **SRolledVisitor**
- typedef **SMARTP**< TRoutedVisitor > **SRoutedVisitor**
- typedef **SMARTP**< **TPartSummary** > **SPartSummary**
- typedef **SMARTP**< **TScanVisitor** > **SScanVisitor**
- typedef **SMARTP**< visitable > **Svisitable**

Chapter 7

LibMusicXML Class Documentation

7.1 bimap Class Template Reference

implements a bijective map

`#include <bimap.h>`

Public Member Functions

- **bimap** (const T1 tbl1[], const T2 tbl2[], int n)
- const T2 **operator[]** (const T1 key)

  returns the second type value indexed by the first type

- const T1 **operator[]** (const T2 key)

  returns the first type value indexed by the second type

- long **size** ()

  returns the map size

- **bimap** & **add** (const T1 &v1, const T2 &v2)

  adds a pair of values

7.1.1 Detailed Description

template<typename T1, typename T2> class bimap< T1, T2 >

A bijective map is a map where values and keys are interchangeable. To operate correctly, typenames T1 and T2 must be different.

7.2 FullCue Class Reference

provides conversions between numeric note size types and strings

`#include <conversions.h>`

Public Types

- enum **type** { **undefined**, **full**, **cue**, **last** = cue }

Static Public Member Functions

- const string **xml** (type d)

  convert a numeric size value to a MusicXML string

- type **xml** (const string str)

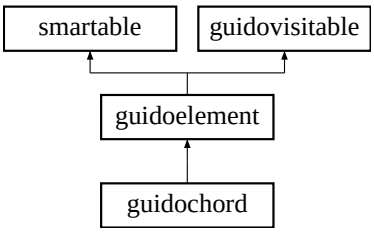
  convert a MusicXML string to a numeric size value

7.3 guidochord Class Reference

The guido chord element.

`#include <guido.h>`

Inheritance diagram for guidochord::



Public Member Functions

- void **accept** (**guidovisitor** &visitor)

Friends

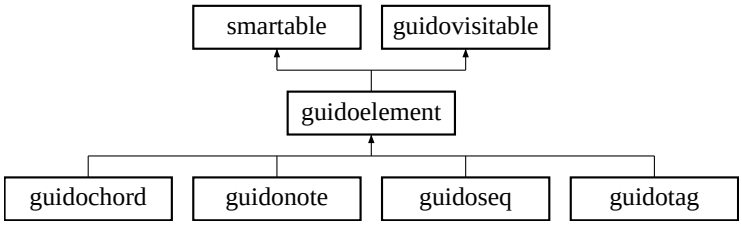
- VEXP friend **SMARTP**< **guidochord** > **new_guidochord** ()

7.4 guidoelement Class Reference

A generic guido element representation.

`#include <guido.h>`

Inheritance diagram for guidoelement::



Public Member Functions

- long **add** (**Sguidoelement** &elt)
- long **add** (**Sguidoparam** &param)
- long **add** (**Sguidoparam** param)
- void **print** (ostream &os)
- void **setName** (string name)

  the element name

- string **getName** () const
- string **getStart** () const
- string **getEnd** () const
- string **getSep** () const
- const vector< **Sguidoelement** > & **elements** () const
- const vector< **Sguidoparam** > & **parameters** () const
- bool **empty** () const
- virtual void **accept** (**guidovisitor** &visitor)

Protected Member Functions

- **guidoelement** (string name, string sep="")

Protected Attributes

- string **fName**
- string **fStartList**

  the contained element start marker (default to empty)

- string **fEndList**

  the contained element end marker (default to empty)

- string **fSep**

  the element separator (default to space)

- vector< **Sguidoelement** > **fElements**
  list of the enclosed elements

- vector< **Sguidoparam** > **fParams**
  list of optional parameters

Friends

- VEXP friend **SMARTP**< **guidoelement** > **new_guidoelement** (string name, string sep="")

7.4.1 Detailed Description

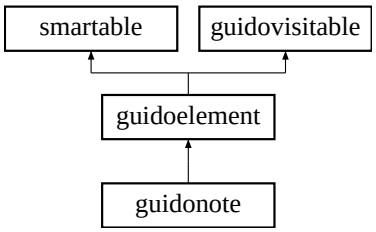
An element is represented by its name and the list of its enclosed elements plus optional parameters.

7.5 guidonote Class Reference

A guido note representation.

`#include <guido.h>`

Inheritance diagram for guidonote::



Public Member Functions

- void **set** (unsigned short voice, string name, char octave, **guidonoteduration** &dur, string acc)
- void **accept** (**guidovisitor** &visitor)
- const char * **name** () const
- const char * **accidental** () const
- char **octave** () const
- const **guidonoteduration** & **duration** () const

Protected Member Functions

- **guidonote** (unsigned short voice, string name, char octave, **guidonoteduration** &dur, string acc="")

Protected Attributes

- string **fNote**
- string **fAccidental**
- char **fOctave**
- **guidonoteduration fDuration**

Friends

- VEXP friend **SMARTP**< **guidonote** > **new_guidonote** (unsigned short voice, string name, char octave, **guidonoteduration** &dur, string acc="")

7.5.1 Detailed Description

A note is represented by its name, optional accidentals, duration (in the form of numerator/denominator) and optional dots.

7.6 guidonoteduration Class Reference

A guido note duration representation.

`#include <guido.h>`

Public Member Functions

- **guidonoteduration** (long num, long denom, long dots=0)
- void **set** (long num, long denom, long dots=0)
- **guidonoteduration** & **operator=** (const **guidonoteduration** &dur)
- bool **operator!=** (const **guidonoteduration** &dur) const

Public Attributes

- long **fNum**
- long **fDenom**
- long **fDots**

7.6.1 Detailed Description

A note duration is represented by a numerator (denotes the number of beats), a denominator (denotes the beat value) and optional dots. Triplets are repesented as 1/3, 1/6, ... quintuplets, septuplets and so on are handled analogously.

7.7 guidonotestatus Class Reference

Represents the current status of notes duration and octave.

`#include <guido.h>`

Public Types

- enum { **kMaxInstances** = 128 }
- enum { **defoctave** = 1, **defnum** = 1, **defdenom** = 4 }

Public Member Functions

- void **reset** ()
- **guidonotestatus** & **operator=** (const **guidonoteduration** &dur)
- bool **operator!=** (const **guidonoteduration** &dur) const

Static Public Member Functions

- **guidonotestatus** ∗ **get** (unsigned short voice)
- void **resetall** ()

Public Attributes

- char **fOctave**
- **guidonoteduration fDur**

7.7.1 Detailed Description

Octave and duration may be ommitted for guido notes. If so, they are infered from preceeding notes (or rest), within the same sequence or chord, or assumed to have standard values.

The object is defined as a multi-voices singleton: a single object is allocated for a specific voice and thus it will not operate correctly on a same voice parrallel formatting operations.

Todo

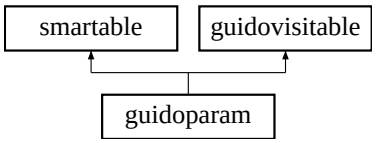
handling the current beat value for ∗*num* duration form.

7.8 guidoparam Class Reference

A guidotag parameter representation.

`#include <guido.h>`

Inheritance diagram for guidoparam::



Public Member Functions

- void **set** (string value, bool quote=true)

  the parameter value

- void **set** (long value, bool quote=true)
- string **get** () const
- bool **quote** () const
- void **accept** (**guidovisitor** &visitor)

Protected Member Functions

- **guidoparam** (string value, bool quote)
- **guidoparam** (long value, bool quote)

Friends

- VEXP friend **SMARTP**< **guidoparam** > **new_guidoparam** (string value, bool quote=true)
- VEXP friend **SMARTP**< **guidoparam** > **new_guidoparam** (long value, bool quote=true)

7.8.1 Detailed Description

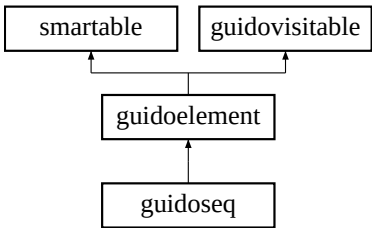
A parameter is represented by its value.

7.9 guidoseq Class Reference

The guido sequence element.

`#include <guido.h>`

Inheritance diagram for guidoseq::



Public Member Functions

- void **accept** (**guidovisitor** &visitor)

Friends

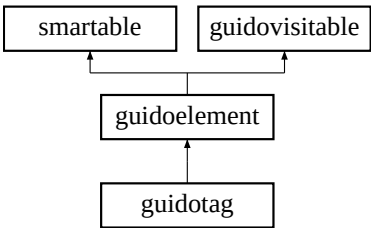
- VEXP friend **SMARTP**< **guidoseq** > **new_guidoseq** ()

7.10 guidotag Class Reference

A guido tag representation.

`#include <guido.h>`

Inheritance diagram for guidotag::



Public Member Functions

- void **accept** (**guidovisitor** &visitor)

Protected Member Functions

- **guidotag** (string name)

Friends

- VEXP friend **SMARTP**< **guidotag** > **new_guidotag** (string name)

7.10.1 Detailed Description

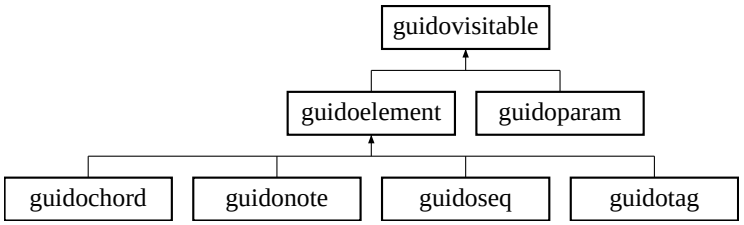
A tag is represented by its name and optional parameters. A range tag contains enclosed elements.

7.11 guidovisitable Class Reference

interface for visitable guido objects.

`#include <guidovisitor.h>`

Inheritance diagram for guidovisitable::



Public Member Functions

- virtual void **accept** (**guidovisitor** &visitor)=0

7.12 guidovisitor Class Reference

an abstract generic guido visitor

`#include <guidovisitor.h>`

Public Member Functions

- virtual void **visite** (**Sguidoelement** &elt)=0
- virtual void **visite** (**Sguidonote** &elt)=0
- virtual void **visite** (**Sguidoseq** &elt)=0
- virtual void **visite** (**Sguidochord** &elt)=0
- virtual void **visite** (**Sguidotag** &elt)=0
- virtual void **visite** (**Sguidoparam** &elt)=0

7.13 LineType Class Reference

provides conversions between numeric line types and strings

`#include <conversions.h>`

Public Types

- enum **type** {
  undefined, **solid**, **dashed**, **dotted**,
  wavy, **last** = wavy }

Static Public Member Functions

- const string **xml** (type d)
  convert a numeric start-stop value to a MusicXML string
- type **xml** (const string str)
  convert a MusicXML string to a numeric start-stop value

7.14 NoteType Class Reference

provides conversions between numeric note types and strings

`#include <conversions.h>`

Public Types

- enum **type** {
  undefined, **t256th** = 1, **t128th** = 1<<1, **t64th** = 1<<2,
  t32nd = 1<<3, **t16th** = 1<<4, **eighth** = 1<<5, **quarter** = 1<<6,
  half = 1<<7, **whole** = 1<<8, **breve** = 1<<9, **tlong** = 1<<10,
  count = 11 }

Static Public Member Functions

- **TRational** & **rational** (type d, **TRational** &r)
  convert an integer note to a rational representation
- const string **xml** (type d)
  convert an integer note type to a MusicXML string
- type **xml** (const string str)
  convert an MusicXML string to an integer note type

7.14.1 Detailed Description

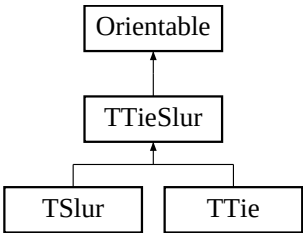
Type indicates the graphic note type. Valid values (from shortest to longest) are 256th, 128th, 64th, 32nd, 16th, eighth, quarter, half, whole, breve, and long.

7.15 Orientable Class Reference

base class for all elements that have an orientation.

`#include <common.h>`

Inheritance diagram for Orientable::



Public Member Functions

- **TOrientation** & **orientation** ()

7.16 pairmap Class Template Reference

implements a multimap where <key, value> pairs are unique

`#include <bimap.h>`

Public Types

- typedef **pairmap**< T1, T2 >::**iterator iterator**
- typedef **pairmap**< T1, T2 >::**const_iterator const_iterator**

Public Member Functions

- **iterator insert** (const pair< T1, T2 > &x)
- **iterator insert** (**iterator** position, const pair< T1, T2 > &x)

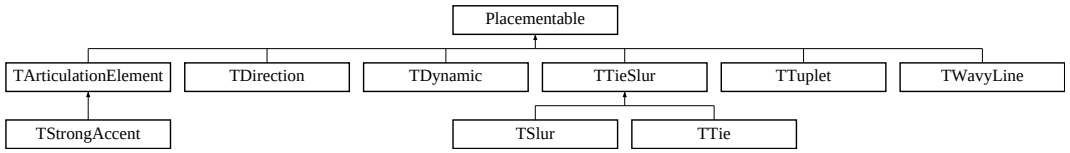
template<typename T1, typename T2> class pairmap< T1, T2 >

7.17 Placementable Class Reference

base class for all elements that have a placement.

`#include <common.h>`

Inheritance diagram for Placementable::



Public Member Functions

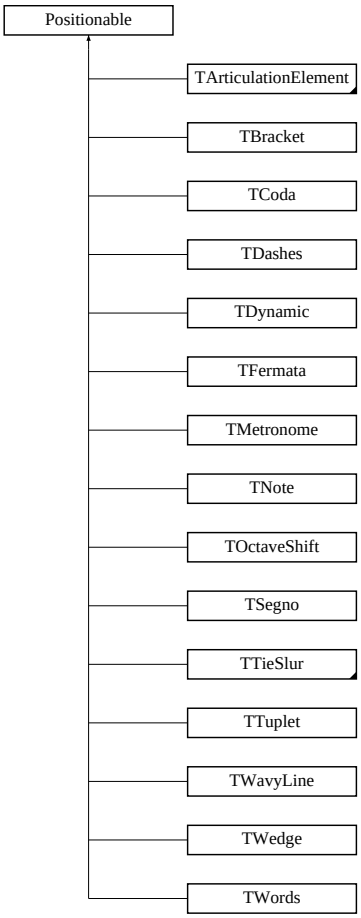
- **TPlacement** & **placement** ()

7.18 Positionable Class Reference

base class for all elements that have a position.

`#include <common.h>`

Inheritance diagram for Positionable::



Public Member Functions

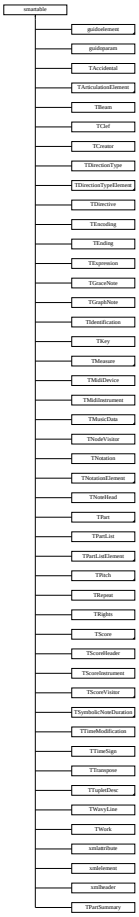
- **TPosition** & **position** ()

7.19 smartable Class Reference

the base class for smart pointers implementation

`#include <smartpointer.h>`

Inheritance diagram for smartable::



Public Member Functions

- unsigned **refs** () const
  gives the reference count of the object
- void **addReference** ()
  addReference increments the ref count and checks for refCount overflow
- void **removeReference** ()
  removeReference delete the object when refCount is zero

Protected Member Functions

- **smartable** (const **smartable** &)

- virtual ~**smartable** ()

  destructor checks for non-zero refCount

- **smartable** & **operator=** (const **smartable** &)

7.19.1 Detailed Description

Any object that want to support smart pointers should inherit from the smartable class which provides reference counting and automatic delete when the reference count drops to zero.

7.20 SMARTP Class Template Reference

the smart pointer implementation

`#include <smartpointer.h>`

Public Member Functions

- **SMARTP** ()

  an empty constructor - points to null

- **SMARTP** (T *rawptr)

  build a smart pointer from a class pointer

- template<class T2> **SMARTP** (const **SMARTP**< T2 > &ptr)

  build a smart pointer from an convertible class reference

- **SMARTP** (const **SMARTP** &ptr)

  build a smart pointer from another smart pointer reference

- ~**SMARTP** ()

  the smart pointer destructor: simply removes one reference count

- **operator T** * () const

  cast operator to retrieve the actual class pointer

- T & **operator** * () const

  '' operator to access the actual class pointer*

- T * **operator** → () const

  operator -> overloading to access the actual class pointer

- template<class T2> **SMARTP** & **operator=** (T2 p1_)

  operator = that moves the actual class pointer

- **SMARTP** & **operator=** (T *p_)

  operator = that moves the actual class pointer

- **SMARTP** & **operator=** (const **SMARTP**< T > &p_)

  operator = to support inherited class reference

- template<class T2> **SMARTP** & **cast** (T2 *p_)

  dynamic cast support

- template<class T2> **SMARTP** & **cast** (const **SMARTP**< T2 > &p_)

  dynamic cast support

7.20.1 Detailed Description

template<class T> class SMARTP< T >

A smart pointer is in charge of maintaining the objects reference count by the way of pointers operators overloading. It supports class inheritance and conversion whenever possible.

Instances of the SMARTP class are supposed to use *smartable* types (or at least objects that implements the *addReference* and *removeReference* methods in a consistent way).

7.21 StartStop Class Reference

provides conversions between numeric start-stop types and strings

`#include <conversions.h>`

Public Types

- enum **type** {
  undefined, **start**, **stop**, **cont**,
  last = cont }

Static Public Member Functions

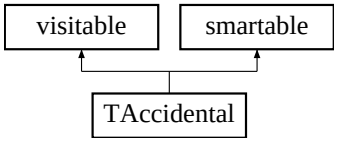
- const string **xml** (type d)
  convert a numeric start-stop value to a MusicXML string
- type **xml** (const string str)
  convert a MusicXML string to a numeric start-stop value

7.22 TAccidental Class Reference

Representation of an accidental.

`#include <TNote.h>`

Inheritance diagram for TAccidental::



Public Types

- enum **accident** {
  undefined = -1, **sharp** = 1, **natural**, **flat**,
  double_sharp, **sharp_sharp**, **flat_flat**, **natural_sharp**,
  natural_flat, **quarter_flat**, **quarter_sharp**, **three_quarters_flat**,
  three_quarters_sharp, **last** = three_quarters_sharp }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- accident **getAccident** () const
- YesNo::type **getCautionary** () const
- YesNo::type **getEditorial** () const
- FullCue::type **getSize** () const
- void **setAccident** (accident acc)
- void **setCautionary** (YesNo::type yn)
- void **setEditorial** (YesNo::type yn)
- void **setSize** (FullCue::type size)

Static Public Member Functions

- const string **xmlaccidental** (accident d)

  convert a numeric accidental type to a MusicXML string

- accident **xmlaccidental** (const string str)

  convert an MusicXML string to a numeric accidental type

Friends

- EXP friend **SMARTP**< **TAccidental** > **newAccidental** ()

7.22.1 Detailed Description

Actual notated accidentals. Valid values include: sharp, natural, flat, double-sharp, sharp-sharp, flat-flat, natural-sharp, natural-flat, quarter-flat, quarter-sharp, three-quarters-flat, and three-quarters-sharp. Editorial and cautionary indications, are indicated by attributes. Values for these attributes are "no" if not present.

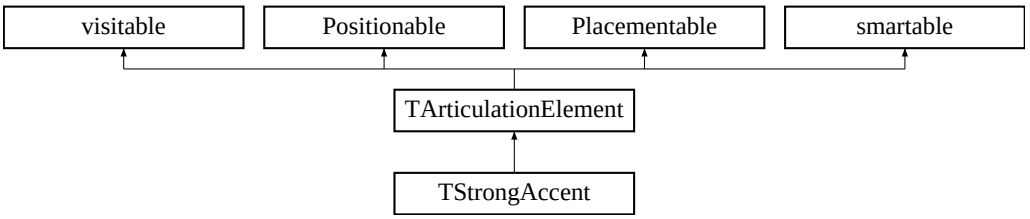
The MusicXML *accidental* element is defined in note.dtd.

7.23 TArticulationElement Class Reference

articulations and accents definition.

`#include <TNotation.h>`

Inheritance diagram for TArticulationElement::



Public Types

- enum **articulation** {
  first = 1, **accent** = first, **strong_accent**, **staccato**,
  tenuto, **detached_legato**, **staccatissimo**, **spiccato**,
  scoop, **plop**, **doit**, **falloff**,
  breath_mark, **caesura**, **last** = caesura }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- articulation **getArtType** () const
- void **setArtType** (articulation a)

Static Public Member Functions

- const string **xmlname** (articulation d)
  convert a numeric articulation name to a MusicXML string
- articulation **xmlname** (const string str)
  convert an MusicXML string to a numeric articulation name
- const string * **getArticulationStrings** ()

Protected Member Functions

- **TArticulationElement** (articulation type)

Friends

- EXP friend **SMARTP**< **TArticulationElement** > **newArticulationElement** (articulation type)

7.23.1 Detailed Description

Articulations elements are all handled by a single object with a type field used to specify the articulation type. All the articulations have a position and placement attributes. Apart the *other-articulation* element, all are empty elements.

TArticulation handles a set of elements defined in note.dtd.

Note:
the *other-articulation* element is not implemented.

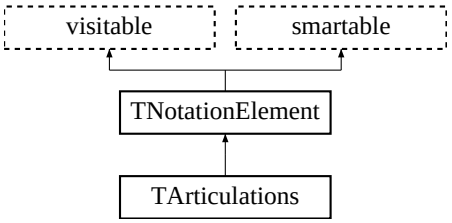
Todo
support of the line-shape and line-type attributes

7.24 TArticulations Class Reference

The MusicXML *articulations* element.

`#include <TNotation.h>`

Inheritance diagram for TArticulations::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **SArticulationElement** &art)
- **vvector**< **SArticulationElement** > & **articulations** ()

Friends

- EXP friend **SMARTP**< **TArticulations** > **newArticulations** ()

7.24.1 Detailed Description

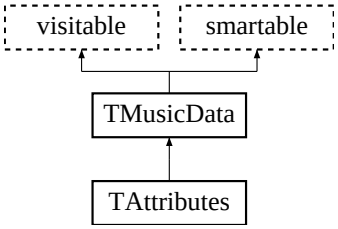
Defined in note.dtd.

7.25 TAttributes Class Reference

Contains musical information that typically changes on measure boundaries.

`#include <TAttributes.h>`

Inheritance diagram for TAttributes::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setDivisions** (long div)
- void **setStaves** (long staves)
- void **setInstrument** (long inst)
- long **getDivision** () const
- long **getStaves** () const
- long **getInstrument** () const
- **SKey** & **key** ()
- **STimeSign** & **timeSign** ()
- **STranspose** & **transpose** ()
- **SDirective** & **directive** ()
- **vvector**< **SClef** > & **clefs** ()
- long **add** (const **SClef** &clef)

Friends

- EXP friend **SMARTP**< **TAttributes** > **newAttributes** ()

7.25.1 Detailed Description

It includes key and time signatures, clefs, transpositions, and staving. **TAttributes**(p. 52) corresponds to the MusicXML *attribute* element as defined in attributes.dtd.

Note:
MusicXML *editorial* information is ignored.

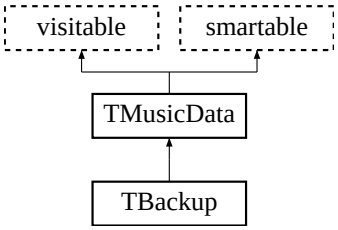
Todo
clarify the Instrument type and semantic
support of the *staff-details* and *measure-style* elements

7.26 TBackup Class Reference

Represents the MusicXML backup element.

`#include <TBackupForward.h>`

Inheritance diagram for TBackup::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setDuration** (long dur)

  Duration is a required element and is set to zero by the constructor.

- long **getDuration** () const

Friends

- EXP friend **SMARTP**< **TBackup** > **newBackup** ()

7.26.1 Detailed Description

Backup and forward are required to coordinate multiple voices in one part, including music on a grand staff. Backup is generally used to move between voices and staves. Thus the backup element does not include voice or staff elements. Duration values should always be positive, and should not cross measure boundaries.

The MusicXML *backup* element is defined in note.dtd.

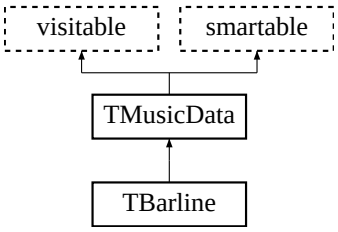
Note:
Editorial information is ignored.

7.27 TBarline Class Reference

Represents a MusicXML barline element.

`#include <TBarline.h>`

Inheritance diagram for TBarline::



Public Types

- enum {
  undefined = 0, **right** = 1, **left**, **middle**,
  lastloc = middle }
- enum {
  none = 1, **regular**, **dotted**, **heavy**,
  light_light, **light_heavy**, **heavy_light**, **heavy_heavy**,
  last = heavy_heavy }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setLocation** (int loc)
- void **setBarStyle** (int style)
- int **getLocation** () const
- int **getBarStyle** () const
- **SWavyLine** & **wavyLine** ()
- **SSegno** & **segno** ()
- **SCoda** & **coda** ()
- **SFermata** & **fermata** ()
- **SEnding** & **ending** ()
- **SRepeat** & **repeat** ()

Static Public Member Functions

- const string **xmllocation** (int d)
  convert an integer location to a MusicXML string
- int **xmllocation** (const string str)
  convert an MusicXML string to an integer location

- const string **xmlstyle** (int d)

  convert an integer bar style to a MusicXML string

- int **xmlstyle** (const string str)

  convert an MusicXML string to an integer bar style

Friends

- EXP friend **SMARTP**< **TBarline** > **newBarline** ()

7.27.1 Detailed Description

If a barline is other than a normal single barline, it should be represented by a barline element that describes it. This includes information about repeats and multiple endings, as well as line style. The two fermata elements allow for fermatas on both sides of the barline (the lower one inverted).

Barlines have a location attribute.It must match where the barline element occurs within the rest of the musical data in the score. If location is left, it should be the first element in the measure; if location is right, it should be the last element. If no location is specified, the right barline is the default.

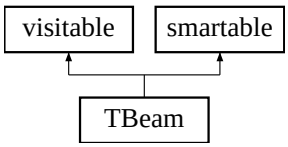
The MusicXML *barline* element is defined in barline.dtd.

7.28 TBeam Class Reference

beaming representation.

`#include <TNote.h>`

Inheritance diagram for TBeam::



Public Types

- enum **beamtype** {
  undefined = -1, **beambegin** = 1, **beamcontinue**, **beamend**,
  beamforward, **beambackward**, **last** = beambackward }

Public Member Functions

- beamtype **getType** () const
- int **getNumber** () const
- YesNo::type **getRepeater** () const
- void **setType** (beamtype type)
- void **setNumber** (int num)
- void **setRepeater** (YesNo::type r)
- virtual void **accept** (**TScoreVisitor** &visitor)

Static Public Member Functions

- const string **xmlbeamtype** (beamtype d)
  convert a numeric beam type to a MusicXML string
- beamtype **xmlbeamtype** (const string str)
  convert an MusicXML string to a numeric beam type

Friends

- EXP friend **SMARTP**< **TBeam** > **newBeam** ()

7.28.1 Detailed Description

Beam types include begin, continue, end, forward hook, and backward hook. In MuseData, up to six concurrent beams are available to cover up to 256th notes. This seems sufficient so we use an enumerated type defined in common.dtd. The repeater attribute needs to be specified with a "yes" value for each beam using it.

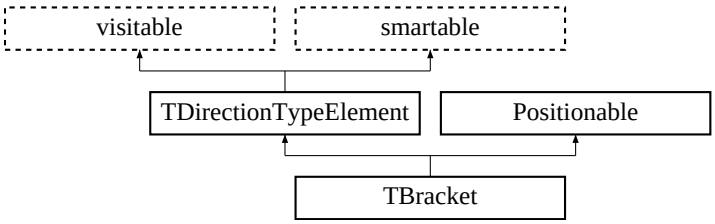
The MusicXML *beam* element as defined in note.dtd.

7.29 TBracket Class Reference

The MusicXML *bracket* element.

`#include <TDirection.h>`

Inheritance diagram for TBracket::



Public Types

- enum {
  undefined = -1, **up** = 1, **down**, **both**,
  arrow, **none**, **last** = none }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setType** (StartStop::type type)
- void **setNumber** (int num)
- void **setLineEnd** (int lend)
- void **setLineType** (LineType::type t)
- StartStop::type **getType** () const
- int **getNumber** () const
- int **getLineEnd** () const
- LineType::type **getLineType** () const

Static Public Member Functions

- const string **xmllineend** (int d)
  convert an numeric line end value to a MusicXML string
- int **xmllineend** (const string str)
  convert an MusicXML string to a numeric line end value

Protected Member Functions

- **TBracket** (StartStop::type t)

Friends

- EXP friend **SMARTP< TBracket > newBracket** (StartStop::type t)

7.29.1 Detailed Description

Brackets are combined with words in a variety of modern directions. The line-end attribute specifies if there is a jog up or down (or both), an arrow, or nothing at the start or end of the bracket. The line-type is solid by default.

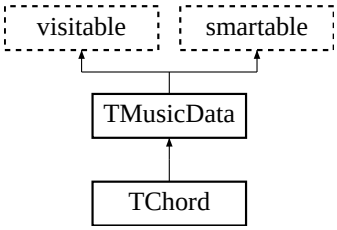
The MusicXML *bracket* element is defined in direction.dtd.

7.30 TChord Class Reference

a notes container that denotes a chord.

`#include <TChord.h>`

Inheritance diagram for TChord::



Public Types

- enum **sorttype** { **pitch**, **duration** }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **SNote** &note)
- **vvector**< **SNote** > & **notes** ()
- unsigned long **getDuration** () const

  returns the chord duration that is the shortest note duration (or 0 for an empty chord)

- long **getVoice** () const

  returns the voice of the chord notes

- long **getStaff** () const

  returns the staff of the first chord note

- TNote::style **getStyle** () const

  returns the style of the chord notes (normal, cue, grace)

- void **sort** (sorttype type)

  sorts the notes of the chord

Friends

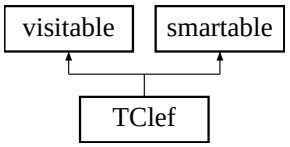
- EXP friend **SMARTP**< **TChord** > **newChord** ()

7.31 TClef Class Reference

Represents a clef.

`#include <TClef.h>`

Inheritance diagram for TClef::



Public Types

- enum {
  undefined = -1, **none** = 1, **G**, **F**,
  C, **percussion**, **TAB**, **last** = TAB }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setStaffNum** (long num)
- void **setSign** (long sign)
- void **setLine** (long line)
- void **setOctave** (long oct)
- long **getStaffNum** () const
- long **getSign** () const
- long **getLine** () const
- long **getOctave** () const

Static Public Member Functions

- const string **xmlsign** (int d)

  convert an integer clef sign to a MusicXML string

- int **xmlsign** (const string str)

  convert an MusicXML string to an integer clef sign

Friends

- EXP friend **SMARTP**< **TClef** > **newClef** ()

7.31.1 Detailed Description

Clefs are represented by the sign, line, and clef-octave-change elements. Sign values include G, F, C, percussion, TAB, and none. Line numbers are counted from the bottom of the staff. Standard values are 2 for the G sign (treble clef), 4 for the F sign (bass clef), 3 for the C sign (alto clef) and 5 for TAB (on a 6-line staff). The clef-octave-change element is used for transposing clefs (e.g., a treble clef for tenors would have a clef-octave-change value of -1). The optional number attribute refers to staff numbers, from top to bottom on the system. A value of 1 is assumed if not present.

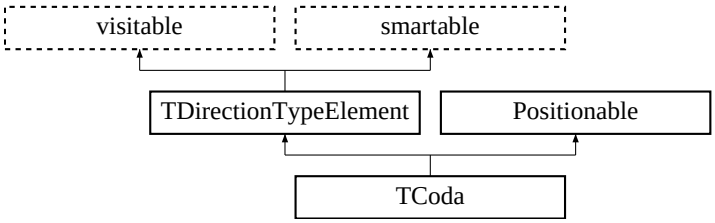
TClef(p. 60) corresponds to the MusicXML *clef* element as defined in attributes.dtd

7.32 TCoda Class Reference

Represents the MusicXML *coda* element.

`#include <common.h>`

Inheritance diagram for TCoda::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)

Friends

- EXP friend **SMARTP**< **TCoda** > **newCoda** ()

7.32.1 Detailed Description

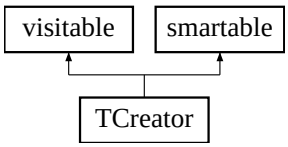
TCoda(p. 62) is a visual indicator only. It is defined in common.dtd.

7.33 TCreator Class Reference

Represents the author(s) of the score .

`#include <TIdentification.h>`

Inheritance diagram for TCreator::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setType** (const string &type)
- void **setName** (const string &name)
- const string & **getType** () const
- const string & **getName** () const

Friends

- EXP friend **SMARTP**< **TCreator** > **newCreator** ()

7.33.1 Detailed Description

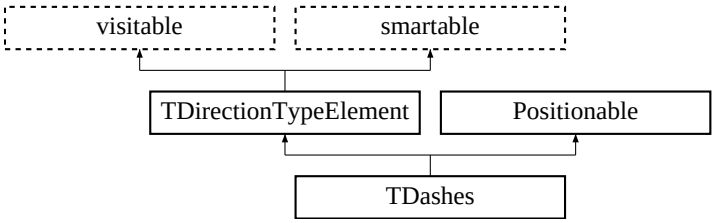
The type attribute can be used to distinguish words and music so as to represent lyricists and librettists as well as composers. Thus, there can be multiple creators in an identification. It corresponds to the MusicXML *creator* element as defined in identity.dtd.

7.34 TDashes Class Reference

Dashes, used for instance with cresc. and dim. marks.

`#include <TDirection.h>`

Inheritance diagram for TDashes::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setType** (StartStop::type t)
- void **setNumber** (int num)
- StartStop::type **getType** () const
- int **getNumber** () const

Protected Member Functions

- **TDashes** (StartStop::type t)

Friends

- EXP friend **SMARTP**< **TDashes** > **newDashes** (StartStop::type t)

7.34.1 Detailed Description

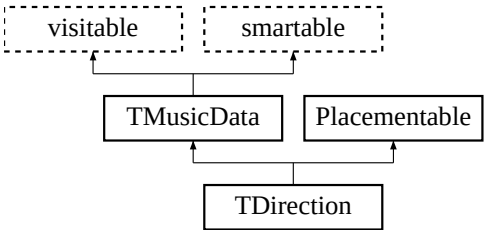
The MusicXML *dashes* element is defined in direction.dtd.

7.35 TDirection Class Reference

The MusicXML *direction* element.

`#include <TDirection.h>`

Inheritance diagram for TDirection::



Public Types

- enum { **undefined** = -1 }
  an undefined constant, used for offset, voice and staff

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **SDirectionType** &dir)
- void **setOffset** (long offset)
- void **setVoice** (long voice)
- void **setStaff** (long staff)
- **SSound** & **sound** ()
- long **getOffset** () const
- long **getVoice** () const
- long **getStaff** () const
- **vvector**< **SDirectionType** > & **types** ()

Friends

- EXP friend **SMARTP**< **TDirection** > **newDirection** ()

7.35.1 Detailed Description

A direction is a musical indication that is not attached to a specific note. Two or more may be combined to indicate starts and stops of wedges, dashes, etc.

Defined in direction.dtd.

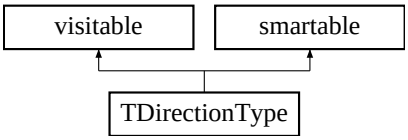
Note:
editorial information is ignored

7.36 TDirectionType Class Reference

Represents textual direction indications.

`#include <TDirection.h>`

Inheritance diagram for TDirectionType::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **SDirectionTypeElement** &elt)
- **vvector**< **SDirectionTypeElement** > & **elements** ()

Friends

- EXP friend **SMARTP**< **TDirectionType** > **newDirectionType** ()

7.36.1 Detailed Description

Textual direction types may have more than 1 component due to multiple font numbers.

TDirectionType(p. 66) corresponds to the MusicXML *direciton-type* element as defined in direction.dtd.

Todo

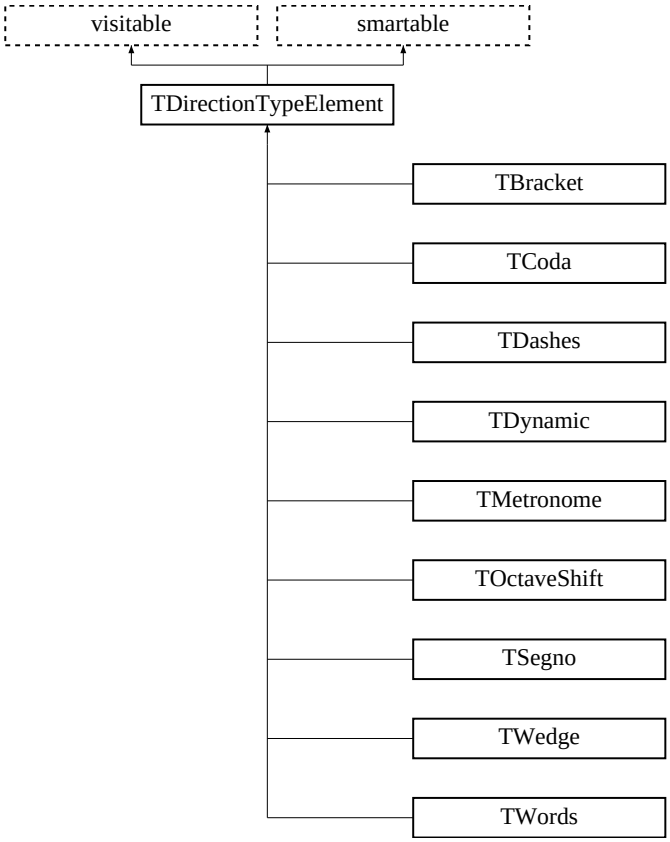
support of the following elements: rehearsal, pedal, damp, damp-all, eyeglasses, other-direction.

7.37 TDirectionTypeElement Class Reference

base class for all the elements of the *direction-type* element.

`#include <common.h>`

Inheritance diagram for TDirectionTypeElement::



7.37.1 Detailed Description

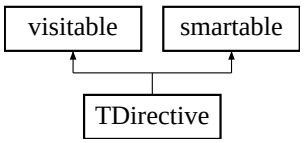
TDirectionTypeElement(p. 67) is an abstract class which purpose is to define a common type for all the elements of a *direction-type* element.

7.38 TDirective Class Reference

Represents musical directives.

`#include <TAttributes.h>`

Inheritance diagram for TDirective::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setLang** (const string &lang)

  the xml:lang attribute

- void **setValue** (const string &val)

  textual directive

- const string & **getLang** () const
- const string & **getValue** () const

Friends

- EXP friend **SMARTP**< **TDirective** > **newDirective** ()

7.38.1 Detailed Description

Directives are like directions, but can be grouped together with attributes for convenience. This is typically used for tempo markings at the beginning of a piece of music. The language is Italian ("it") by default.

TDirective(p. 68) corresponds to the MusicXML *directive* element.

Todo

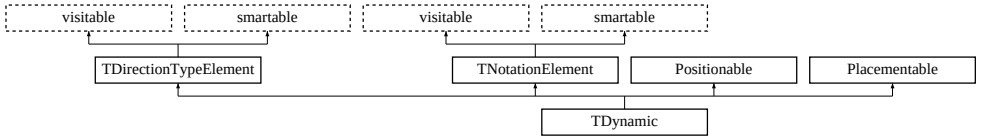
support of the *font* entity

7.39 TDynamic Class Reference

Represents the MusicXML *dynamics* element.

`#include <common.h>`

Inheritance diagram for TDynamic::



Public Types

- enum **dynamic** {

  ppppppp = 1, **pppppp**, **ppppp**, **pppp**,

  ppp, **pp**, **p**, **mp**, **mf**,

  f, **ff**, **fff**, **ffff**,

  fffff, **ffffff**, **sf**, **sfp**,

  sfpp, **fp**, **rf**, **rfz**,

  sfz, **sffz**, **fz**, **last** = fz }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- vector< dynamic > & **dynamics** ()
- long **add** (dynamic d)

Static Public Member Functions

- const string **xmldynamic** (dynamic d)

  convert an integer dynamic to a MusicXML string

- dynamic **xmldynamic** (const string str)

  convert an MusicXML string to an integer dynamic

- const string * **getDynamicStrings** ()

Friends

- EXP friend **SMARTP**< **TDynamic** > **newDynamic** ()

7.39.1 Detailed Description

It is defined in common.dtd.

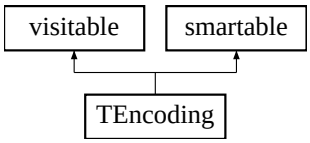
Todo
font support

7.40 TEncoding Class Reference

Contains information about encoding.

`#include <TIdentification.h>`

Inheritance diagram for TEncoding::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setDate** (const string &date)
- void **setEncoder** (const string &encoder)
- void **setEncoderType** (const string &etype)
- void **setSoftware** (const string &soft)
- void **setDescription** (const string &desc)
- const string & **getDate** () const
- const string & **getEncoder** () const
- const string & **getEncoderType** () const
- const string & **getSoftware** () const
- const string & **getDescription** () const

Friends

- EXP friend **SMARTP**< **TEncoding** > **newEncoding** ()

7.40.1 Detailed Description

Encoding contains information about who did the digital encoding, when, with what software, and in what aspects. The supports element indicates if the encoding supports a particular MusicXML element. This is recommended for elements like beam, stem, and accidental, where the absence of an element is ambiguous if you do not know if the encoding supports that element.

It corresponds to the MusicXML *encoding* element as defined in identity.dtd.

Todo

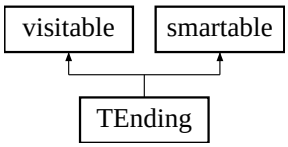
support of the new *support* element

7.41 TEnding Class Reference

Endings refers to multiple (e.g. first and second) endings.

`#include <TBarline.h>`

Inheritance diagram for TEnding::



Public Types

- enum { **start** = 1, **stop**, **discontinue**, **last** = discontinue }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setNum** (long num)
- void **setType** (int type)
- long **getNum** () const
- int **getType** () const

Static Public Member Functions

- const string **xmltype** (int d)

  convert an integer ending type to a MusicXML string

- int **xmltype** (const string str)

  convert an MusicXML string to an integer ending type

Protected Member Functions

- **TEnding** (long num, int type)

Friends

- EXP friend **SMARTP**< **TEnding** > **newEnding** (long num, int type)

7.41.1 Detailed Description

Typically, the start type is associated with the left barline of the first measure in an ending. The stop and discontinue types are associated with the right barline of the last measure in an ending. Stop is used when the ending mark concludes with a downward jog, as is the case for first endings. Discontinue is used when there is no downward jog, as in the final ending.

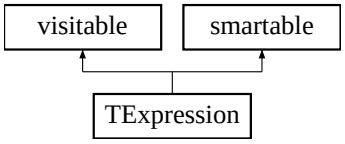
The MusicXML *ending* element is defined in barline.dtd.

7.42 TExpression Class Reference

Represents the expression context of a note.

`#include <TNote.h>`

Inheritance diagram for TExpression::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setDynamic** (long dyn)

  reflects the MIDI key on velocity (expressed in terms of percentages of a standard MIDI forte volume of 90)

- void **setEndDynamic** (long dyn)

  reflects the MIDI key off velocity (expressed in terms of percentages of a standard MIDI forte volume of 90)

- void **setAttack** (long attack)

  used to alter the starting time of the note (expressed in term of divisions)

- void **setRelease** (long rel)

  used to alter the stopping time of the note (expressed in term of divisions)

- void **setPizzicato** (YesNo::type pizz)

  used when just this note is sounded pizzicato

- long **getDynamic** () const
- long **getEndDynamic** () const
- long **getAttack** () const
- long **getRelease** () const
- YesNo::type **getPizzicato** () const

Friends

- EXP friend **SMARTP**< **TExpression** > **newExpression** ()

7.42.1 Detailed Description

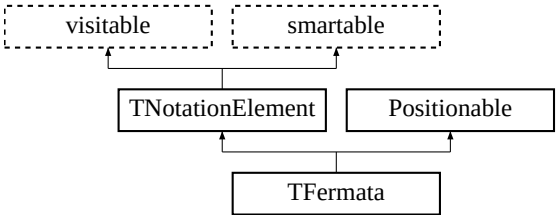
TExpression(p. 74) corresponds to a subset of the *note* attributes as defined in notes.dtd

7.43 TFermata Class Reference

Represents the fermata sign.

`#include <common.h>`

Inheritance diagram for TFermata::



Public Types

- enum { **undefined**, **upright**, **inverted**, **last** = inverted }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setType** (int type)
- int **getType** () const

Static Public Member Functions

- const string **xmltype** (int d)

  convert an integer type to a MusicXML string

- int **xmltype** (const string str)

  convert an MusicXML string to an integer type

Friends

- EXP friend **SMARTP**< **TFermata** > **newFermata** ()

7.43.1 Detailed Description

Fermata elements can be applied both to notes and to measures. Fermata type is upright if not specified.

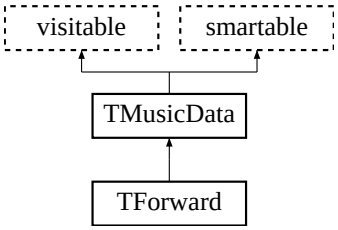
The MusicXML *fermata* element is defined in common.dtd

7.44 TForward Class Reference

Represents the MusicXML forward element.

`#include <TBackupForward.h>`

Inheritance diagram for TForward::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- **TForward** & **setDuration** (long dur)

  Duration is a required element and is set to zero by the constructor.

- **TForward** & **setVoice** (long voice)
- **TForward** & **setStaff** (long staff)
- long **getDuration** () const
- long **getVoice** () const
- long **getStaff** () const

Friends

- EXP friend **SMARTP**< **TForward** > **newForward** ()

7.44.1 Detailed Description

Backup and forward are required to coordinate multiple voices in one part, including music on a grand staff. Forward is generally used within voices and staves. Duration values should always be positive, and should not cross measure boundaries.

The MusicXML *forward* element is defined in note.dtd.

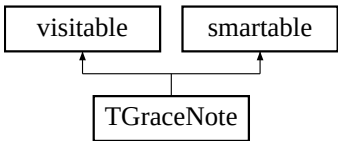
Note:
Editorial information is ignored.

7.45 TGraceNote Class Reference

grace note representation.

`#include <TNote.h>`

Inheritance diagram for TGraceNote::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- int **getStealPrevious** () const
- int **getStealFollowing** () const
- int **getMakeTime** () const
- YesNo::type **getSlash** () const
- void **setStealPrevious** (int v)
- void **setStealFollowing** (int v)
- void **setMakeTime** (int v)
- void **setSlash** (YesNo::type v)

Friends

- EXP friend **SMARTP**< **TGraceNote** > **newGraceNote** ()

7.45.1 Detailed Description

The slash attribute for a grace note is yes for slashed eighth notes. The other grace note attributes come from MuseData sound suggestions. Steal-time-previous indicates the percentage of time to steal from the previous note for the grace note. Steal-time-following indicates the percentage of time to steal from the following note for the grace note. Make-time indicates to make time, not steal time; the units are in real-time divisions for the grace note.

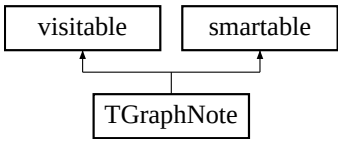
The MusicXML *grace* element is defined in note.dtd.

7.46 TGraphNote Class Reference

Represents the graphic elements of a note.

`#include <TNote.h>`

Inheritance diagram for TGraphNote::



Public Types

- enum {
  undefined = -1, **stemdown** = 1, **stemup**, **stemnone**,
  stemdouble, **last** = stemdouble }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setVoice** (long voice)
  the corresponding note voice
- void **setType** (NoteType::type type)
  the graphical note sign (whole, half, quarter, ...) corresponds to the type *element*
- void **setSize** (FullCue::type size)
  the note size (full or cue)
- void **setDots** (unsigned long dots)
  the number of dots
- void **setStem** (long stem)
  stems can be down, up, none, or double.
- void **setStaff** (long staff)
  staff assignment (only needed for music notated on the grand staff)
- long **getVoice** () const
- NoteType::type **getType** () const
- FullCue::type **getSize** () const
- unsigned int **getDots** () const
- int **getStem** () const
- int **getStaff** () const
- long **add** (const **SBeam** &beam)
- long **add** (const **SNotation** &notation)
- **SAccidental** & **accidental** ()

graphical accidentals

- **STimeModification** & **timemodification** ()
  graphical element for tuplets
- **SNoteHead** & **notehead** ()
  the shape of the note head
- **vvector**< **SBeam** > & **beamList** ()
  beaming definition
- **vvector**< **SNotation** > & **notationList** ()
  notations elements (accents, slurs, etc...)

Static Public Member Functions

- const string **xmlstem** (int d)
  convert an integer stem attribute to a MusicXML string
- int **xmlstem** (const string str)
  convert an MusicXML string to an integer stem attribute

Friends

- EXP friend **SMARTP**< **TGraphNote** > **newGraphNote** ()

7.46.1 Detailed Description

Graphic elements of a note represents the note type (whole, quarter, etc...), possible dots, accidentals, tuplets notation, stems, notehead, staff, beams and other notation elements.

Todo

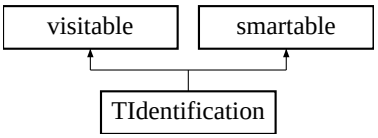
the MusicXML stem definition includes now a position attribute and should be implemented as a separate element

7.47 TIdentification Class Reference

Contains metadata about the score.

`#include <TIdentification.h>`

Inheritance diagram for TIdentification::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **SCreator** &creator)
- long **add** (const **SRights** &rights)
- void **setSource** (const string &src)
- **vvector**< **SCreator** > & **creators** ()
- **vvector**< **SRights** > & **rights** ()
- const string & **getSource** ()
- **SEncoding** & **encoding** ()

Friends

- EXP friend **SMARTP**< **TIdentification** > **newIdentification** ()

7.47.1 Detailed Description

It includes information from the initial 13+ records in a MuseData file that may apply at a score-wide, movement- wide, or part-wide level. It corresponds to the MusicXML *identification* element as defined in identity.dtd.

Todo

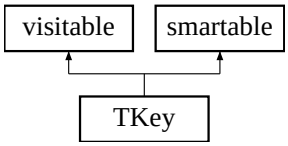
support of the miscellaneaous element

7.48 TKey Class Reference

Represents a key signature.

`#include <TKey.h>`

Inheritance diagram for TKey::



Public Types

- enum { **undefined** = -1 }
- enum {

  major = 1, **minor**, **dorian**, **phrygian**,

  lydian, **mixolydian**, **aeolian**, **ionian**,

  locrian, **last** = locrian }

Public Member Functions

- void **setCancel** (long cancel)
- void **setFifths** (long fifths)
- void **setMode** (long mode)
- long **getCancel** () const
- long **getFifths** () const
- long **getMode** () const
- virtual void **accept** (**TScoreVisitor** &visitor)

Static Public Member Functions

- const string **xmlmode** (int d)

  convert an integer mode to a MusicXML string

- int **xmlmode** (const string str)

  convert an MusicXML string to an integer mode

Friends

- EXP friend **SMARTP**< **TKey** > **newKey** ()

7.48.1 Detailed Description

Traditional key signatures are represented by the number of flats and sharps, plus an optional mode for major/ minor/mode distinctions. Negative numbers are used for flats and positive numbers for sharps, reflecting the key's placement within the circle of fifths (hence the element name). A cancel element indicates that the old key signature should be cancelled before the new one appears. This will always happen when changing to C major or A minor and need not be specified then. The cancel value matches the fifths value of the cancelled key signature (e.g., a cancel of -2 will provide an explicit cancellation for changing from B flat major to F major).

Non-traditional key signatures can be represented using the Humdrum/Scot concept of a list of altered tones. The key-step and key-alter elements are represented the same way as the step and alter elements are in the pitch element in note.dtd. The different element names indicate the different meaning of altering notes in a scale versus altering a sounding pitch.

Valid mode values include major, minor, dorian, phrygian, lydian, mixolydian, aeolian, ionian, and locrian.

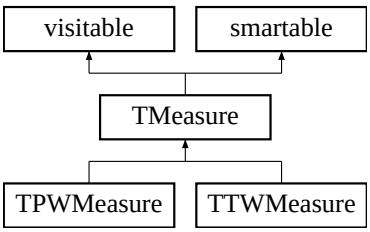
TKey(p. 81) corresponds to the MusicXML *key* element as defined in attributes.dtd.

7.49 TMeasure Class Reference

The MusicXML base class for partwise and timewise *measure* elements.

`#include <TMeasure.h>`

Inheritance diagram for TMeasure::



Public Member Functions

- void **setStringNumber** (const string &num)
- void **setImplicit** (YesNo::type type)
- void **setNonControlling** (YesNo::type type)
- const string & **getStringNumber** () const
- long **getNumber** () const
- YesNo::type **getImplicit** () const
- YesNo::type **getNonControlling** () const

Protected Member Functions

- **TMeasure** (long num)
- **TMeasure** (string num)

7.49.1 Detailed Description

Measures have a required measure number attribute.

The *implicit* attribute is set to "yes" for measures where the measure number should never appear, such as pickup measures and the last half of mid-measure repeats. The implicit attribute is "no" if not specified.

The *non-controlling* attribute indicates that this measure in this part does not necessarily synchronize with other measures in other parts.

The MusicXML *measure* element is defined in score.dtd.

Todo

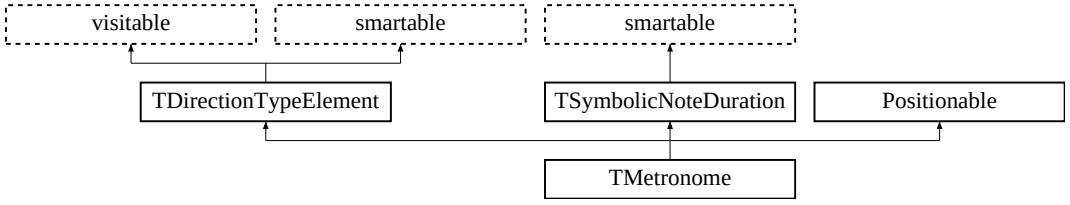
support of the *implicit* and *non-controlling* attributes.

7.50 TMetronome Class Reference

Standard metronome marks.

`#include <TDirection.h>`

Inheritance diagram for TMetronome::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setParentheses** (YesNo::type val)
- void **setPerMinute** (string pm)
- void **setPerMinute** (long pm)
- YesNo::type **getParentheses** () const
- string **getPerMinute** () const
- **SSymbolicNoteDuration** & **equal** ()

Friends

- EXP friend **SMARTP**< **TMetronome** > **newMetronome** ()

7.50.1 Detailed Description

The beat-unit values are the same as for a type element, and the beat-unit-dot works like the dot element. The per-minute element can be a number, or a text description including numbers. The parentheses attribute indicates whether or not to put the metronome mark in parentheses; its value is no if not specified.

The MusicXML *metronome* element is defined in direction.dtd.

7.50.2 Member Function Documentation

7.50.2.1 SSymbolicNoteDuration& equal () `[inline]`

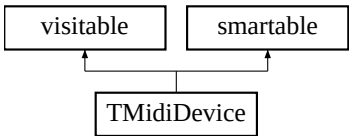
used to specify metronome marking such as <symbolic duration = symbolic duration> ("eight note = eight note" for example). When the metronome uses this symbolic form, the PerMinute value is ignored. To cancel the symbolic form, set **equal()**(p. 84)=0

7.51 TMidiDevice Class Reference

Corresponds to the DeviceName meta event in Standard MIDI Files.

`#include <TMidiDevice.h>`

Inheritance diagram for TMidiDevice::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setPort** (long port)
- void **setName** (const string &name)
- long **getPort** () const
- const string & **getName** () const

Friends

- EXP friend **SMARTP**< **TMidiDevice** > **newMidiDevice** ()

7.51.1 Detailed Description

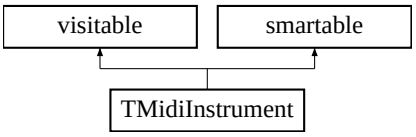
The optional port attribute is a number from 1 to 16 that can be used with the unofficial MIDI port (or cable) meta event. It correspond to the MusicXML *midi-device* element as defined in score.dtd.

7.52 TMidiInstrument Class Reference

Represents a MIDI instrument.

`#include <TMidiInstrument.h>`

Inheritance diagram for TMidiInstrument::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- void **setID** (const string &id)

  ID is a MusicXML required attribute that refers to the score-instrument affected by the change.

- void **setChannel** (long chan)

  optionnal MIDI channel numbers range from 1 to 16.

- void **setName** (const string &name)

  optionnal MIDI name, corresponds to ProgramName meta-events within a Standard MIDI File.

- void **setBank** (long bank)

  optionnal MIDI bank number, range from 1 to 16,384.

- void **setProgram** (long prog)

  optionnal MIDI program number, range from 1 to 128.

- void **setUnpitched** (long pitch)
- const string & **getID** () const
- long **getChannel** () const
- const string & **getName** () const
- long **getBank** () const
- long **getProgram** () const
- long **getUnpitched** () const
- virtual void **accept** (**TScoreVisitor** &visitor)

Protected Member Functions

- **TMidiInstrument** (string ident)

Friends

- EXP friend **SMARTP< TMidiInstrument > newMidiInstrument** (string id)

7.52.1 Detailed Description

TMidiInstrument(p. 86) includes the necessary to address a specific MIDI instrument (channel, bank, program) + additionnal information required by the MusicXML *midi-instrument* element as defined in common.dtd.

7.52.2 Member Function Documentation

7.52.2.1 void setUnpitched (long *pitch*) [inline]

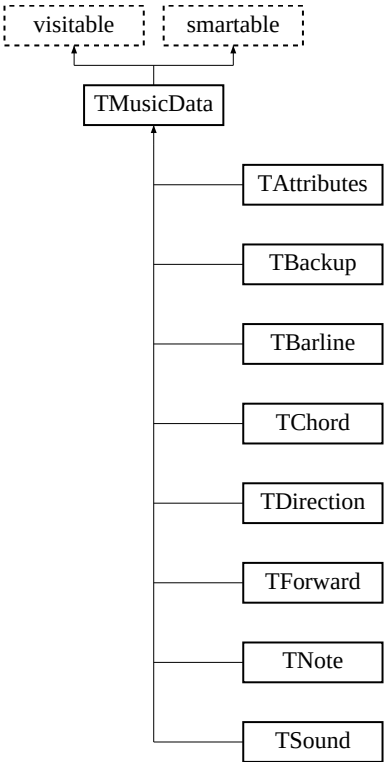
optionnal : for unpitched instruments, specify a MIDI note number ranging from 1 to 128. Usually used with MIDI banks for percussion.

7.53 TMusicData Class Reference

base class for all the elements of the *music-data* entity.

`#include <common.h>`

Inheritance diagram for TMusicData::



7.53.1 Detailed Description

TMusicData(p. 88) is an abstract class which purpose is to define a common type for all the elements covered by the *music-data* entity.

7.54 TMusicXMLFile Class Reference

provides MusicXML files reading and writing

`#include <TMusicXMLFile.h>`

Public Member Functions

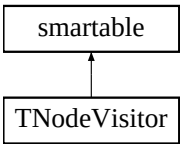
- **SScore read** (const string &file)
- bool **write** (**SScore** score, const string &file)
- bool **write** (**SScore** score, ostream &os)

7.55 TNodeVisitor Class Reference

A node Visitor of MusicXML scores.

`#include <TNodeVisitor.h>`

Inheritance diagram for TNodeVisitor::



Public Member Functions

- virtual void **visiteEnter** (**SAccidental** &elt)
- virtual void **visiteEnter** (**SArticulationElement** &elt)
- virtual void **visiteEnter** (**SArticulations** &elt)
- virtual void **visiteEnter** (**SAttributes** &elt)
- virtual void **visiteEnter** (**SBackup** &elt)
- virtual void **visiteEnter** (**SBarline** &elt)
- virtual void **visiteEnter** (**SBeam** &elt)
- virtual void **visiteEnter** (**SBracket** &elt)
- virtual void **visiteEnter** (**SChord** &elt)
- virtual void **visiteEnter** (**SClef** &elt)
- virtual void **visiteEnter** (**SCoda** &elt)
- virtual void **visiteEnter** (**SCreator** &elt)
- virtual void **visiteEnter** (**SDashes** &elt)
- virtual void **visiteEnter** (**SDirection** &elt)
- virtual void **visiteEnter** (**SDirectionType** &elt)
- virtual void **visiteEnter** (**SDirective** &elt)
- virtual void **visiteEnter** (**SDynamic** &elt)
- virtual void **visiteEnter** (**SEncoding** &elt)
- virtual void **visiteEnter** (**SEnding** &elt)
- virtual void **visiteEnter** (**SExpression** &elt)
- virtual void **visiteEnter** (**SFermata** &elt)
- virtual void **visiteEnter** (**SForward** &elt)
- virtual void **visiteEnter** (**SGraceNote** &elt)
- virtual void **visiteEnter** (**SGraphNote** &elt)
- virtual void **visiteEnter** (**SIdentification** &elt)
- virtual void **visiteEnter** (**SKey** &elt)
- virtual void **visiteEnter** (**SMeasure** &elt)
- virtual void **visiteEnter** (**SMetronome** &elt)
- virtual void **visiteEnter** (**SMidiDevice** &elt)
- virtual void **visiteEnter** (**SMidiInstrument** &elt)
- virtual void **visiteEnter** (**SNotation** &elt)
- virtual void **visiteEnter** (**SNote** &elt)
- virtual void **visiteEnter** (**SNoteHead** &elt)
- virtual void **visiteEnter** (**SOctaveShift** &elt)

- virtual void **visiteEnter** (**SPWMeasure** &elt)
- virtual void **visiteEnter** (**SPWPart** &elt)
- virtual void **visiteEnter** (**SPart** &elt)
- virtual void **visiteEnter** (**SPartGroup** &elt)
- virtual void **visiteEnter** (**SPartList** &elt)
- virtual void **visiteEnter** (**SPitch** &elt)
- virtual void **visiteEnter** (**SRepeat** &elt)
- virtual void **visiteEnter** (**SRest** &elt)
- virtual void **visiteEnter** (**SRights** &elt)
- virtual void **visiteEnter** (**SScoreHeader** &elt)
- virtual void **visiteEnter** (**SScoreInstrument** &elt)
- virtual void **visiteEnter** (**SScorePart** &elt)
- virtual void **visiteEnter** (**SScorePartwise** &elt)
- virtual void **visiteEnter** (**SScoreTimewise** &elt)
- virtual void **visiteEnter** (**SSegno** &elt)
- virtual void **visiteEnter** (**SSlur** &elt)
- virtual void **visiteEnter** (**SSound** &elt)
- virtual void **visiteEnter** (**SStrongAccent** &elt)
- virtual void **visiteEnter** (**STWMeasure** &elt)
- virtual void **visiteEnter** (**STWPart** &elt)
- virtual void **visiteEnter** (**STie** &elt)
- virtual void **visiteEnter** (**STimeModification** &elt)
- virtual void **visiteEnter** (**STimeSign** &elt)
- virtual void **visiteEnter** (**STranspose** &elt)
- virtual void **visiteEnter** (**STuplet** &elt)
- virtual void **visiteEnter** (**STupletDesc** &elt)
- virtual void **visiteEnter** (**SUnpitched** &elt)
- virtual void **visiteEnter** (**SWavyLine** &elt)
- virtual void **visiteEnter** (**SWedge** &elt)
- virtual void **visiteEnter** (**SWords** &elt)
- virtual void **visiteEnter** (**SWork** &elt)
- virtual void **visiteLeave** (**SAccidental** &elt)
- virtual void **visiteLeave** (**SArticulationElement** &elt)
- virtual void **visiteLeave** (**SArticulations** &elt)
- virtual void **visiteLeave** (**SAttributes** &elt)
- virtual void **visiteLeave** (**SBackup** &elt)
- virtual void **visiteLeave** (**SBarline** &elt)
- virtual void **visiteLeave** (**SBeam** &elt)
- virtual void **visiteLeave** (**SBracket** &elt)
- virtual void **visiteLeave** (**SChord** &elt)
- virtual void **visiteLeave** (**SClef** &elt)
- virtual void **visiteLeave** (**SCoda** &elt)
- virtual void **visiteLeave** (**SCreator** &elt)
- virtual void **visiteLeave** (**SDashes** &elt)
- virtual void **visiteLeave** (**SDirection** &elt)
- virtual void **visiteLeave** (**SDirectionType** &elt)
- virtual void **visiteLeave** (**SDirective** &elt)
- virtual void **visiteLeave** (**SDynamic** &elt)
- virtual void **visiteLeave** (**SEncoding** &elt)
- virtual void **visiteLeave** (**SEnding** &elt)

- virtual void **visiteLeave** (**SExpression** &elt)
- virtual void **visiteLeave** (**SFermata** &elt)
- virtual void **visiteLeave** (**SForward** &elt)
- virtual void **visiteLeave** (**SGraceNote** &elt)
- virtual void **visiteLeave** (**SGraphNote** &elt)
- virtual void **visiteLeave** (**SIdentification** &elt)
- virtual void **visiteLeave** (**SKey** &elt)
- virtual void **visiteLeave** (**SMeasure** &elt)
- virtual void **visiteLeave** (**SMetronome** &elt)
- virtual void **visiteLeave** (**SMidiDevice** &elt)
- virtual void **visiteLeave** (**SMidiInstrument** &elt)
- virtual void **visiteLeave** (**SNotation** &elt)
- virtual void **visiteLeave** (**SNote** &elt)
- virtual void **visiteLeave** (**SNoteHead** &elt)
- virtual void **visiteLeave** (**SOctaveShift** &elt)
- virtual void **visiteLeave** (**SPWMeasure** &elt)
- virtual void **visiteLeave** (**SPWPart** &elt)
- virtual void **visiteLeave** (**SPart** &elt)
- virtual void **visiteLeave** (**SPartGroup** &elt)
- virtual void **visiteLeave** (**SPartList** &elt)
- virtual void **visiteLeave** (**SPitch** &elt)
- virtual void **visiteLeave** (**SRepeat** &elt)
- virtual void **visiteLeave** (**SRest** &elt)
- virtual void **visiteLeave** (**SRights** &elt)
- virtual void **visiteLeave** (**SScoreHeader** &elt)
- virtual void **visiteLeave** (**SScoreInstrument** &elt)
- virtual void **visiteLeave** (**SScorePart** &elt)
- virtual void **visiteLeave** (**SScorePartwise** &elt)
- virtual void **visiteLeave** (**SScoreTimewise** &elt)
- virtual void **visiteLeave** (**SSegno** &elt)
- virtual void **visiteLeave** (**SSlur** &elt)
- virtual void **visiteLeave** (**SSound** &elt)
- virtual void **visiteLeave** (**SStrongAccent** &elt)
- virtual void **visiteLeave** (**STWMeasure** &elt)
- virtual void **visiteLeave** (**STWPart** &elt)
- virtual void **visiteLeave** (**STie** &elt)
- virtual void **visiteLeave** (**STimeModification** &elt)
- virtual void **visiteLeave** (**STimeSign** &elt)
- virtual void **visiteLeave** (**STranspose** &elt)
- virtual void **visiteLeave** (**STuplet** &elt)
- virtual void **visiteLeave** (**STupletDesc** &elt)
- virtual void **visiteLeave** (**SUnpitched** &elt)
- virtual void **visiteLeave** (**SWavyLine** &elt)
- virtual void **visiteLeave** (**SWedge** &elt)
- virtual void **visiteLeave** (**SWords** &elt)
- virtual void **visiteLeave** (**SWork** &elt)
- virtual void **visiteEnter** (**TOrientation** *elt)
- virtual void **visiteEnter** (**TPlacement** *elt)
- virtual void **visiteEnter** (**TPosition** *elt)

Friends

- EXP friend **SMARTP< TNodeVisitor > newNodeVisitor** ()

7.55.1 Detailed Description

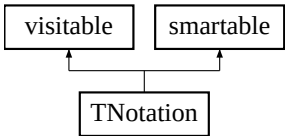
A node Visitor of MusicXML scores. Nodes visitor are to be used with a **TRolledVisitor**(p. 121) or **TUnrolledVisitor**(p. 156) that does the actual score traversing. A node Visitor can implement `visiteEnter` and/or `visiteLeave` methods for each type of noode.

7.56 TNotation Class Reference

Represents musical notations.

`#include <TNotation.h>`

Inheritance diagram for TNotation::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **SNotationElement** &elt)
- **vvector**< **SNotationElement** > & **list** ()

Friends

- EXP friend **SMARTP**< **TNotation** > **newNotation** ()

7.56.1 Detailed Description

Multiple notations are allowed in order to represent multiple editorial levels. The set of notations will be refined and expanded over time, especially to handle more instrument-specific technical notations.

TNotation(p. 94) corresponds to the MusicXML *notation* element as defined in note.dtd.

Note:
editorial level is not implemented

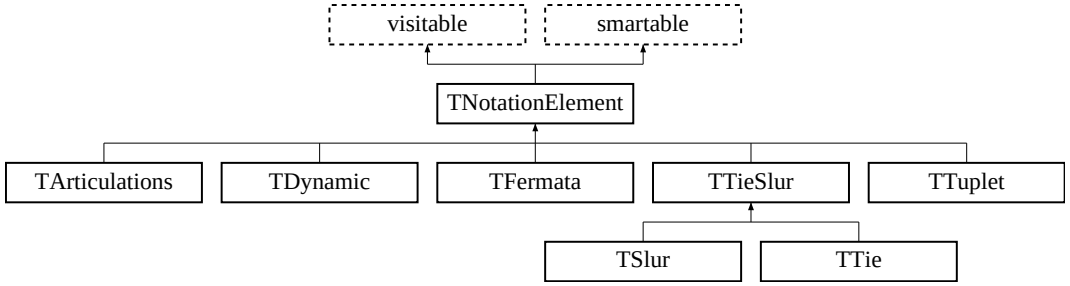
Todo
implementation of the following elements: glissando, slide, ornaments technical, arpeggiate, non-arpeggiate and other-notation

7.57 TNotationElement Class Reference

base class for all the elements of the *notation* element.

`#include <common.h>`

Inheritance diagram for TNotationElement::



7.57.1 Detailed Description

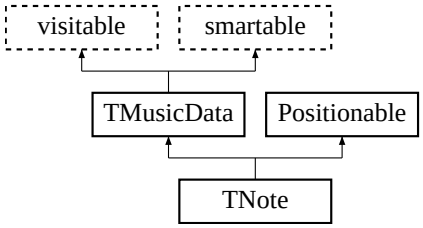
TNotationElement(p. 95) is an abstract class which purpose is to define a common type for all the elements of a *notation* element.

7.58 TNote Class Reference

TNote(p. 96) is the main common type of the music representation.

`#include <TNote.h>`

Inheritance diagram for TNote::



Public Types

- enum **style** { **normal**, **cue**, **grace** }
- enum **type** { **pitched**, **rest**, **unpitched** }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setDuration** (unsigned long dur)
- unsigned long **getDuration** () const
- unsigned long **getFullDuration** ()
- void **setInstrumentID** (string id)

  the score-instrument (if any)

- const string **getInstrumentID** () const
- void **setStyle** (style t)

  to set the style of the note (normal, cue or grace - defaults to normal)

- style **getStyle** () const
- void **setType** (type t)

  to set the type of the note (regular, rest, unpitched - defaults to regular)

- type **getType** () const
- **SPitch** & **pitch** ()

  the note pitch. May also be used for unpitched and rest placement.

- **SExpression** & **expression** ()

  the expression context of the note, expressed in terms of current division

- **SNote** & **tiePrevious** ()

  the optional previous tied note (corresponds to the tie *element)*

- **SNote** & **tieNext** ()

  the optional next tied note (corresponds to the tie *element)*

- **SGraphNote** & **graphics** ()

  the graphical attributes of the note

- **SGraceNote** & **graceNote** ()

  the grace note optionnal element

- bool **isTie** ()

  the note is the first of a group of tied notes

- bool **isSingle** ()

  the note is "single", not part of a tied group

- bool **isNormal** ()
- bool **isGrace** ()
- bool **isCue** ()
- bool **isPitched** ()
- bool **isRest** ()
- bool **isUnpitched** ()
- long **getVoice** () const

Friends

- EXP friend **SMARTP**< **TNote** > **newNote** ()

7.58.1 Detailed Description

It is derived from the MusicXML note element as described in note.dtd but only the sounding attributes has been retained at core level. All the other elements are stored in contextual information which is divided in 2 parts:

- the graphical notation context (like voice, notehead, accidentals etc...), available using the **graphics()**(p. 97) method
- the expression context (such as dynamics, attack, release etc...), available using the **expression()**(p. 96) method

Todo

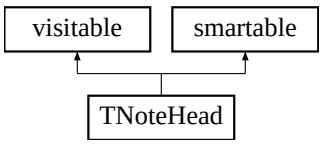
lyrics support

7.59 TNoteHead Class Reference

the shape of the note head

`#include <TNote.h>`

Inheritance diagram for TNoteHead::



Public Types

- enum **notehead** {
  undefined = -1, **slash** = 1, **triangle**, **diamond**,
  square, **cross**, **x**, **circle_x**,
  normal, **none**, **last** = none }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setShape** (notehead shape)
- void **setFilled** (YesNo::type filled)
- void **setParentheses** (YesNo::type par)
- notehead **getShape** () const
- YesNo::type **getFilled** () const
- YesNo::type **getParentheses** () const

Static Public Member Functions

- const string **xmlnotehead** (notehead d)

  convert a numeric beam type to a MusicXML string

- notehead **xmlnotehead** (const string str)

  convert an MusicXML string to a numeric beam type

Friends

- EXP friend **SMARTP**< **TNoteHead** > **newNoteHead** ()

7.59.1 Detailed Description

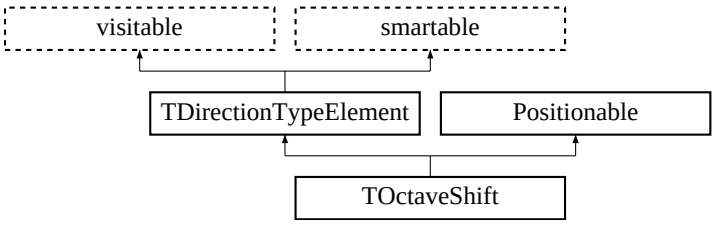
The notehead element indicates shapes other than the open and closed ovals associated with note durations. The element value can be slash, triangle, diamond, square, cross, x, circle-x, normal, or none. For the enclosed shapes, the default is to be hollow for half notes and longer, and filled otherwise. The filled attribute can be set to change this if needed. If the parentheses attribute is set to yes, the notehead is parenthesized. It is no by default.

The MusicXML *notehead* element as defined in note.dtd.

7.60 TOctaveShift Class Reference

`#include <TDirection.h>`

Inheritance diagram for TOctaveShift::



Public Types

- enum {
  undefined = -1, **up** = 1, **down**, **stop**,
  last = stop }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setType** (int type)
- void **setNumber** (int num)
- void **setSize** (int size)
- int **getType** () const
- int **getNumber** () const
- int **getSize** () const

Static Public Member Functions

- const string **xmltype** (int d)
  convert an numeric type value to a MusicXML string
- int **xmltype** (const string str)
  convert an MusicXML string to a numeric type value

Protected Member Functions

- **TOctaveShift** (int type)

Friends

- EXP friend **SMARTP**< **TOctaveShift** > **newOctaveShift** (int type)

7.60.1 Detailed Description

The MusicXML *octave-shift* element.

Octave shifts indicate where notes are shifted up or down from their true pitched values because of printing difficulty. Thus a treble clef line noted with 8va will be indicated with an octave-shift down from the pitch data indicated in the notes. A size of 8 indicates one octave; a size of 15 indicates two octaves.

Defined in direction.dtd.

7.61 TOrientation Class Reference

Represents the MusicXml *orientation* entity.

`#include <common.h>`

Public Types

- enum **type** { **undefined**, **over**, **under**, **last** = under }

Public Member Functions

- **TOrientation** (int orient=undefined)
- virtual void **accept** (**TScoreVisitor** &visitor)
- void **set** (int orient)
- int **get** () const

Static Public Member Functions

- const string **xmlorientation** (int d)

  convert an integer orientation to a MusicXML string

- int **xmlorientation** (const string str)

  convert an MusicXML string to an integer orientation

7.61.1 Detailed Description

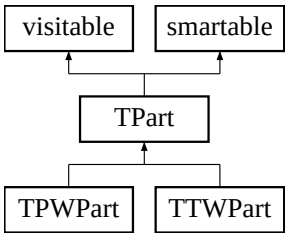
The orientation attribute indicates whether slurs and ties are overhand (tips down) or underhand (tips up). It is defined in common.dtd.

7.62 TPart Class Reference

The base class for the MusicXML timewise and partwise *part* elements.

`#include <TPart.h>`

Inheritance diagram for TPart::



Public Member Functions

- void **setID** (const string &id)
- const string & **getID** () const

Protected Member Functions

- **TPart** (const string &id)

7.62.1 Detailed Description

In either format (partwise or timewise), the *part* element has a required id attribute that is an IDREF back to a score-part in the part-list.

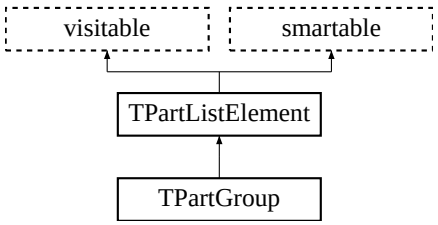
The MusicXML *part* element is defined in score.dtd.

7.63 TPartGroup Class Reference

Indicates groupings of parts in a score.

`#include <TPartGroup.h>`

Inheritance diagram for TPartGroup::



Public Types

- enum **symbol** {
  undefined = -1, **none** = 1, **brace**, **line**,
  bracket, **last** = bracket }
  the possible values for the group symbol

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setType** (StartStop::type type)
  the required type attribute
- void **setNumber** (int num)
  the optional group number
- void **setGroupName** (string name)
  optional group name
- void **setGroupAbbrev** (string abbrev)
  optional name abbreviation
- void **setSymbol** (**symbol** sym)
  optional group symbol
- StartStop::type **getType** () const
- int **getNumber** () const
- string **getGroupName** () const
- string **getGroupAbbrev** () const
- **symbol getSymbol** () const

Static Public Member Functions

- const string **xmlsymbol** (**symbol** d)
  convert an integer dynamic to a MusicXML string
- **symbol xmlsymbol** (const string str)
  convert an MusicXML string to an integer dynamic

Protected Member Functions

- **TPartGroup** (StartStop::type type)

Friends

- EXP friend **SMARTP**< **TPartGroup** > **newPartGroup** (StartStop::type type)

7.63.1 Detailed Description

Groupings of parts in the score are usually indicated by braces and brackets. The number attribute is used to distinguish overlapping and nested part-groups, not the sequence of groups. As with parts, groups can have a name and abbreviation. The group-symbol element indicates how the group is indicated on the score. Values include none (default), brace, line, and bracket. Values for the child elements are ignored at the stop of a group.

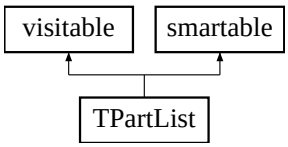
TPartGroup(p. 104) corresponds to the MusicXML *score-part* element as defined in score.dtd.

7.64 TPartList Class Reference

Identifies the different musical parts in a movement.

`#include <TPartList.h>`

Inheritance diagram for TPartList::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **SScorePart** &part)
- long **add** (const **SPartGroup** &part)
- **vvector**< **SPartListElement** > & **parts** ()

Friends

- EXP friend **SMARTP**< **TPartList** > **newPartList** ()

7.64.1 Detailed Description

There must be at least one score-part, combined as desired with part-group elements that indicate braces and brackets. Parts are ordered from top to bottom in a score based on the order in which they appear in the part-list.

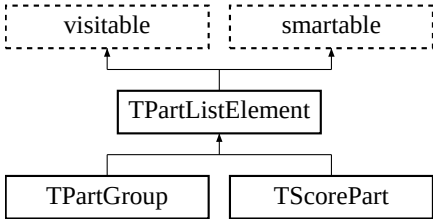
TPartList(p. 106) corresponds to the MusicXML *part-list* element as defined in score.dtd.

7.65 TPartListElement Class Reference

base class for the elements of a *part-list*.

`#include <common.h>`

Inheritance diagram for TPartListElement::



7.65.1 Detailed Description

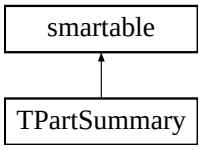
TPartListElement(p. 107) is an abstract class which purpose is to define a common type for all the elements of a *part-list* element.

7.66 TPartSummary Class Reference

summary of a MusicXML part.

`#include <TScanVisitor.h>`

Inheritance diagram for TPartSummary::



Public Member Functions

- const string **getID** () const
  returns the part ID
- void **setStaff** (long staff, long index)
  associates the staff to a value (a global unique index when used by **TScanVisitor**(p. 125)*)*
- long **getStaffIndex** (long staff)
  returns the staff associated value (a global unique index when used by **TScanVisitor**(p. 125)*)*
- long **getStaff** (long index) const
  returns the staff number from its index in the staff list
- void **incCue** (long staff)
  increments the number of cue notes for the corresponding staff
- void **incVoice** (long voice, long staff)
  increments the number of notes for the corresponding voice
- long **countStaves** () const
  returns the number of staves for the part
- long **countCueNotes** (long staff)
  returns the number of cue notes for the staff
- long **countVoices** () const
  returns the number of voices for the part
- long **countVoices** (long staff)
  returns the number of voices for the staff
- long **countVoiceNotes** (long voice)
  returns the number of notes on a voice
- long **countStaffNotes** (long staff)

returns the number of notes on a staff

- long **countNotes** (long voice, long staff)
  returns the number of notes for both a voices and a staff
- long **getVoice** (long index) const
  returns the voice number for the corresponding index
- long **getMainStaff** (long voice)
  returns the main staff for a voice (the staff where most of the notes are located)

Protected Member Functions

- **TPartSummary** (string id)

Friends

- VEXP friend **SMARTP**< **TPartSummary** > **newPartSummary** (string id)

7.66.1 Detailed Description

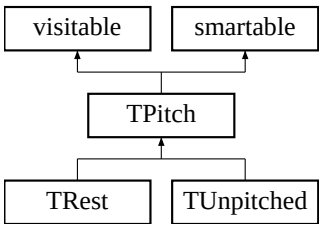
A part summary consists in a list of voices included in the part, a count of cue notes associated to each staff, a count of staves associated to a unique index in the global score context.

7.67 TPitch Class Reference

The MusicXML representation of the pitch.

`#include <TPitch.h>`

Inheritance diagram for TPitch::



Public Types

- enum {
  undefined = 0, **C** = 1, **D**, **E**,
  F, **G**, **A**, **B**,
  last = B, **diatonicSteps** = last }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- int **getStep** () const
- float **getAlter** () const
- int **getOctave** () const
- void **setStep** (int step)
- void **setAlter** (float alter)
- void **setOctave** (int oct)
- **TPitch** & **operator++** (int)

  the ++ operator adds a sharp

- **TPitch** & **operator–** (int)

  the – operator adds a flat

- **TPitch** & **operator+=** (int n)
- **TPitch** & **operator+=** (unsigned int n)

  adds n diatonic steps to the pitch field and update the octave when necessary

- **TPitch** & **operator-=** (int n)
- **TPitch** & **operator-=** (unsigned int n)

  substracts n diatonic steps to the pitch field and update the octave when necessary

- unsigned short **MIDIPitch** () const

  returns the corresponding MIDI pitch

Static Public Member Functions

- const string **xmlpitch** (int d)

  convert a numeric pitch to a MusicXML string

- int **xmlpitch** (const string str)

  convert a MusicXML string to a numeric pitch

Protected Attributes

- int **fStep**
- float **fAlter**
- int **fOctave**

Friends

- EXP friend **SMARTP**< **TPitch** > **newPitch** ()

7.67.1 Detailed Description

Pitch is represented as a combination of the step of the diatonic scale, the chromatic alteration, and the octave. The step element uses the English letters A through G: in a future revision, this could expand to international namings. The alter element represents chromatic alteration in number of semitones (e.g., -1 for flat, 1 for sharp). Decimal values like 0.5 (quarter tone sharp) may be used for microtones. The octave element is represented by the numbers 0 to 9, where 4 indicates the octave started by middle C.

The MusicXML *pitch* element is defined in note.dtd.

7.68 TPlacement Class Reference

Represents the MusicXml *placement* entity.

`#include <common.h>`

Public Types

- enum **type** { **undefined**, **above**, **below**, **last** = below }

Public Member Functions

- **TPlacement** (int place=undefined)
- void **accept** (**TScoreVisitor** &visitor)
- void **set** (int place)
- int **get** () const

Static Public Member Functions

- const string **xmlplace** (int d)

  convert an integer placement to a MusicXML string

- int **xmlplace** (const string str)

  convert an MusicXML string to an integer placement

7.68.1 Detailed Description

It is defined in common.dtd.

7.69 TPosition Class Reference

Represents a MusicXML *position* entity.

`#include <common.h>`

Public Types

- enum {
  undefined = 0, **defaultx** = 1, **defaulty**, **relativex**,
  relativey, **last** = relativey }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **set** (int pos, long val)
- long **get** (int pos) const

Static Public Member Functions

- const string **xmlpos** (int d)
  convert an integer position to a MusicXML string
- int **xmlpos** (const string str)
  convert an MusicXML string to an integer position

7.69.1 Detailed Description

The position attributes are based on MuseData print suggestions. For most elements, any program will compute a default x and y position. The position attribute lets this be changed two ways. The default-x and default-y attributes change the computation of the default position. The origin becomes the left-hand side of the note or the musical position within the bar (x) and the top line of the staff (y). The relative-x and relative-y attributes change the position relative to the default position, either as computed by the individual program, or as overridden by the default-x and default-y attributes.

Positive x is right, negative x is left; positive y is up, negative y is down. All units are in tenths of interline space. Positions can be applied to notes, notations, directions, and stems. For stems, positive y lengthens a stem while negative y shortens it. Negative values for default-y are not allowed.

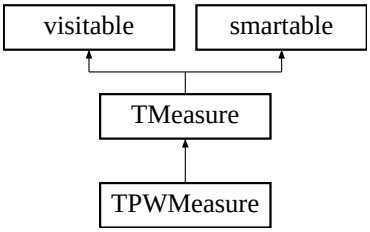
position is defined in common.dtd

7.70 TPWMeasure Class Reference

The MusicXML partwise *measure* element.

`#include <TScorePartwise.h>`

Inheritance diagram for TPWMeasure::



Public Types

- enum { **allVoices** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **SMusicData** &data)

  adds music data and returns its index;

- **vvector**< **SMusicData** > & **data** ()
- unsigned long **getDuration** (long voice=allVoices)

Protected Member Functions

- **TPWMeasure** (string num)
- **TPWMeasure** (long num)

Friends

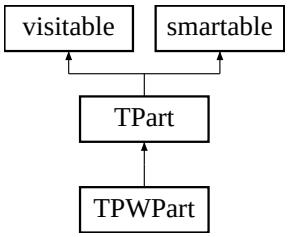
- EXP friend **SMARTP**< **TPWMeasure** > **newPWMeasure** (string num)
- EXP friend **SMARTP**< **TPWMeasure** > **newPWMeasure** (long num)

7.71 TPWPart Class Reference

The MusicXML partwise *part* element.

`#include <TScorePartwise.h>`

Inheritance diagram for TPWPart::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **SPWMeasure** &measure)

  ! adds a measure and returns its index;

- **vvector**< **SPWMeasure** > & **measures** ()

Protected Member Functions

- **TPWPart** (const string &id)

Friends

- EXP friend **SMARTP**< **TPWPart** > **newPWPart** (string id)

7.72 TRational Class Reference

rational number representation.

`#include <TRational.h>`

Public Member Functions

- **TRational** (long int num=0, long int denom=1)
- **TRational** (const **TRational** &d)
- **TRational** (const string &str)
- long int **getNumerator** () const
- long int **getDenominator** () const
- void **setNumerator** (long int d)
- void **setDenominator** (long int d)
- void **set** (long int n, long int d)
- **TRational operator+** (const **TRational** &dur) const
- **TRational operator-** (const **TRational** &dur) const
- **TRational operator** ∗ (const **TRational** &dur) const

  Useful for notes with dots.

- **TRational operator/** (const **TRational** &dur) const
- **TRational operator** ∗ (int num) const
- **TRational operator/** (int num) const
- **TRational** & **operator+=** (const **TRational** &dur)
- **TRational** & **operator-=** (const **TRational** &dur)
- **TRational** & **operator** ∗= (const **TRational** &dur)

  Useful for notes with dots.

- **TRational** & **operator/=** (const **TRational** &dur)
- **TRational** & **operator** ∗= (long int num)
- **TRational** & **operator/=** (long int num)
- **TRational** & **operator=** (const **TRational** &dur)
- bool **operator>** (const **TRational** &dur) const
- bool **operator>=** (const **TRational** &dur) const
- bool **operator<** (const **TRational** &dur) const
- bool **operator<=** (const **TRational** &dur) const
- bool **operator==** (const **TRational** &dur) const
- bool **operator!=** (const **TRational** &dur) const
- bool **operator>** (double num) const
- bool **operator>=** (double num) const
- bool **operator<** (double num) const
- bool **operator<=** (double num) const
- bool **operator==** (double) const
- void **rationalise** ()
- **operator string** () const
- **operator double** () const
- **operator float** () const
- **operator int** () const
- virtual string **toString** () const

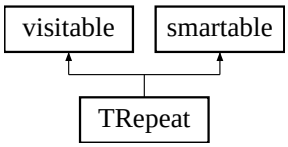
- double **toDouble** () const
- float **toFloat** () const
- int **toInt** () const

7.73 TRepeat Class Reference

Represents repeat marks.

`#include <TBarline.h>`

Inheritance diagram for TRepeat::



Public Types

- enum { **undefined** = 0, **backward**, **forward**, **last** = forward }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- int **getDirection** () const
- int **getTimes** () const
- void **setDirection** (int dir)
- void **setTimes** (int times)

Static Public Member Functions

- const string **xmldirection** (int d)
  convert an integer direction to a MusicXML string
- int **xmldirection** (const string str)
  convert an MusicXML string to an integer direction

Protected Member Functions

- **TRepeat** (int direction)

Friends

- EXP friend **SMARTP**< **TRepeat** > **newRepeat** (int direction)

7.73.1 Detailed Description

The start of the repeat has a forward direction while the end of the repeat has a backward direction. Backward repeats that are not part of an ending can use the times attribute to indicate the number of times the repeated section is played.

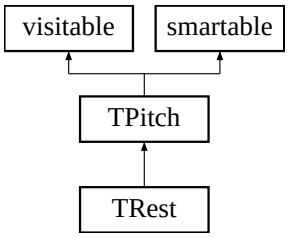
The MusicXML *repear* element is defined in barline.dtd.

7.74 TRest Class Reference

The MusicXML representation of a rest.

`#include <TPitch.h>`

Inheritance diagram for TRest::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)

Friends

- EXP friend **SMARTP**< **TRest** > **newRest** ()

7.74.1 Detailed Description

The rest element indicates notated rests or silences. Rest are usually empty, but placement on the staff can be specified using display-step and display-octave elements.

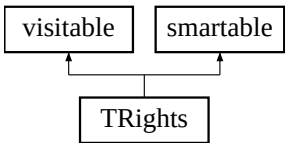
The MusicXML *rest* element is defined in note.dtd.

7.75 TRights Class Reference

Contains copyright and other intellectual property notices.

`#include <TIdentification.h>`

Inheritance diagram for TRights::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- const string & **getType** () const
- const string & **getRight** () const
- void **setType** (const string &type)
- void **setRights** (const string &rights)

Friends

- EXP friend **SMARTP**< **TRights** > **newRights** ()

7.75.1 Detailed Description

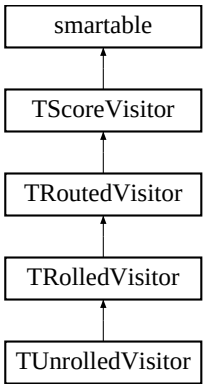
Words, music, and derivatives can have different types, so multiple rights tags with different type attributes are supported. It corresponds to the MusicXML *right* element as defined in identity.dtd.

7.76 TRolledVisitor Class Reference

A visitor that transport a **TNodeVisitor**(p. 90) along a score.

`#include <TRolledVisitor.h>`

Inheritance diagram for TRolledVisitor::



Public Member Functions

- **TRolledVisitor** (**TNodeVisitor** ∗visitor)
- virtual void **visite** (**SAccidental** &elt)
- virtual void **visite** (**SArticulationElement** &elt)
- virtual void **visite** (**SArticulations** &elt)
- virtual void **visite** (**SAttributes** &elt)
- virtual void **visite** (**SBackup** &elt)
- virtual void **visite** (**SBarline** &elt)
- virtual void **visite** (**SBeam** &elt)
- virtual void **visite** (**SBracket** &elt)
- virtual void **visite** (**SChord** &elt)
- virtual void **visite** (**SClef** &elt)
- virtual void **visite** (**SCoda** &elt)
- virtual void **visite** (**SCreator** &elt)
- virtual void **visite** (**SDashes** &elt)
- virtual void **visite** (**SDirection** &elt)
- virtual void **visite** (**SDirectionType** &elt)
- virtual void **visite** (**SDirective** &elt)
- virtual void **visite** (**SDynamic** &elt)
- virtual void **visite** (**SEncoding** &elt)
- virtual void **visite** (**SEnding** &elt)
- virtual void **visite** (**SExpression** &elt)
- virtual void **visite** (**SFermata** &elt)
- virtual void **visite** (**SForward** &elt)
- virtual void **visite** (**SGraceNote** &elt)
- virtual void **visite** (**SGraphNote** &elt)
- virtual void **visite** (**SIdentification** &elt)
- virtual void **visite** (**SKey** &elt)
- virtual void **visite** (**SMeasure** &elt)

- virtual void **visite** (**SMetronome** &elt)
- virtual void **visite** (**SMidiDevice** &elt)
- virtual void **visite** (**SMidiInstrument** &elt)
- virtual void **visite** (**SNotation** &elt)
- virtual void **visite** (**SNote** &elt)
- virtual void **visite** (**SNoteHead** &elt)
- virtual void **visite** (**SOctaveShift** &elt)
- virtual void **visite** (**SPWMeasure** &elt)
- virtual void **visite** (**SPWPart** &elt)
- virtual void **visite** (**SPart** &elt)
- virtual void **visite** (**SPartGroup** &elt)
- virtual void **visite** (**SPartList** &elt)
- virtual void **visite** (**SPitch** &elt)
- virtual void **visite** (**SRepeat** &elt)
- virtual void **visite** (**SRest** &elt)
- virtual void **visite** (**SRights** &elt)
- virtual void **visite** (**SScoreHeader** &elt)
- virtual void **visite** (**SScoreInstrument** &elt)
- virtual void **visite** (**SScorePart** &elt)
- virtual void **visite** (**SScorePartwise** &elt)
- virtual void **visite** (**SScoreTimewise** &elt)
- virtual void **visite** (**SSegno** &elt)
- virtual void **visite** (**SSlur** &elt)
- virtual void **visite** (**SSound** &elt)
- virtual void **visite** (**SStrongAccent** &elt)
- virtual void **visite** (**STWMeasure** &elt)
- virtual void **visite** (**STWPart** &elt)
- virtual void **visite** (**STie** &elt)
- virtual void **visite** (**STimeModification** &elt)
- virtual void **visite** (**STimeSign** &elt)
- virtual void **visite** (**STranspose** &elt)
- virtual void **visite** (**STuplet** &elt)
- virtual void **visite** (**STupletDesc** &elt)
- virtual void **visite** (**SUnpitched** &elt)
- virtual void **visite** (**SWavyLine** &elt)
- virtual void **visite** (**SWedge** &elt)
- virtual void **visite** (**SWords** &elt)
- virtual void **visite** (**SWork** &elt)

Protected Attributes

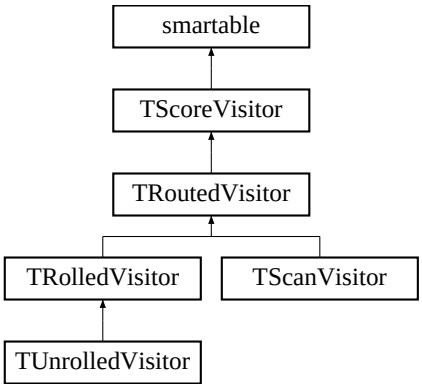
- **TNodeVisitor** ∗ **fVisitor**

7.77 TRoutedVisitor Class Reference

A visitor that implements a traversing of a score.

`#include <TRoutedVisitor.h>`

Inheritance diagram for TRoutedVisitor::



Public Member Functions

- virtual void **visite** (**SArticulationElement** &elt)
- virtual void **visite** (**SArticulations** &elt)
- virtual void **visite** (**SAttributes** &elt)
- virtual void **visite** (**SBarline** &elt)
- virtual void **visite** (**SBracket** &elt)
- virtual void **visite** (**SChord** &elt)
- virtual void **visite** (**SCoda** &elt)
- virtual void **visite** (**SDashes** &elt)
- virtual void **visite** (**SDirection** &elt)
- virtual void **visite** (**SDirectionType** &elt)
- virtual void **visite** (**SDynamic** &elt)
- virtual void **visite** (**SFermata** &elt)
- virtual void **visite** (**SGraphNote** &elt)
- virtual void **visite** (**SIdentification** &elt)
- virtual void **visite** (**SNotation** &elt)
- virtual void **visite** (**SNote** &elt)
- virtual void **visite** (**SOctaveShift** &elt)
- virtual void **visite** (**SPWMeasure** &elt)
- virtual void **visite** (**SPWPart** &elt)
- virtual void **visite** (**SPartList** &elt)
- virtual void **visite** (**SScoreHeader** &elt)
- virtual void **visite** (**SScorePart** &elt)
- virtual void **visite** (**SScorePartwise** &elt)
- virtual void **visite** (**SScoreTimewise** &elt)
- virtual void **visite** (**SSegno** &elt)
- virtual void **visite** (**SSlur** &elt)
- virtual void **visite** (**SSound** &elt)
- virtual void **visite** (**SStrongAccent** &elt)

- virtual void **visite** (**STWMeasure** &elt)
- virtual void **visite** (**STWPart** &elt)
- virtual void **visite** (**STie** &elt)
- virtual void **visite** (**STuplet** &elt)
- virtual void **visite** (**SWavyLine** &elt)
- virtual void **visite** (**SWedge** &elt)
- virtual void **visite** (**SWords** &elt)

7.77.1 Detailed Description

A **TRoutedVisitor**(p. 123) provides a path for traversing of a score tree. Each implemented *visite* method calls *accept()* for its subclasses. Order of these calls is the following:

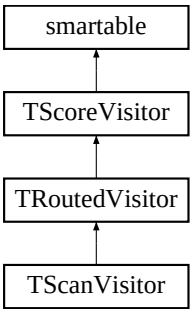
- subclasses representing attributes are called first,
- subclasses representing elements are called in the MusicXML defined order.

7.78 TScanVisitor Class Reference

A visitor that only scans a score to gather information.

`#include <TScanVisitor.h>`

Inheritance diagram for TScanVisitor::



Public Member Functions

- virtual void **visite** (**SNote** &elt)
- virtual void **visite** (**SPWPart** &elt)
- long **countParts** () const
- map< string, **SPartSummary** > & **summary** ()
- const **SPartSummary** & **getPartSummary** (string partID)
- const **SPartSummary** & **current** () const

Friends

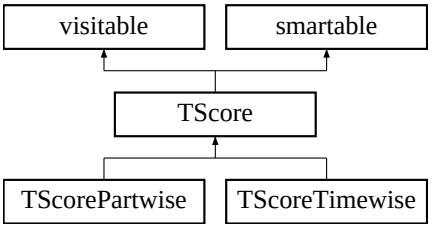
- VEXP friend **SMARTP**< **TScanVisitor** > **newScanVisitor** ()

7.79 TScore Class Reference

a MusicXML score.

`#include <TScore.h>`

Inheritance diagram for TScore::



Public Member Functions

- **Sxmlheader** & **xmlheader** ()
- **SScoreHeader** & **scoreHeader** ()

Protected Attributes

- **Sxmlheader fHeader**
- **SScoreHeader fScoreHeader**

7.79.1 Detailed Description

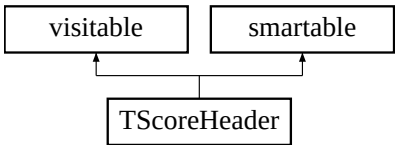
Base class for the partwise and timewise MusicXML formats.

7.80 TScoreHeader Class Reference

Contains basic score meta-data plus the part list.

`#include <TScoreHeader.h>`

Inheritance diagram for TScoreHeader::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setMovementNum** (const string &num)
- void **setMovementTitle** (const string &title)
- const string & **getMovementNum** ()
- const string & **getMovementTitle** ()
- **SWork** & **work** ()
- **SIdentification** & **identification** ()
- **SPartList** & **partList** ()

Friends

- EXP friend **SMARTP**< **TScoreHeader** > **newScoreHeader** ()

7.80.1 Detailed Description

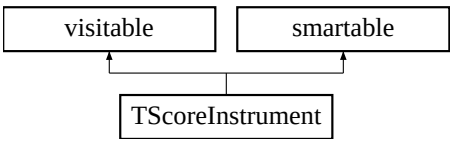
Meta-data includes work and movement name, movement-wide identification data. It corresponds to the MusicXML *score-header* entity as defined in score.dtd.

7.81 TScoreInstrument Class Reference

Represents an instrument on the score.

`#include <TScoreInstrument.h>`

Inheritance diagram for TScoreInstrument::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setID** (const string &id)

  a MusicXML required attribute used for midi-instrument assignment.

- void **setName** (const string &name)

  the instrument name.

- void **setAbbrev** (const string &abbrev)

  the optional name abbreviation.

- const string & **getID** () const
- const string & **getName** () const
- const string & **getAbbrev** () const

Protected Member Functions

- **TScoreInstrument** (string id)

Friends

- EXP friend **SMARTP**< **TScoreInstrument** > **newScoreInstrument** (string id)

7.81.1 Detailed Description

The score-instrument element allows for multiple instruments per score-part. As with the score-part element, each score-instrument has a required ID attribute, a name, and an optional abbreviation. A score-instrument element is also required if the score specifies MIDI channels, banks, or programs. An initial midi-instrument assignment can also be made here. MusicXML software should be able to automatically assign reasonable channels and instruments without these elements in simple cases, such as where part names match General MIDI instrument names.

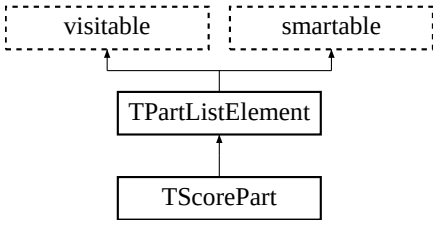
The MusicXML *score-instrument* is defined in score.dtd.

7.82 TScorePart Class Reference

Identifies a musical part in a score.

`#include <TScorePart.h>`

Inheritance diagram for TScorePart::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- const string & **getID** () const
- const string & **getPartName** () const
- const string & **getPartAbbrev** () const
- **vvector**< **SScoreInstrument** > & **scoreInstruments** ()
- **vvector**< **SMidiInstrument** > & **midiInstruments** ()
- void **setID** (const string &id)
- void **setPartName** (const string &name)
- void **setPartAbbrev** (const string &abbrev)
- long **add** (const **SScoreInstrument** &instr)
- long **add** (const **SMidiInstrument** &instr)
- **SIdentification** & **identification** ()
- **SMidiDevice** & **mididevice** ()

Protected Member Functions

- **TScorePart** (string id)

Friends

- EXP friend **SMARTP**< **TScorePart** > **newScorePart** (string id)

7.82.1 Detailed Description

Each part has an ID that is used later within the musical data. Since parts may be encoded separately and combined later, identification elements are present at both the score and score-part levels.

Each MusicXML part corresponds to a track in a Standard MIDI Format 1 file. The score-instrument elements are used when there are multiple instruments per track. The midi-device element is used to make a MIDI device or port assignment for the given track. Initial midi-instrument assignments may be made here as well.

TScorePart(p. 129) corresponds to the MusicXML *score-part* element as defined in score.dtd.

Todo

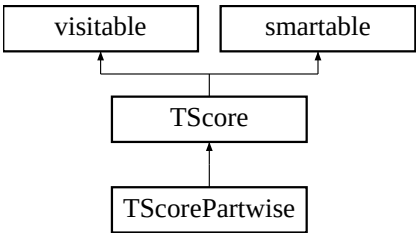
Support of the MusicXML *group* element (design will change soon).

7.83 TScorePartwise Class Reference

a MusicXML partwise score.

`#include <TScorePartwise.h>`

Inheritance diagram for TScorePartwise::



Public Member Functions

- long **add** (const **SPWPart** &part)

  ! adds a part and returns its index;

- **vvector**< **SPWPart** > & **partList** ()
- virtual void **accept** (**TScoreVisitor** &visitor)

Friends

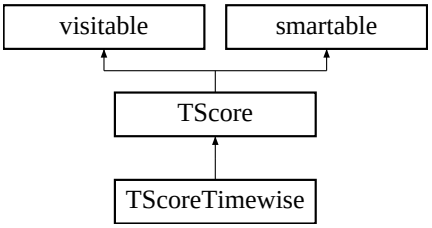
- EXP friend **SMARTP**< **TScorePartwise** > **newScorePartwise** ()

7.84 TScoreTimewise Class Reference

a MusicXML timewise score.

`#include <TScoreTimewise.h>`

Inheritance diagram for TScoreTimewise::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **STWMeasure** &part)
- **vvector**< **STWMeasure** > & **measureList** ()

Friends

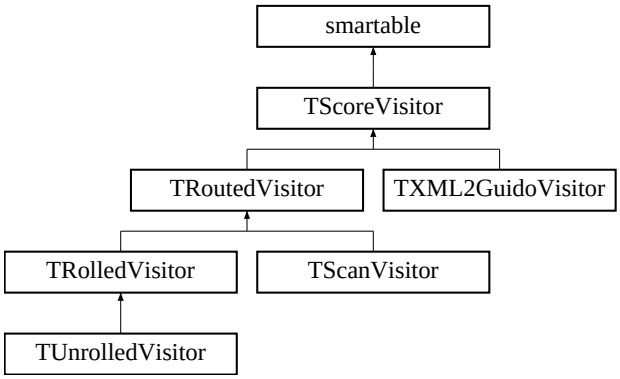
- EXP friend **SMARTP**< **TScoreTimewise** > **newScoreTimewise** ()

7.85 TScoreVisitor Class Reference

A Visitor of MusicXML scores.

`#include <TScoreVisitor.h>`

Inheritance diagram for TScoreVisitor::



Public Member Functions

- virtual void **visite** (**SAccidental** &elt)
- virtual void **visite** (**SArticulationElement** &elt)
- virtual void **visite** (**SArticulations** &elt)
- virtual void **visite** (**SAttributes** &elt)
- virtual void **visite** (**SBackup** &elt)
- virtual void **visite** (**SBarline** &elt)
- virtual void **visite** (**SBeam** &elt)
- virtual void **visite** (**SBracket** &elt)
- virtual void **visite** (**SChord** &elt)
- virtual void **visite** (**SClef** &elt)
- virtual void **visite** (**SCoda** &elt)
- virtual void **visite** (**SCreator** &elt)
- virtual void **visite** (**SDashes** &elt)
- virtual void **visite** (**SDirection** &elt)
- virtual void **visite** (**SDirectionType** &elt)
- virtual void **visite** (**SDirective** &elt)
- virtual void **visite** (**SDynamic** &elt)
- virtual void **visite** (**SEncoding** &elt)
- virtual void **visite** (**SEnding** &elt)
- virtual void **visite** (**SExpression** &elt)
- virtual void **visite** (**SFermata** &elt)
- virtual void **visite** (**SForward** &elt)
- virtual void **visite** (**SGraceNote** &elt)
- virtual void **visite** (**SGraphNote** &elt)
- virtual void **visite** (**SIdentification** &elt)
- virtual void **visite** (**SKey** &elt)
- virtual void **visite** (**SMeasure** &elt)
- virtual void **visite** (**SMetronome** &elt)

- virtual void **visite** (**SMidiDevice** &elt)
- virtual void **visite** (**SMidiInstrument** &elt)
- virtual void **visite** (**SNotation** &elt)
- virtual void **visite** (**SNote** &elt)
- virtual void **visite** (**SNoteHead** &elt)
- virtual void **visite** (**SOctaveShift** &elt)
- virtual void **visite** (**SPWMeasure** &elt)
- virtual void **visite** (**SPWPart** &elt)
- virtual void **visite** (**SPart** &elt)
- virtual void **visite** (**SPartGroup** &elt)
- virtual void **visite** (**SPartList** &elt)
- virtual void **visite** (**SPitch** &elt)
- virtual void **visite** (**SRepeat** &elt)
- virtual void **visite** (**SRest** &elt)
- virtual void **visite** (**SRights** &elt)
- virtual void **visite** (**SScoreHeader** &elt)
- virtual void **visite** (**SScoreInstrument** &elt)
- virtual void **visite** (**SScorePart** &elt)
- virtual void **visite** (**SScorePartwise** &elt)
- virtual void **visite** (**SScoreTimewise** &elt)
- virtual void **visite** (**SSegno** &elt)
- virtual void **visite** (**SSlur** &elt)
- virtual void **visite** (**SSound** &elt)
- virtual void **visite** (**SStrongAccent** &elt)
- virtual void **visite** (**STWMeasure** &elt)
- virtual void **visite** (**STWPart** &elt)
- virtual void **visite** (**STie** &elt)
- virtual void **visite** (**STimeModification** &elt)
- virtual void **visite** (**STimeSign** &elt)
- virtual void **visite** (**STranspose** &elt)
- virtual void **visite** (**STuplet** &elt)
- virtual void **visite** (**STupletDesc** &elt)
- virtual void **visite** (**SUnpitched** &elt)
- virtual void **visite** (**SWavyLine** &elt)
- virtual void **visite** (**SWedge** &elt)
- virtual void **visite** (**SWords** &elt)
- virtual void **visite** (**SWork** &elt)
- virtual void **visite** (**TOrientation** *elt)
- virtual void **visite** (**TPlacement** *elt)
- virtual void **visite** (**TPosition** *elt)

7.85.1 Detailed Description

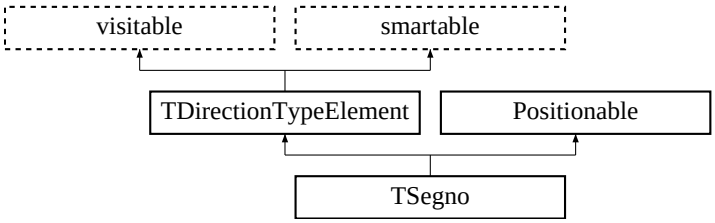
TScoreVisitor(p. 133) implements the concept of visitors as defined in *Design Patterns - Elements of Reusable Object-Oriented Software* E. Gamma, R.Helm, R.Johnson, J.Vlissides - Addison-Wesley 1999

7.86 TSegno Class Reference

Represents the MusicXML *segno* element.

`#include <common.h>`

Inheritance diagram for TSegno::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)

Friends

- EXP friend **SMARTP**< **TSegno** > **newSegno** ()

7.86.1 Detailed Description

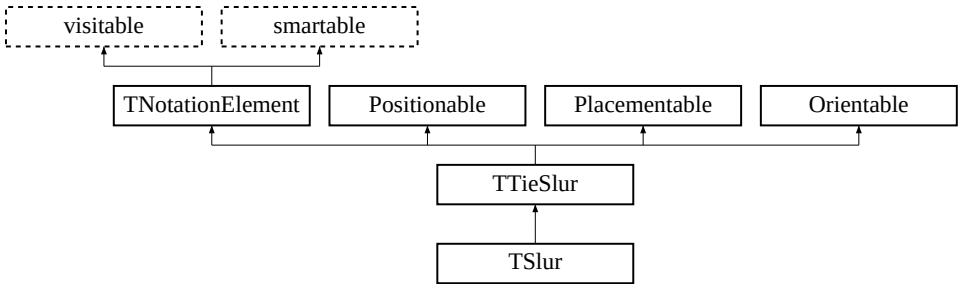
TSegno(p. 135) is a visual indicator only. It is defined in common.dtd.

7.87 TSlur Class Reference

Slur representation.

`#include <TNotation.h>`

Inheritance diagram for TSlur::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)

Protected Member Functions

- **TSlur** (StartStop::type type)

Friends

- EXP friend **SMARTP**< **TSlur** > **newSlur** (StartStop::type type)

7.87.1 Detailed Description

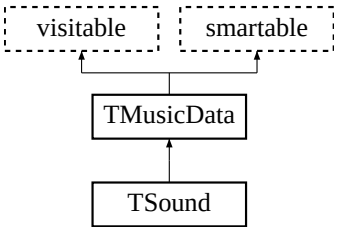
MusicXML *slur* are defined in note.dtd.

7.88 TSound Class Reference

Represents the MusicXML *sound* element.

`#include <TSound.h>`

Inheritance diagram for TSound::



Public Types

- enum { **yes** = -2, **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setTempo** (long tempo)

  Tempo is expressed in quarter notes per minute.

- void **setDynamics** (long dyn)

  Dynamics (or MIDI velocity) are expressed as a percentage of the default forte value (90 for MIDI).

- void **setDacapo** (YesNo::type yn)

  Dacapo indicates to go back to the beginning of the movement.

- void **setSegno** (long segno)

  a segno mark. The value can be used to distinguish multiple jumps.

- void **setDalsegno** (long ds)

  dalsegno is used for backward jump to a segno sign. The value can be used to distinguish multiple jumps.

- void **setCoda** (long coda)

  a coda mark. The value can be used to distinguish multiple jumps.

- void **setTocoda** (long tc)

  tocoda is used for forward jump to a coda sign. The value can be used to distinguish multiple jumps.

- void **setDivisions** (long div)

  to be used with segno or coda attributes to avoid computation of divisions.

- void **setForwardRepeat** (YesNo::type yn)

  Forward-repeat is used when a forward repeat sign is implied, and usually follows a bar line.

- void **setFine** (long fine)

  The fine attribute follows the final note or rest in a movement with a da capo direction. It may take the special yes *value.*

- void **setPizzicato** (YesNo::type pizz)

  Pizzicato in a sound element effects all following notes. true *indicates pizzicato,* false *indicates arco.*

- void **setMidiChannel** (long chan)

  MIDI channel indication allow for detailed control of MIDI assignment.

- void **setMidiIntrument** (long instr)

  MIDI instrument indication allow for detailed control of MIDI assignment.

- long **add** (const **SMidiInstrument** &instr)
- long **getTempo** () const
- long **getDynamics** () const
- YesNo::type **getDacapo** () const
- long **getSegno** () const
- long **getDalsegno** () const
- long **getCoda** () const
- long **getTocoda** () const
- long **getDivisions** () const
- YesNo::type **getForwardRepeat** () const
- long **getFine** () const
- YesNo::type **getPizzicato** () const
- long **getMidiChannel** () const
- long **getMidiIntrument** () const
- **vvector**< **SMidiInstrument** > & **instrumentList** ()

Protected Member Functions

- void **init** ()

Friends

- EXP friend **SMARTP**< **TSound** > **newSound** ()

7.88.1 Detailed Description

The sound element contains general playback parameters, based on MuseData's C0 sound suggestions. They can stand alone within a part/measure, or be a component element within a direction.

Segno and dalsegno are used for backwards jumps to a segno sign; coda and tocoda are used for forward jumps to a coda sign. If there are multiple jumps, the value of these parameters can be used to name and distinguish them. If segno or coda is used, the divisions attribute can also be

used to indicate the number of divisions per quarter note. Otherwise sound and MIDI generating programs may have to recompute this.

A dalsegno or dacapo attribute indicates that the jump should occur the first time through; a tocoda attribute indicates that the jump should occur the second time through.

The fine attribute follows the final note or rest in a movement with a da capo direction. If numeric, the value represents the actual duration of the final note or rest, which can be ambiguous in written notation and different among parts and voices. The value may also be "yes" to indicate no change to the final duration.

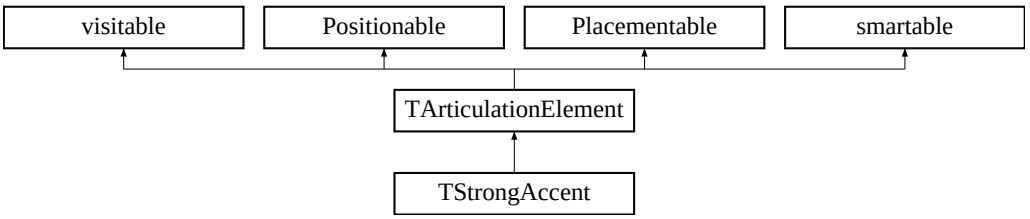
The MusicXML *sound* element is defined in direction.dtd. All its attributes are implied.

7.89 TStrongAccent Class Reference

a strong accent representation.

`#include <TNotation.h>`

Inheritance diagram for TStrongAccent::



Public Types

- enum { **undefined**, **up** = 1, **down**, **last** = down }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- int **getType** () const
- void **setType** (int type)

Static Public Member Functions

- const string **xmltype** (int d)

  convert a numeric accent type to a MusicXML string

- int **xmltype** (const string str)

  convert an MusicXML string to a numeric accent type

Friends

- EXP friend **SMARTP**< **TStrongAccent** > **newStrongAccent** ()

7.89.1 Detailed Description

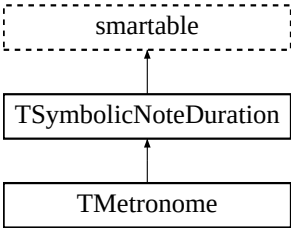
a strong accent is an articulation but have an additional type attributes.

7.90 TSymbolicNoteDuration Class Reference

the base class for symbolic note duration

`#include <TDirection.h>`

Inheritance diagram for TSymbolicNoteDuration::



Public Member Functions

- void **setBeat** (NoteType::type beatUnit)
- void **setDots** (unsigned dots)
- NoteType::type **getBeat** () const
- unsigned **getDots** () const

Friends

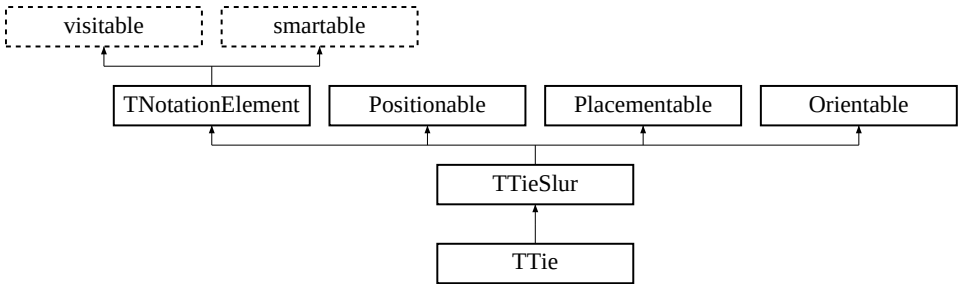
- EXP friend **SMARTP< TSymbolicNoteDuration > newSymbolicNoteDuration** ()

7.91 TTie Class Reference

Ties representation.

`#include <TNotation.h>`

Inheritance diagram for TTie::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)

Protected Member Functions

- **TTie** (StartStop::type type)

Friends

- EXP friend **SMARTP**< **TTie** > **newTie** (StartStop::type type)

7.91.1 Detailed Description

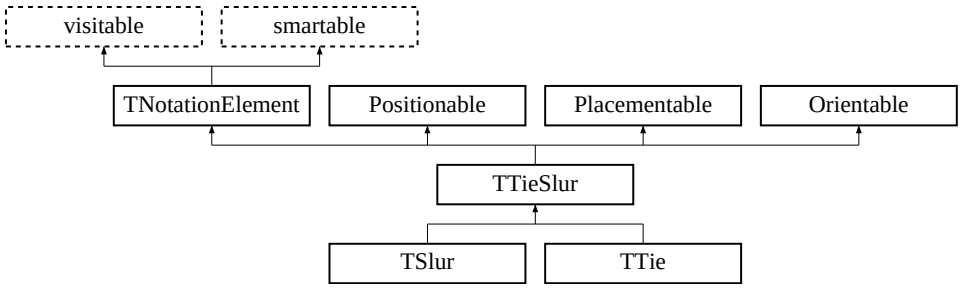
MusicXML *ties* are defined in note.dtd.

7.92 TTieSlur Class Reference

base class for ties and slurs

`#include <TNotation.h>`

Inheritance diagram for TTieSlur::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)=0
- StartStop::type **getType** () const
- int **getNumber** () const
- LineType::type **getLineType** () const
- void **setType** (StartStop::type type)
- void **setNumber** (int num)
- void **setLineType** (LineType::type type)
- const string & **getElement** () const

Protected Member Functions

- **TTieSlur** (string elt, StartStop::type type)

7.92.1 Detailed Description

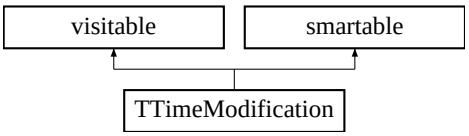
Ties and slurs share common attributes and are represented by a single object derived into **TTie**(p. 142) and **TSlur**(p. 136).

7.93 TTimeModification Class Reference

Represents tuplets and other durational changes.

`#include <TTimeModification.h>`

Inheritance diagram for TTimeModification::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setActualNotes** (long v)
- void **setNormalNotes** (long v)
- void **setNormalType** (NoteType::type v)
- void **setNormalDots** (long v)
- long **getActualNotes** () const
- long **getNormalNotes** () const
- NoteType::type **getNormalType** () const
- long **getNormalDots** () const

Friends

- EXP friend **SMARTP**< **TTimeModification** > **newTimeModification** ()

7.93.1 Detailed Description

The actual-notes element describes how many notes are played in the time usually occupied by the number of normal-notes. If the normal-notes type is different than the current note type (e.g., a quarter note within an eighth note triplet), then the normal-notes type (e.g. eighth) is specified in the normal-type and normal-dot elements.

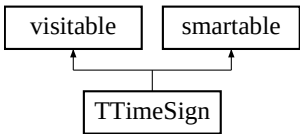
TTimeModification(p. 144) corresponds to the MusicXML *time-modification* as defined in note.dtd.

7.94 TTimeSign Class Reference

Represents time signatures.

`#include <TTimeSign.h>`

Inheritance diagram for TTimeSign::



Public Types

- enum {
  undefined = -1, **groupbeats** = -1, **common** = 1, **cut**,
  single_number, **normal**, **last** = normal }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setSymbol** (int symbol)
- int **getSymbol** () const
- long **add** (long beat, long type)
  beat type must be `groupbeat` *for the first elements of a composite like 3+2/8*
- bool **measured** () const
  senza measures are implicitely denoted by an empty beat list
- **TRational rational** ()
  returns the time signature as a rational number (senza measure is -1/1)
- vector< long > & **getBeats** ()
- vector< long > & **getBeatTypes** ()

Static Public Member Functions

- const string **xmlsymbol** (int d)
  convert an integer symbol to a MusicXML string
- int **xmlsymbol** (const string str)
  convert an MusicXML string to an integer symbol

Protected Member Functions

- **TTimeSign** ()
  empty constructor denotes unmeasured time

Friends

- EXP friend **SMARTP< TTimeSign > newTimeSign** ()

7.94.1 Detailed Description

Time signatures are represented by two elements. The beats element indicates the number of beats, as found in the numerator of a time signature. The beat-type element indicates the beat unit, as found in the denominator of a time signature. The symbol attribute is used to indicate another notation beyond a fraction: the common and cut time symbols, as well as a single number with an implied denominator. Normal (a fraction) is the implied symbol type if none is specified. Multiple pairs of beat and beat-type elements are used for composite time signatures with multiple denominators, such as 2/4 + 3/8. A composite such as 3+2/8 requires only one beat/beat-type pair. A senza-misura element explicitly indicates that no time signature is present.

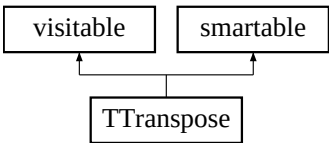
The MusicXML *time* element is defined in attributes.dtd.

7.95 TTranspose Class Reference

used for transposing instruments.

`#include <TTranspose.h>`

Inheritance diagram for TTranspose::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setDiatonic** (long diat)
- void **setChromatic** (long chrom)
- void **setOctaveChge** (long oct)
- void **setDouble** (long dbl)
- long **getDiatonic** () const
- long **getChromatic** () const
- long **getOctaveChge** () const
- long **getDouble** () const

Friends

- EXP friend **SMARTP**< **TTranspose** > **newTranspose** ()

7.95.1 Detailed Description

If the part is being encoded for a transposing instrument in written vs. concert pitch, the transposition must be encoded in the transpose element. The transposition is represented by chromatic steps (required) and three optional elements: diatonic pitch steps, octave changes, and doubling an octave down. The chromatic and octave-change elements are numeric values added to the encoded pitch data to create the sounding pitch. The diatonic element is also numeric and allows for correct spelling of enharmonic transpositions.

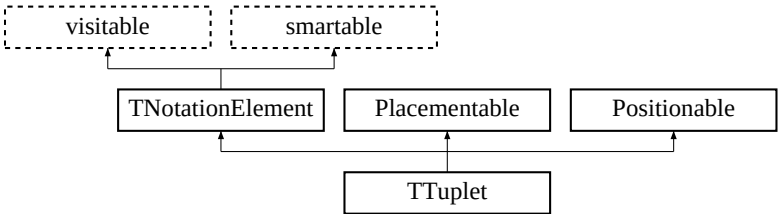
The MusicXML *transpose* element is defined in attributes.dtd.

7.96 TTuplet Class Reference

Graphical representation of a tuplet.

`#include <TNotation.h>`

Inheritance diagram for TTuplet::



Public Types

- enum {
  undefined = -1, **actual** = 1, **both**, **none**,
  last = none }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setType** (StartStop::type type)
  the required type attribute
- void **setNumber** (int num)
  optional number for nested tuplets
- void **setBracket** (YesNo::type bracket)
  optional bracket attribute
- void **setShowNum** (int sn)
  optional display control
- void **setShowType** (int st)
  optional display control
- StartStop::type **getType** () const
- int **getNumber** () const
- YesNo::type **getBracket** () const
- int **getShowNum** () const
- int **getShowType** () const
- **STupletDesc** & **actualDesc** ()
- **STupletDesc** & **normalDesc** ()

Static Public Member Functions

- const string **xmlshow** (int d)

  convert a numeric show type to a MusicXML string

- int **xmlshow** (const string str)

  convert an MusicXML string to a numeric show type

Protected Member Functions

- **TTuplet** (StartStop::type type)

Friends

- EXP friend **SMARTP**< **TTuplet** > **newTuplet** (StartStop::type type)

7.96.1 Detailed Description

A tuplet element is present when a tuplet is to be displayed graphically, in addition to the sound data provided by the time-modification elements. The number attribute is used to distinguish nested tuplets. The bracket attribute is used to indicate the presence of a bracket. If unspecified, the results are implementation- dependent.

The tuplet-actual and tuplet-normal elements provide optional full control over tuplet specifications. Each allows the number and note type (including dots) describing a single tuplet. If any of these elements are absent, their values are based on the time-modification element.

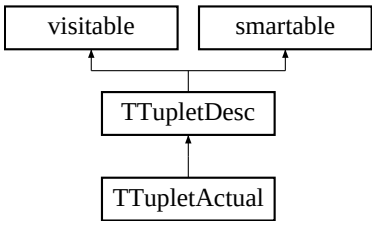
The show-number attribute is used to display either the number of actual notes, the number of both actual and normal notes, or neither. It is actual by default. The show-type attribute is used to display either the actual type, both the actual and normal types, or neither. It is none by default.

7.97 TTupletActual Class Reference

an actual tuplet description.

`#include <TNotation.h>`

Inheritance diagram for TTupletActual::



Friends

- EXP friend **STupletDesc newTupletActual** ()

7.97.1 Detailed Description

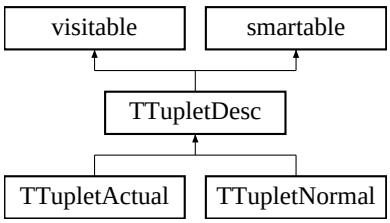
It correspond to the *tuplet-actual* MusicXML elements as defined in note.dtd.

7.98 TTupletDesc Class Reference

a tuplet description.

`#include <TNotation.h>`

Inheritance diagram for TTupletDesc::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setNumber** (int num)
- void **setType** (int type)
- void **setDots** (int dots)
- int **getNumber** () const
- int **getType** () const
- int **getDots** () const
- const string **getName** () const

Protected Member Functions

- **TTupletDesc** (string suffix)

7.98.1 Detailed Description

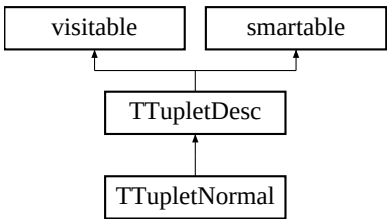
It correspond to the *tuplet-actual* and *tuplet-normal* MusicXML elements as defined in note.dtd.

7.99 TTupletNormal Class Reference

a normal tuplet description.

`#include <TNotation.h>`

Inheritance diagram for TTupletNormal::



Friends

- EXP friend **STupletDesc newTupletNormal** ()

7.99.1 Detailed Description

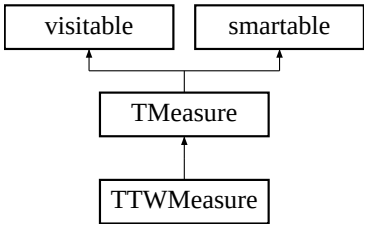
It correspond to the *tuplet-normal* MusicXML elements as defined in note.dtd.

7.100 TTWMeasure Class Reference

The MusicXML timewise *measure* element.

`#include <TScoreTimewise.h>`

Inheritance diagram for TTWMeasure::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **SMARTP**< **TTWPart** > &part)
- **vvector**< **SMARTP**< **TTWPart** > > & **parts** ()

Protected Member Functions

- **TTWMeasure** (string num)
- **TTWMeasure** (long num)

Friends

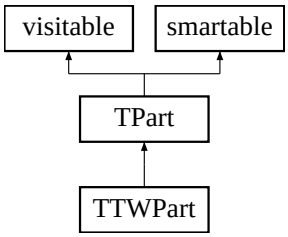
- EXP friend **SMARTP**< **TTWMeasure** > **newTWMeasure** (string num)
- EXP friend **SMARTP**< **TTWMeasure** > **newTWMeasure** (long num)

7.101 TTWPart Class Reference

The MusicXML timewise *part* element.

`#include <TScoreTimewise.h>`

Inheritance diagram for TTWPart::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- long **add** (const **SMusicData** &data)
- **vvector**< **SMusicData** > & **data** ()

Protected Member Functions

- **TTWPart** (string id)

Friends

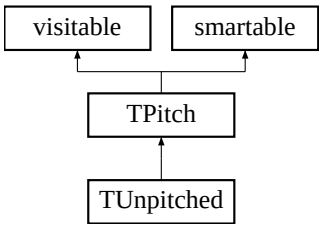
- EXP friend **SMARTP**< **TTWPart** > **newTWPart** (string id)

7.102 TUnpitched Class Reference

The MusicXML representation of an unpitched note.

`#include <TPitch.h>`

Inheritance diagram for TUnpitched::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)

Friends

- EXP friend **SMARTP**< **TUnpitched** > **newUnpitched** ()

7.102.1 Detailed Description

The unpitched element indicates musical elements that are notated on the staff but lack definite pitch, such as unpitched percussion and speaking voice. Like notes, it uses step and octave elements to indicate placement on the staff, following the current clef. If percussion clef is used, the display-step and display-octave are interpreted as if in treble clef, with a G in octave 4 on line 2. If not present, the note is placed on the middle line of the staff, generally used for one-line staffs.

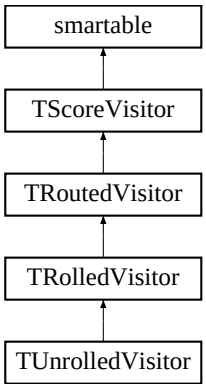
The MusicXML *unpitched* element is defined in note.dtd.

7.103 TUnrolledVisitor Class Reference

A **TUnrolledVisitor**(p. 156) "unroll" the sequence by interpreting **repeat**, **ending**, **coda** and **segno** signs.

`#include <TUnrolledVisitor.h>`

Inheritance diagram for TUnrolledVisitor::



Public Member Functions

- **TUnrolledVisitor** (**TNodeVisitor** *visitor)
- virtual void **visite** (**SPWMeasure** &elt)
- virtual void **visite** (**SPWPart** &elt)
- virtual void **visite** (**SSound** &elt)
- const list< TSection > & **GetSectionList** ()

7.103.1 Detailed Description

Todo

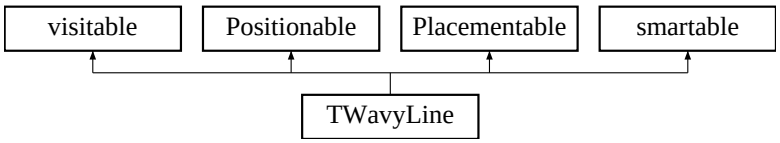
Management of multiple jump and sound **ForwardRepeat** attribute.

7.104 TWavyLine Class Reference

Represents a MusicXML *wavy-line* element.

`#include <common.h>`

Inheritance diagram for TWavyLine::



Public Types

- enum { **undefined** = -1 }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setType** (StartStop::type t)
- StartStop::type **getType** () const
- void **setNumber** (int num)
- int **getNumber** () const

Protected Member Functions

- **TWavyLine** (StartStop::type type)

Friends

- EXP friend **SMARTP**< **TWavyLine** > **newWavyLine** (StartStop::type t)

7.104.1 Detailed Description

Wavy lines are one way to indicate trills; when used with a measure element, they should always have type="continue" set.

The MusicXML *wavy-line* element is defined in common.dtd

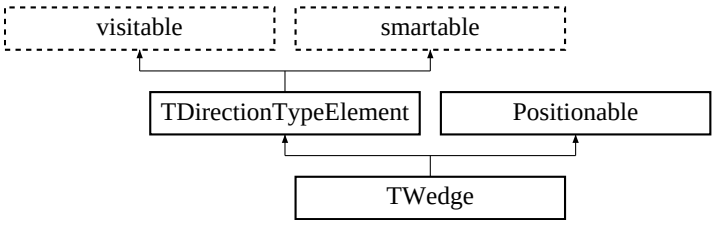
Todo
support of *trill-sound*

7.105 TWedge Class Reference

the MusicXML *wedge* element.

`#include <TDirection.h>`

Inheritance diagram for TWedge::



Public Types

- enum {
  undefined = -1, **crescendo** = 1, **diminuendo**, **stop**,
  last = stop }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setType** (int type)
- void **setNumber** (int num)
- void **setSpread** (int sp)
- int **getType** () const
- int **getNumber** () const
- int **getSpread** () const

Static Public Member Functions

- const string **xmltype** (int d)
  convert an numeric wedge type to a MusicXML string
- int **xmltype** (const string str)
  convert an MusicXML string to a numeric wedge type

Protected Member Functions

- **TWedge** (int type)

Friends

- EXP friend **SMARTP**< **TWedge** > **newWedge** (int type)

7.105.1 Detailed Description

Wedge spread is measured in tenths of staff line space. The type is crescendo for the start of a wedge that is closed at the left side, and diminuendo for the start of a wedge that is closed on the right side. Spread values at the start of a crescendo wedge or end of a diminuendo wedge are ignored.

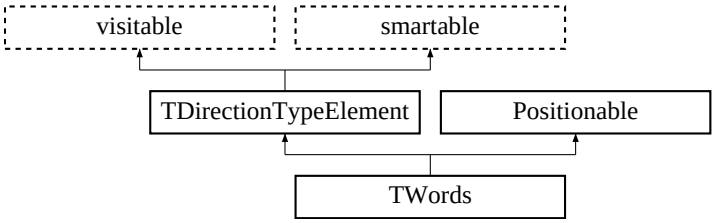
The MusicXML *wedge* element is defined in direction.dtd.

7.106 TWords Class Reference

The MusicXml *words* element.

`#include <TDirection.h>`

Inheritance diagram for TWords::



Public Types

- enum {
  undefined = -1, **left** = 1, **center**, **right**,
  last = right }

Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setJustify** (int just)
- void **setLang** (const string &lang)
- void **setValue** (const string &value)
- int **getJustify** () const
- const string & **getLang** () const
- const string & **getValue** () const

Static Public Member Functions

- const string **xmljustify** (int d)

  convert an numeric justification to a MusicXML string

- int **xmljustify** (const string str)

  convert an MusicXML string to a numeric justification

Friends

- EXP friend **SMARTP**< **TWords** > **newWords** ()

7.106.1 Detailed Description

Left justification is assumed if not specified. Language is Italian ("it") by default. Defined in direction.dtd.

Todo

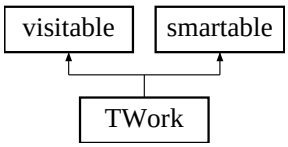
font support to be implemented

7.107 TWork Class Reference

Represents works and movements.

`#include <TWork.h>`

Inheritance diagram for TWork::



Public Member Functions

- virtual void **accept** (**TScoreVisitor** &visitor)
- void **setTitle** (const string &title)

  the optionnal work title

- void **setNum** (const string &num)

  the optionnal work number

- const string & **getTitle** () const
- const string & **getNum** () const

Friends

- EXP friend **SMARTP**< **TWork** > **newWork** ()

7.107.1 Detailed Description

Works and movements are optionally identified by number and title as in MuseData. The work element also may indicate a link to the opus document that composes multiple movements into a collection.

Todo

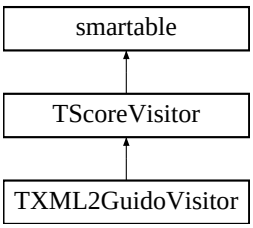
opus support is to be implemented

7.108 TXML2GuidoVisitor Class Reference

A score visitor to produce a generic Guido representation.

`#include <TXML2GuidoVisitor.h>`

Inheritance diagram for TXML2GuidoVisitor::



Public Member Functions

- **TXML2GuidoVisitor** (bool comments=true, bool checkStem=true, bool bar=false)
- void **visite** (**SAccidental** &elt)
- void **visite** (**SArticulationElement** &elt)
- void **visite** (**SArticulations** &elt)
- void **visite** (**SAttributes** &elt)
- void **visite** (**SBackup** &elt)
- void **visite** (**SBarline** &elt)
- void **visite** (**SBeam** &elt)
- void **visite** (**SBracket** &elt)
- void **visite** (**SChord** &elt)
- void **visite** (**SClef** &elt)
- void **visite** (**SCoda** &elt)
- void **visite** (**SCreator** &elt)
- void **visite** (**SDirection** &elt)
- void **visite** (**SDirectionType** &elt)
- void **visite** (**SDirective** &elt)
- void **visite** (**SDynamic** &elt)
- void **visite** (**SEncoding** &elt)
- void **visite** (**SEnding** &elt)
- void **visite** (**SFermata** &elt)
- void **visite** (**SForward** &elt)
- void **visite** (**SGraceNote** &elt)
- void **visite** (**SGraphNote** &elt)
- void **visite** (**SIdentification** &elt)
- void **visite** (**SKey** &elt)
- void **visite** (**SMetronome** &elt)
- void **visite** (**SMidiInstrument** &elt)
- void **visite** (**SNotation** &elt)
- void **visite** (**SNote** &elt)
- void **visite** (**SNoteHead** &elt)
- void **visite** (**SOctaveShift** &elt)
- void **visite** (**SPWMeasure** &elt)

- void **visite** (**SPWPart** &elt)
- void **visite** (**SPartGroup** &elt)
- void **visite** (**SPartList** &elt)
- void **visite** (**SPitch** &elt)
- void **visite** (**SRepeat** &elt)
- void **visite** (**SRest** &elt)
- void **visite** (**SScoreInstrument** &elt)
- void **visite** (**SScorePart** &elt)
- void **visite** (**SScorePartwise** &elt)
- void **visite** (**SSegno** &elt)
- void **visite** (**SSlur** &elt)
- void **visite** (**SSound** &elt)
- void **visite** (**SStrongAccent** &elt)
- void **visite** (**STie** &elt)
- void **visite** (**STimeModification** &elt)
- void **visite** (**STimeSign** &elt)
- void **visite** (**STranspose** &elt)
- void **visite** (**STuplet** &elt)
- void **visite** (**STupletDesc** &elt)
- void **visite** (**SUnpitched** &elt)
- void **visite** (**SWavyLine** &elt)
- void **visite** (**SWedge** &elt)
- void **visite** (**SWords** &elt)
- **Sguidoelement** & **current** ()

Protected Member Functions

- void **add** (**Sguidoelement** &elt)
- void **push** (**Sguidoelement** &elt)
- void **pop** ()

Protected Attributes

- long **fCurrentVoice**
- long **fCurrentDivision**
- bool **fMeasureEmpty**

Friends

- VEXP friend **SMARTP**< **TXML2GuidoVisitor** > **newXML2GuidoVisitor** (bool comments=true, bool checkStem=true, bool bar=false)

7.108.1 Member Function Documentation

7.108.1.1 void visite (STimeSign & *elt*) [virtual]

Todo

check the way unmeasured time is handled
check the symbol translation and handle the single number symbol

Reimplemented from **TScoreVisitor** (p. 133).

7.108.1.2 void visite (SScoreInstrument & *elt*) [virtual]

Todo
support of the MIDI prog change

Reimplemented from **TScoreVisitor** (p. 133).

7.108.1.3 void visite (SMetronome & *elt*) [virtual]

Todo
complete translation of TMetronome

Reimplemented from **TScoreVisitor** (p. 133).

7.108.1.4 void visite (SCreator & *elt*) [virtual]

Todo
handling the different creator types as defined in MusicXML. There is actually no corresponding tag apart the `composer` tag in guido.

Reimplemented from **TScoreVisitor** (p. 133).

7.108.1.5 void visite (SClef & *elt*) [virtual]

Todo
translation of none and TAB clefs

Reimplemented from **TScoreVisitor** (p. 133).

7.108.1.6 void visite (SBarline & *elt*) [virtual]

visit of a barline may add several guido elements: \bar, \fermata, \repeatBegin \repeatEnd ...

Reimplemented from **TScoreVisitor** (p. 133).

7.108.2 Friends And Related Function Documentation

7.108.2.1 VEXP friend SMARTP<TXML2GuidoVisitor> newXML2GuidoVisitor (bool *comments* = true, bool *checkStem* = true, bool *bar* = false) [friend]

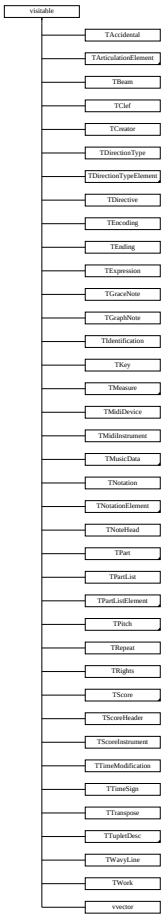
Todo
support of IDs for tags other than slurs

7.109 visitable Class Reference

interface for visitable objects.

`#include <TScoreVisitor.h>`

Inheritance diagram for visitable::



Public Member Functions

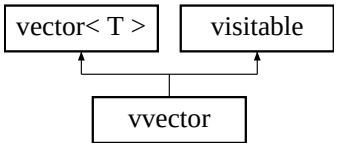
- virtual void **accept** (**TScoreVisitor** &visitor)=0

7.110 vvector Class Template Reference

a visitable vector

`#include <TScoreVisitor.h>`

Inheritance diagram for vvector::



Public Member Functions

- void **accept** (**TScoreVisitor** &visitor)

7.110.1 Detailed Description

template<typename T> class MusicXML::vvector< T >

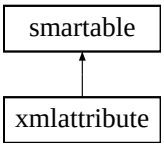
relay the accept method to all the vector elements.

7.111 xmlattribute Class Reference

A generic xml attribute representation.

`#include <xml.h>`

Inheritance diagram for xmlattribute::



Public Member Functions

- void **setName** (const string &name)
- void **setValue** (const string &value)
- void **setValue** (long value)
- const string & **getName** () const
- const string & **getValue** () const
- void **print** (ostream &os) const

Protected Member Functions

- **xmlattribute** (string name, string value)
- **xmlattribute** (string name, long value)

Friends

- EXP friend **SMARTP**< **xmlattribute** > **new_xmlattribute** (string name, string value)
- EXP friend **SMARTP**< **xmlattribute** > **new_xmlattribute** (string name, long value)

7.111.1 Detailed Description

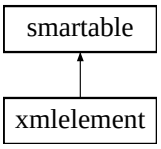
An attribute is represented by its name and its value.

7.112 xmlelement Class Reference

A generic xml element representation.

`#include <xml.h>`

Inheritance diagram for xmlelement::



Public Member Functions

- void **setValue** (unsigned long value)
- void **setValue** (long value)
- void **setValue** (int value)
- void **setValue** (float value)
- void **setValue** (const string &value)
- void **setName** (const string &name)
- const string & **getName** () const
- const string & **getValue** () const
- long **add** (const **Sxmlattribute** &attr)
- long **add** (const **Sxmlelement** &elt)
- const vector< **Sxmlelement** > & **elements** () const
- const vector< **Sxmlattribute** > & **attributes** () const
- bool **empty** () const
- void **print** (ostream &os) const

Protected Member Functions

- **xmlelement** (string name)
- **xmlelement** (string name, unsigned long value)
- **xmlelement** (string name, long value)
- **xmlelement** (string name, int value)
- **xmlelement** (string name, float value)
- **xmlelement** (string name, string value)

Friends

- EXP friend **SMARTP**< **xmlelement** > **new_xmlelement** (string name)
- EXP friend **SMARTP**< **xmlelement** > **new_xmlelement** (string name, unsigned long val)
- EXP friend **SMARTP**< **xmlelement** > **new_xmlelement** (string name, long val)
- EXP friend **SMARTP**< **xmlelement** > **new_xmlelement** (string name, int val)
- EXP friend **SMARTP**< **xmlelement** > **new_xmlelement** (string name, float val)
- EXP friend **SMARTP**< **xmlelement** > **new_xmlelement** (string name, string val)

7.112.1 Detailed Description

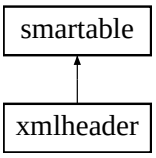
An element is represented by its name, its value, the lists of its attributes and its enclosed elements. Attributes and elements must be added in the required order.

7.113 xmlheader Class Reference

a class for generating the MusicXM header.

`#include <xml.h>`

Inheritance diagram for xmlheader::



Public Types

- enum { **partwise** = 1, **timewise**, **last** = timewise }

Public Member Functions

- void **setMode** (int mode)
  sets the MusicXML score mode
- void **setLocation** (const string &loc)
  sets the MusicXML dtd location
- int **getMode** () const
  returns the MusicXML score mode
- const string & **getLocation** () const
  returns the MusicXML dtd location
- const string **MusicXMLVersion** () const
  returns the current supported MusicXML version
- void **print** (ostream &os) const

Static Public Member Functions

- const string **scoreMode** (int mode)
  provides conversion from numeric to string modes
- int **scoreMode** (string mode)
  provides conversion from string to numeric modes

Protected Member Functions

- **xmlheader** (int mode, string loc)

Friends

- EXP friend **SMARTP< xmlheader > new_xmlheader** (int mode=partwise, string loc="http://www.musicxml.org/dtds/")

7.114 YesNo Class Reference

provides conversions between numeric yes-no types and strings

`#include <conversions.h>`

Public Types

- enum **type** { **undefined**, **yes**, **no**, **last** = no }

Static Public Member Functions

- const string **xml** (type d)

  convert a numeric yes-no value to a MusicXML string

- type **xml** (const string str)

  convert a MusicXML string to a numeric yes-no value

Chapter 8

LibMusicXML Page Documentation

8.1 The MusicXML format

The MusicXML is a xml format that organizes the music into a header followed by the core music data.

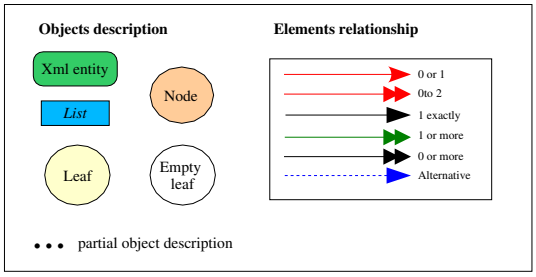


Figure 8.1: MusicXML elements relationship

The core music data may be viewed as *partwise* or *timewise* data:

- *partwise* data are organized into parts containing measures
- *timewise* data are organized into measures containing parts.

8.1.1 MusicXML music data

A MusicXML `part` element may be viewed as a music part, assigned to a given instrument. Therefore a part generally contains a single staff but in case of instruments like the piano or the organ, it includes several staves.

A `measure` contains elements grouped under the `music-data` entity. These elements cover the following purposes:

- music score description. Most of the elements are intended to enumerate the graphic components of a music score. The `note` element is the most important one but the measure

contains also measure specific attributes like `clef`, `key` or `time` signatures, `transpose` indications or `barline` description, as well as `direction` elements attached to a part or the overall score.

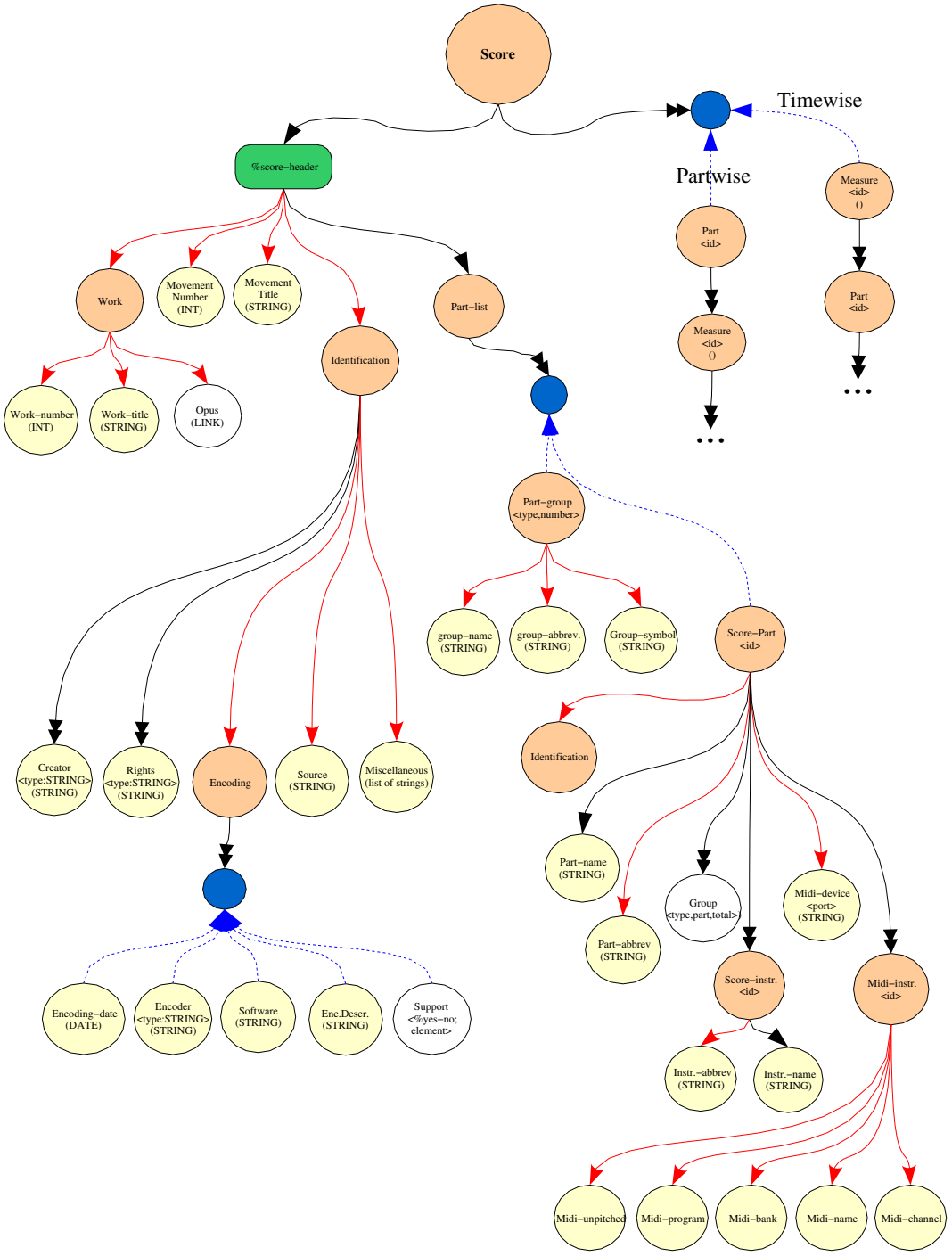


Figure 8.2: The MusicXML Score format

- time description using elements to move the time backward (`backup` element) or forward (`forward` element).

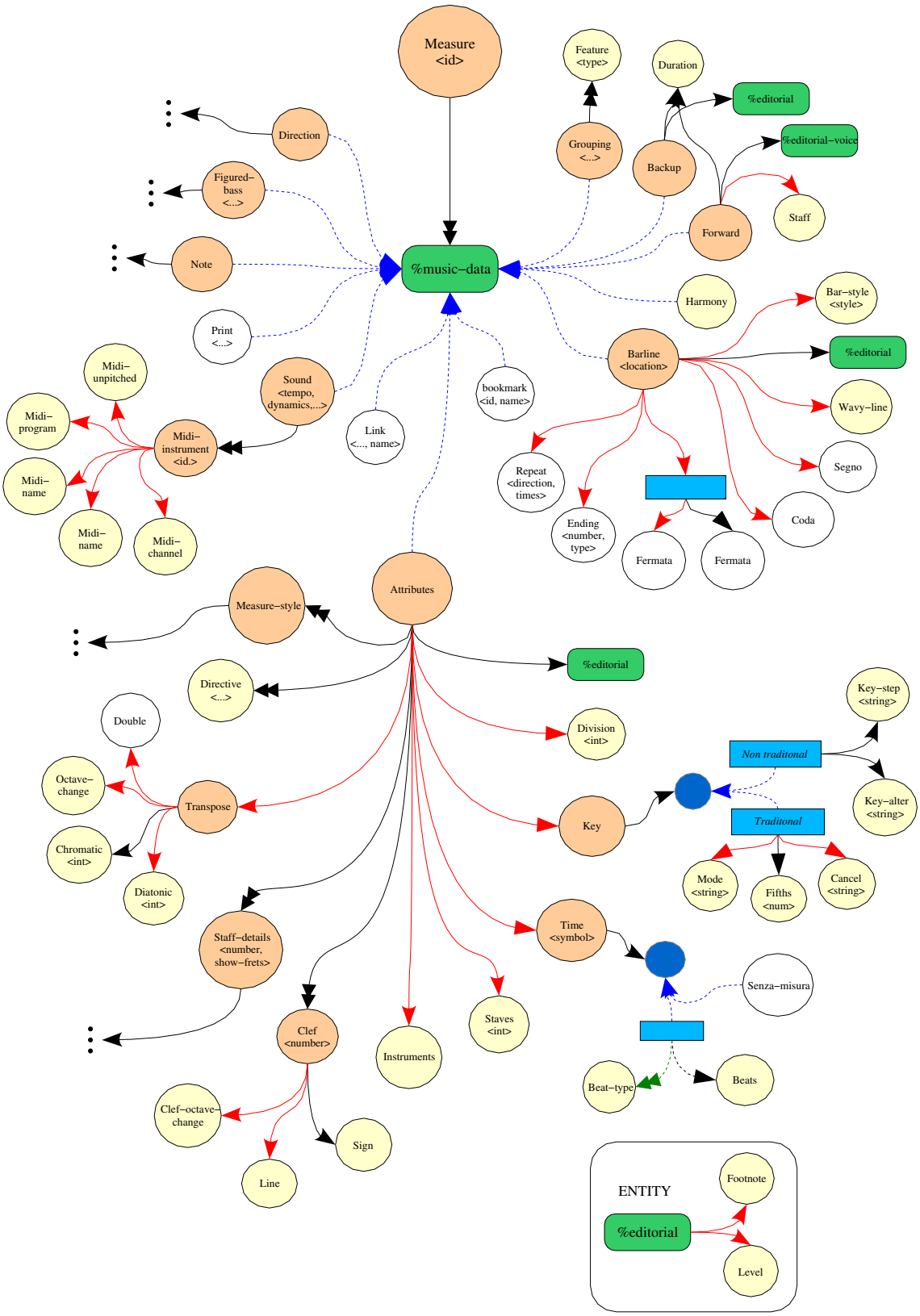


Figure 8.3: The MusicXML `measure` element

- music analysis using elements like `harmony` or `grouping`.
- playback parameters: the `sound` element allows for tempo, dynamics description, but also for sound control including MIDI instrument assignment, and for structural description (dacapo, segno, dalsegno, coda, tocoda).
- XLink support: `link` and `bookmark` elements,
- printing parameters: `print` element.

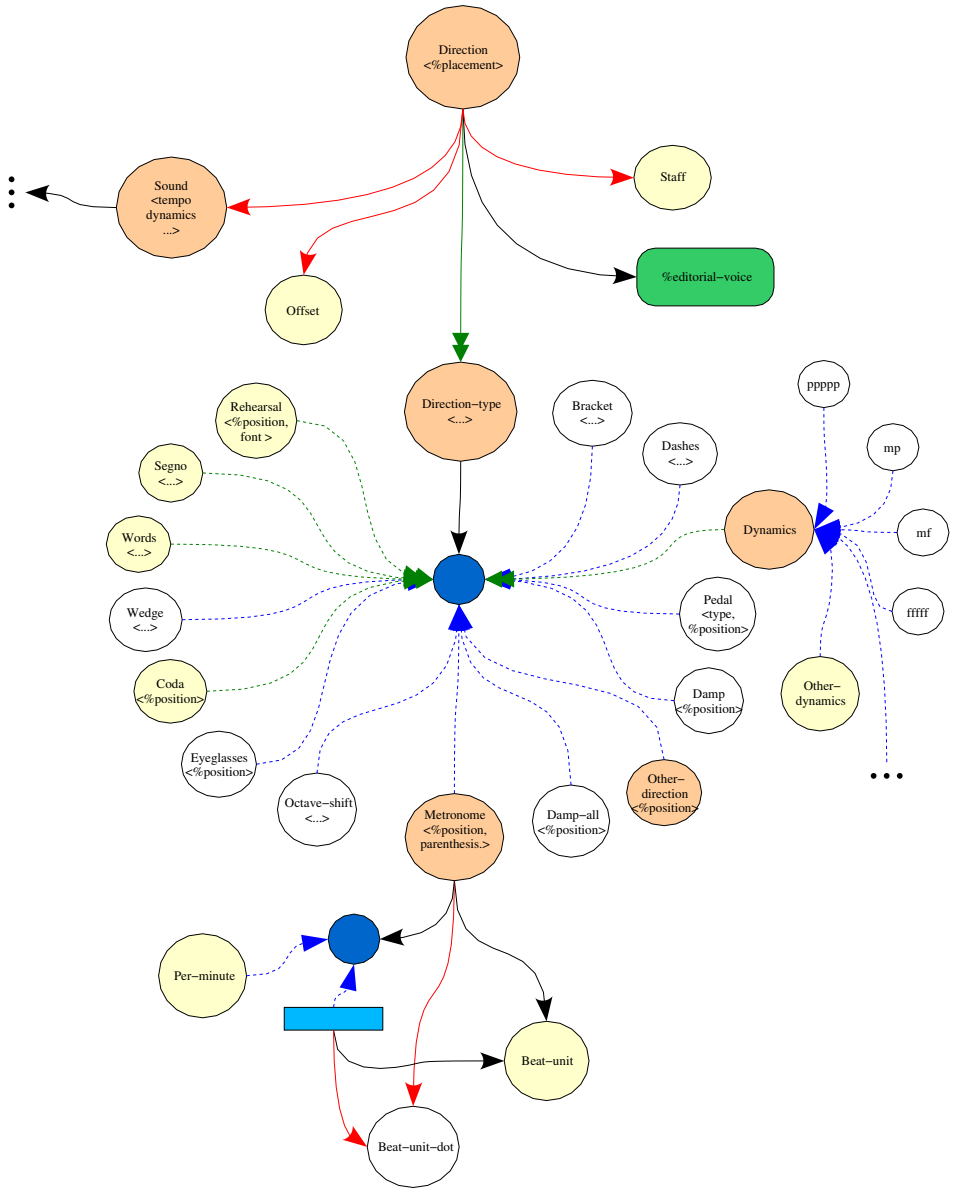


Figure 8.4: The MusicXML `direction` element

The `direction` element is used for musical indications that are not attached to a specific note. Two or more may be combined to indicate starts and stops of wedges, dashes, etc. It supports also dynamics, metronome indications and may include `sound` elements.

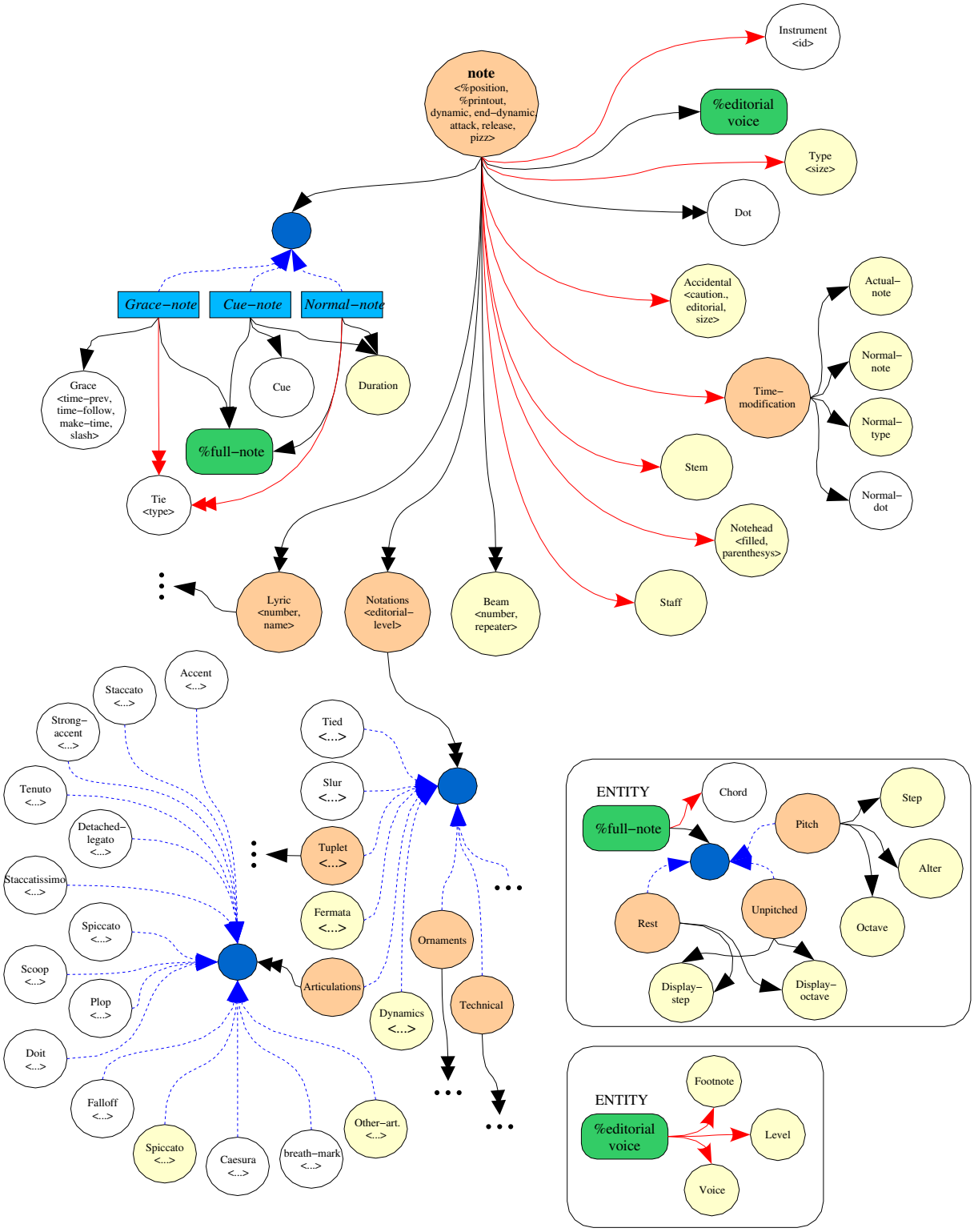


Figure 8.5: The MusicXML `note` element

The `note` element is central to the music description. A note may be a *cue* note, a *grace* note or

a *regular* note. All of them share common elements wich are pitch, chord, and rest information. Unpitched elements are used for unpitched percussion, speaking voice, and other musical elements lacking determinate pitch. A `note` element includes all the necessary for an acurate graphic rendering of all the signs attached to it. It covers:

- the graphic `type` corresponding to the symbolic note duration: whole, half, quarter note...
- possible `accidental` and `dot`,
- `stem` and `beaming` information,
- the graphic `notehead` shape: triangle, diamond, square...
- `staff` assignment
- `notation` elements attached to the note such as `articulation`, `ornament`, `slur`, `tied` ...
- `lyrics`

Note that redundant graphic and sound information may live together, for example:

- a `tuplet` element is present when a tuplet is to be displayed graphically, in addition to the sound data provided by the `time-modification` element.
- a `tie` element indicates sound; the `tied` element indicates notation.
- a `duration` element indicates the note duration in division units. It is the sounding duration vs. notated duration.

A note has also attributes intended for a more acurate graphic placement or to describe differences in duration or dynamics specific to an interpretation or performance.

8.1.2 MusicXML time

MusicXML doesn't include explicit time information in that sense that music events like notes don't carry a date. The time is implicitely moved each time an element that has a duration is encountered. Exception to this rule is the `chord` element: a `chord` element indicates that the note carrying the `chord` element is an additional chord tone with the preceding note.

The duration of a note marked with a `chord` element can be no longer than the preceding note. Although a note that is part of a chord has a duration, the time doesn't move when it is encountered.

Additional elements are provided to explicitely move the time: `backup` and `forward` are intended to coordinate multiple voices in one part, including music on multiple staves. Forward is generally used to move forward within voices and staves, while backup to move backward between voices and staves.

8.2 Sample code

This section provides examples of the MusicXML library use.

8.2.1 Reading and writing MusicXML files

```
/*

  Copyright (C) 2003  Grame
  Grame Research Laboratory, 9 rue du Garet, 69001 Lyon - France
  grame@grame.fr

  This file is provided as an example of the MusicXML Library use.
  It shows how to read and write xml files.
*/

#include <iostream>
#include <string>

#include "TMusicXMLFile.h"

using namespace std;
using namespace MusicXML;


int main (int argc, char * argv[])
{
    // use of a TMusicXMLFile object to read and write xml files
    TMusicXMLFile file;
    for (int i=1; i<argc; i++) {
        // reads the file (file name is readed from the command line)
        SScore score = file.read(argv[i]);
        if (score) {
            // and writes the readed score to the standard output
            // note that 'cout' could be replaced by a file name as well
            file.write (score, cout);
        }
        else {
            cerr << "cannot read file \"" << argv[i] << "\"" << endl;
        }
    }
    return 0;
}
```

8.2.2 Building a score from scratch

```
/*

  Copyright (C) 2003  Grame
  Grame Research Laboratory, 9 rue du Garet, 69001 Lyon - France
  grame@grame.fr

  This file is provided as an example of the MusicXML Library use.
  It shows how to build a memory representation of a score from scratch.

*/


#include <iostream>
#include <sstream>
#include <string>
```

```
#include <cstdlib>
#include <ctime>
#include <math.h>


#ifdef WIN32
#include <windows.h>
#endif

#include "LibMusicXML.h"

using namespace MusicXML;

const string kPartID = "P1";
#define kDivision   4

// a simple function that return random numbers in the given range
int         getrandom(int range);

// the 'scorepart' function builds a score-part MusicXML element
SScorePart  scorepart();

// the 'makeheader' function builds the score header
void        makeheader(SScoreHeader& header);

// creates a measure containing random notes
SPWMeasure  makemeasure(unsigned long num);

// creates specific elements of the score first measure
void        firstmeasure(SPWMeasure& m);

// creates a part containing 'count' measures
SPWPart     makepart(int count);

// the function that creates and writes the score
void        randomMusic(int measuresCount);

//--------------------------------------------------------------------------
// a simple function that return random numbers in the given range
//--------------------------------------------------------------------------
int getrandom(int range) {
    float f = (float)rand() / RAND_MAX;
    return (int)(f * range);
}

//--------------------------------------------------------------------------
// the 'scorepart' function builds a score-part MusicXML element
//--------------------------------------------------------------------------
SScorePart scorepart() {

    SScorePart part = newScorePart(kPartID);// creates the score-part element
    part->setPartName ("Recorder");         // sets the part name

    // creates a score instrument
    SScoreInstrument instrument = newScoreInstrument ("I1");
    instrument->setName ("Recorder");       // sets the instrument name
    part->add(instrument);                  // and adds the instrument to the part

    // creates a MIDI instrument
    SMidiInstrument midi = newMidiInstrument("I1");
    midi->setProgram (75);  // sets the MIDI program to the General MIDI program for the recorder
    part->add(midi);        // and adds the MIDI instrument to the part

    return part;
}

//--------------------------------------------------------------------------
```

```
// the 'makeheader' function builds the score header
// the score header contains various information, including identification,
// parts list etc...
//--------------------------------------------------------------------------
void makeheader(SScoreHeader& header) {
    // creates an identification element
    SIdentification id = newIdentification();

    SCreator creator = newCreator ();       // creates a creator element
    creator->setType("Composer");           // sets the creator type
    creator->setName("Georg Chance");       // sets the creator name
    id->add(creator);                       // adds the creator to the identification

    SEncoding ec = newEncoding();           // creates an encoding element
    ec->setSoftware ("MusicXML Library");   // sets the software name
    id->encoding() = ec;                    // adds the encoding to the identification

    header->identification() = id;          // adds the identification to the header

    header->setMovementTitle ("Random Music");  // sets the movement title
    header->partList() = newPartList();         // creates a part list
    header->partList()->add (scorepart());      // and adds a new part to the part list
}


//--------------------------------------------------------------------------
// creates specific elements of the score first measure
// the first measure is generally a special measure since it contains
// elements like the clef, the time and key signatures etc...
//--------------------------------------------------------------------------
void firstmeasure(SPWMeasure& m) {
    SAttributes attributes = newAttributes();   // creates new attributes

    SClef clef = newClef();                 // creates a new clef
    clef->setSign(TClef::G);                // sets the clef sign
    clef->setLine(2);                       // sets the clef line number
    attributes->add(clef);                  // and adds the clef to the attributes

    STimeSign ts = newTimeSign();           // creates a new time signature
    ts->add (4, 4);                         // sets the time signature to 4/4
    attributes->timeSign() = ts;            // and drop the time sign into the attributes

    // sets the attributes division: division is the MusicXML way to indicates how many
    // divisions per quarter note are used to indicate a note's duration.
    attributes->setDivisions(kDivision);

    m->add (attributes);                    // and adds the attributes to the measure

    SDirection dir = newDirection();        // creates a direction element
    SSound sound = newSound();              // creates a sound element
    long tempo = 120 + getrandom(40)-20;    // computes a random tempo value
    sound->setTempo (tempo);                // and sets the sound tempo
    dir->sound() = sound;                   // and drop the sound element into the direction element

    SDirectionType dt = newDirectionType(); // creates a direction-type element
    SMetronome metro = newMetronome();      // creates a metronome element
    metro->setBeat (NoteType::quarter);     // and sets the metronome to
    metro->setPerMinute (tempo);            // quarter note = tempo
    dt->add(metro);                         // adds the metronome to the direction-type element
    dir->add(dt);                           // adds the direction-type to the direction element
    m->add (dir);                           // and finally adds the direction to the measure
}


//--------------------------------------------------------------------------
// creates a measure containing random notes
// the function takes the measure number as an argument
```

```
//------------------------------------------------------------------------
SPWMeasure makemeasure(unsigned long num) {
    // creates a new partwise measure
    SPWMeasure measure = newPWMeasure(num);

    if (num==1) {                   // if it's the first measure
        firstmeasure(measure);      // creates specific elements of the first measure
    }
    for (int i=0; i<4; i++) {       // next adds 4 quarter notes
        SNote note = newNote();                         // creates the note
        SPitch pitch = newPitch();                      // creates a pitch
        pitch->setStep(getrandom(7)+1);                 // sets the pitch to a random value
        pitch->setOctave (4 + getrandom(2));            // sets the octave to a random value
        note->pitch() = pitch;                          // and drops the pitch to the note
        note->setDuration (kDivision);                  // sets the note duration to a quarter note
        note->graphics() = newGraphNote();              // creates the graphic elements of the note
        note->graphics()->setType (NoteType::quarter);  // and sets the graphic note type to quarter note
        measure->add (note);        // and finally adds the note to the measure
    }
    return measure;
}


//------------------------------------------------------------------------
// creates a part containing 'count' measures
//------------------------------------------------------------------------
SPWPart makepart(int count) {
    SPWPart part = newPWPart(kPartID);      // creates a new part
    for (int i=1; i<=count; i++)            // and 'count' times
        part->add (makemeasure(i));         // adds a new measure to the part
    return part;
}

//------------------------------------------------------------------------
// the function that creates and writes the score
//------------------------------------------------------------------------
void randomMusic(int measuresCount) {
    SScorePartwise score = newScorePartwise();      // creates a partwise score
    score->scoreHeader() = newScoreHeader();        // creates a score header and drops it to the score
    makeheader(score->scoreHeader());               // fills the score header
    score->add(makepart(measuresCount));            // adds a part to the score

    TMusicXMLFile f;
    f.write(score, cout);   // and finally writes the score to the standard output
}

int main (int argc, char * argv[]) {
    // if present, reads the measures count from the command line
    int count = (argc == 2) ? atoi(argv[1]) : 20;
    // sets the random numbers seed
    srand((unsigned)time(0));
    // computes and writes the score
    randomMusic(count);
    return 0;
}
```

8.2.3 Visiting a score

```
/*

  Copyright (C) 2003  Grame
  Grame Research Laboratory, 9 rue du Garet, 69001 Lyon - France
  grame@grame.fr

  This file is provided as an example of the MusicXML Library use.
```

```
  It shows how to read and write xml files. It also gives a concrete
  example of how to browse and modify the memory representation of
  a score.

*/


#include <iostream>

#include "LibMusicXML.h"

using namespace std;
using namespace MusicXML;

//-----------------------------------------------------------------------
//
// This is an example of a visitor implementation. Visitors allows
// to browse the memory representation of a score. Two basic visitors
// are provided by the library:
//  - a TRoutedVisitor which is a visitor that provides a path for
//    traversing the score tree (see the library documentation for the
//    path description).
//  - a TScoreVisitor which is a visitor that does nothing: objects
//    that derive from TScoreVisitor must implement a path for traversing
//    the score tree. This may be necessary only for non-standard way
//    of browsing the score.
// The RawTransposer visitor has no special requirement concerning the path
// for traversing the tree and therefore derives from TRoutedVisitor.
// It is only interested in TNote objects and therefore, only overrides
// the corresponding method.
//
//-----------------------------------------------------------------------
class RawTransposer : public TRoutedVisitor {

    public:
        // constructor is initialized with the transposing interval
        RawTransposer(int interval) : fInterval(interval) {}
        virtual ~RawTransposer() {}
        void visite ( SNote& elt );

    private:
        int fInterval;      // the transposing interval
};

//-----------------------------------------------------------------------
// The raw transposition implementation.
//-----------------------------------------------------------------------
void RawTransposer::visite (SNote& note) {
    // first get the note pitch element (it's a MusicXML pitch)
    SPitch pitch = note->pitch();
    // and next adds the transposing interval
    // the += operator affects the step and octave only (no accidental change)
    *pitch += fInterval;
}

//-----------------------------------------------------------------------
int main (int argc, char * argv[])
{
    if (argc != 3) {
        cerr << "usage: " << argv[0] << " <interval> <file>" << endl;
        exit(1);
    }

    // transposing interval is readed from the command line
    int interval = atoi(argv[1]);
    // use of a TMusicXMLFile object to read and write xml files
    TMusicXMLFile file;
```

```
    // reads the file (file name is readed from the command line)
    SScore score = file.read(argv[2]);
    if (score) {
        if (interval) {
            // declares a RawTransposer object
            RawTransposer transposer(interval);
            // ask the score to accept the transposer
            score->accept(transposer);
        }
        // and finally writes the result on the standard output
        // 'cout' could be replaced by a file as well
        file.write (score, cout);
    }
    else cerr << "cannot read file \"" << argv[2] << "\"" << endl;
    return 0;
}
```

8.3 Todo List

Class guidonotestatus(p. 30) handling the current beat value for *∗num* duration form.

Class TArticulationElement(p. 49) support of the line-shape and line-type attributes

Class TAttributes(p. 52) clarify the Instrument type and semantic
support of the *staff-details* and *measure-style* elements

Class TDirectionType(p. 66) support of the following elements: rehearsal, pedal, damp, damp-all, eyeglasses, other-direction.

Class TDirective(p. 68) support of the *font* entity

Class TDynamic(p. 69) font support

Class TEncoding(p. 71) support of the new *support* element

Class TGraphNote(p. 78) the MusicXML stem definition includes now a position attribute and should be implemented as a separate element

Class TIdentification(p. 80) support of the miscellaneaous element

Class TMeasure(p. 83) support of the *implicit* and *non-controlling* attributes.

Class TNotation(p. 94) implementation of the following elements: glissando, slide, ornaments technical, arpeggiate, non-arpeggiate and other-notation

Class TNote(p. 96) lyrics support

Class TScorePart(p. 129) Support of the MusicXML *group* element (design will change soon).

Class TUnrolledVisitor(p. 156) Management of multiple jump and sound **ForwardRepeat** attribute.

Class TWavyLine(p. 157) support of *trill-sound*

Class TWords(p. 160) font support to be implemented

Class TWork(p. 162) opus support is to be implemented

Member visite(p. 164)**(STimeSign &elt)** check the way unmeasured time is handled
check the symbol translation and handle the single number symbol

Member visite(p. 165)**(SScoreInstrument &elt)** support of the MIDI prog change

Member visite(p. 165)**(SMetronome &elt)** complete translation of TMetronome

Member visite(p. 165)**(SCreator &elt)** handling the different creator types as defined in MusicXML. There is actually no corresponding tag apart the `composer` tag in guido.

Member visite(p. 165)**(SClef &elt)** translation of none and TAB clefs

Member newXML2GuidoVisitor(p. ??)**(bool comments, bool checkStem, bool bar)** support of IDs for tags other than slurs