readosm

1.0.0b

Generated by Doxygen 1.8.1.1

Sun Nov 11 2012 18:19:31

Contents

Chapter 1

Main Page

1.1 Introduction

ReadOSM is a C open source library to extract valid data from within an Open Street Map input file. Such OSM files come in two different formats:

- files identified by the **.osm** suffix simply are plain XML files.
- files identified by the **.pbf** suffix contain the same data, but adopting the Google's Protocol Buffer serialization format (a more concise and compressed binary notation, thus requiring much less storage space).

The ReadOSM design goals are:

- to be simple and lightweight
- to be stable, robust and efficient
- to be easily and universally portable.
- making the whole parsing process of both .osm or .pbf files completely transparent from the application own perspective.

ReadOSM is structurally simple and quite light-weight (typically about 20K of object code, stripped). ReadOSM has only two key dependencies:

- zlib (the well known ZIP library), which is used to decompress zipped binary blocks internally stored within .pbf files.
- expat (a widely used XML parsing library), which is used to parse XML .osm files.
- both libraries are widely available on many platforms.

Building and installing ReadOSM is straightforward:

```
./configure
make
make install
```

Linking ReadOSM to your own code is usually simple:

```
gcc my_program.c -o my_program -lreadosm
```

On some systems you may have to provide a slightly more complex arrangement:

```
gcc -I/usr/local/include my_program.c -o my_program \
  -L/usr/local/lib -lreadosm -lexpat -lz
```

ReadOSM also provides pkg-config support, so you can also do:

```
gcc `pkg-config --cflags readosm` my_program.c -o my_program `pkg-config --libs readosm`
```

I originally developed ReadOSM simply in order to allow the SpatiaLite's own CLI tools to acquire both OSM .osm and .pbf files indifferently. Anyway I feel that supporting OSM files import/parsing in a simple and easy way could be useful to many other developers, so I quickly decided to implement all this stuff as a self-standing library.

ReadOSM is licensed under the MPL tri-license terms: you are free to choose the best-fit license between:

- the MPL 1.1
- the GPL v2.0 or any subsequent version
- the LGPL v2.1 or any subsequent version

Enjoy, and happy coding

Chapter 2

About Open Street Map datasets

Open Street Map aka **OSM** [http://www.openstreetmap.org/] is a very popular community project aimed to produced a map of the world; this map is absolutely free and is released under the ODbL license terms [http://opendatacommons.org/licenses/odbl/].

Selected portions [by Country / Region] of the OSM map are available on the following download sites:

- http://download.geofabrik.de/
- http://downloads.cloudmade.com/

The best known format used to ship OSM datasets is based on XML; we'll shortly examine the XML general layout so to explain the objects used by the OSM data model and their mutual relationships.

2.1 Node

A Node simply corresponds to a 2D POINT Geometry; the geographic coordinates are always expressed as Longitude and Latitude (corresponding to SRID 4326).

A Node doesn't simply have a geometry; it's usually characterized by several data attributes:

- **id:** a number uniquely identifying each Node object.
- **lon** and **lat:** the geographic Longitude and Latitude of the Point.
- **version:** a progressive number identifying subsequent versions of the same object.
- **changeset:** a progressive number identifying a "changeset", i.e. a batch insert/update performed by same user.
- **user:** nickname of the user committing the changeset.
- **uid:** a number uniquely identifying the user
- **timestemp:** commit date-time
- **tag-list**: any object may eventually be further qualified using arbitrary **key:value** pairs.

The following is the XML general layout used to represent a Node object:

```
<node id="12345" lat="6.66666" lon="7.77777" version="1" changeset="54321" user="some-user" uid="66" timestamp
        <tag key="created_by" value="JOSM" />
        <tag key="tourism" value="camp_site" />
</node>
```

2.2 Way

A Way corresponds to a 2D LINESTRING Geometry: anyway the vertices never are directly defined within the Way itself; a list of indirectly referenced Nodes (<**nd ref**> items) is required instead.

The data attributes characterizing a Way are more or less the same used for Nodes, and with identical meaning; and for Ways too an arbitrary collection of Tags (**key:value** pairs) is supported.

The following is the XML general layout used to represent a Way object:

```
<way id="12345" version="1" changeset="54321" user="some-user" uid="66" timestamp="2005-02-28T17:45:15Z">
        <nd ref="12345" />
        <nd ref="12346" />
        <nd ref="12347" />
        <tag key="created_by" value="JOSM" />
        <tag key="tourism" value="camp_site" />
</way>
```

2.3 Relation

A Relation is a complex object: it can correspond to a 2D POLYGON, or to a 2D MULTILINESTRING, or even to a 2D GEOMETRYCOLLECTION.

A Relation object can reference any other kind of OSM objects: each <**member**> item can address a Node object, a Way object or another Relation object; the **type** attribute will always specify the nature of the referenced object, and the optional **role** attribute may eventually better specify the intended scope.

The data attributes characterizing a Relation are exactly the same used for Ways, and with identical meaning; and for Relations too an arbitrary collection of Tags (**key:value** pairs) is supported.

The following is the XML general layout used to represent a Relation object:

```
<relation id="12345" version="1" changeset="54321" user="some-user" uid="66" timestamp="2005-02-28T17:45:15Z">
        <member type="way" ref="12345" role="outer" />
        <member type="way" ref="12346" role="inner" />
        <tag key="created_by" value="JOSM" />
        <tag key="tourism" value="camp_site" />
</relation>
```

Chapter 3

Open Street Map file formats

There are two distinct formats used to ship OSM datasets: both contains the exact same information, but the internal layout is radically different.

3.1 XML (.osm) files

OSM files based on the XML notation are widely used: usually they are identified by the **.osm** suffix.

XML is notoriously verbose and usually requires lots of storage space; happily enough, XML it's strongly compressible.

Accordingly to this consideration, the most commonly found OSM files are identified by the **.osm.bz2** suffix: this practically means that the **.osm** (XML) file has been compressed using **bzip2**. In order to actually process a **.osm.bz2** OSM file a two-steps approach is always required:

- decompressing the file (using **bunzip2** or some other tool)
- then parsing the resulting **.osm** file
- please note: the inflated file will require about 10/15 times the amount space required by the compressed file; many OSM XML files could actually be impressively huge (several GB).

3.2 Protocol Buffer (.pbf) files

An alternative OSM file format is based on the Google's Protocol Buffer encoding [https://developers.google.com/protocol-buffers/docs/encoding]

This OSM format is based on a public and documented specification: [http://wiki.openstreetmap.org/wiki/PBF_Format]

OSM files based on Protocol Buffer encoding are usually identified by the **.pbf** suffix.

The main benefit coming from using **.pbf** files is in that they are much more compact (smaller size) than the corresponding **.osm.bz2**; and they can be immediately parsed, no preliminary decompression step being required at all.

3.3 ReadOSM basic architecture

The intended scope of **ReadOSM** is to allow transparent parsing of both OSM formats indifferently. There is no need to take care of any internal low-level aspect, because the library itself silently handles any required step. The simple and easy abstract interface implemented by ReadOSM is exactly intended so to allow many reader-apps to consume OSM-input files in the most painless way; and all this requires only a very limited memory footprint.

Chapter 4

ReadOSM basic architecture

ReadOSM implements a very simple and straightforward interface; there are only three methods:

- **readosm_open()**: this function is intended to establish a connection to some OSM input file.
- **readosm_close()**: this function is intended to terminate a previously established connection.
- **readosm_parse()**: a single function dispatching the whole parsing process (mainly based on **callback functions**).

Accordingly to the above premises, implementing a complete OSM parser is incredibly simple:

```
#include <readosm.h>

static int
parse_node (const void *user_data, const readosm_node * node)
{
/* callback function consuming Node objects */
  struct some_user_defined_struct *my_struct =
    (struct some_user_defined_struct *) user_data;

  ... some smart code ...

  return READOSM_OK;
}

static int
parse_way (const void *user_data, const readosm_way * way)
{
/* callback function consuming Way objects */
  struct some_user_defined_struct *my_struct =
    (struct some_user_defined_struct *) user_data;

  ... some smart code ...

  return READOSM_OK;
}

static int
parse_relation (const void *user_data, const readosm_relation * relation)
{
/* callback function consuming Relation objects */
  struct some_user_defined_struct *my_struct =
    (struct some_user_defined_struct *) user_data;

  ... some smart code ...

  return READOSM_OK;
}

int main ()
{
```

```
/* the basic OSM parser implementation */
  int ret;
  const void *handle;
  struct some_user_defined_struct my_struct;

  ret = readosm_open ("path-to-some-OSM-file", &handle);

  ... error handling intentionally suppressed ...

  ret = readosm_parse (handle, &my_struct, parse_node, parse_way, parse_relation);

  ... error handling intentionally suppressed ...

  ret = readosm_close (handle);

  ... error handling intentionally suppressed ...

  return 0;
}
```

So the real programming work is simply the one required in order to implement the callback-functions own code.

You can usefully read and study the **Examples** code-samples in order to get any other relevant information about this topic.

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

File Index

6.1 File List

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

Chapter 7

Data Structure Documentation

7.1 readosm_member_struct Struct Reference

a struct representing a RELATION-MEMBER, and wrapping an XML fragment like the following:

`#include <readosm.h>`

Data Fields

- const int member_type

  can be one of: READOSM_MEMBER_NODE, READOSM_MEMBER_WAY or READOSM_MEMBER_RELATION
- const long long id

  ID-value identifying the referenced object.
- const char * role

  intended role for this reference

7.1.1 Detailed Description

a struct representing a RELATION-MEMBER, and wrapping an XML fragment like the following:

```
@verbatim
```

<member type="some-type" ref="12345" role="some-role">

Examples:

test_osm1.c, and test_osm2.c.

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

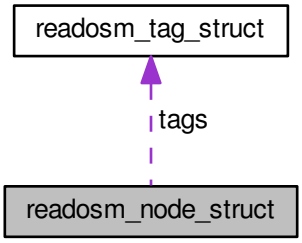
- headers/readosm.h

7.2 readosm_node_struct Struct Reference

a struct representing a NODE object, and wrapping a complex XML fragment like the following:

`#include <readosm.h>`

Collaboration diagram for readosm_node_struct:



Data Fields

- const long long id
  NODE-ID (expected to be a unique value)
- const double latitude
  geographic latitude
- const double longitude
  geographic longitude
- const int version
  object version
- const long long changeset
  ChangeSet ID.
- const char * user
  name of the User defining this NODE
- const int uid
  corresponding numeric UserID
- const char * timestamp
  when this NODE was defined
- const int tag_count
  number of associated TAGs (may be zero)
- const readosm_tag * tags
  array of TAG objects (may be NULL)

7.2.1 Detailed Description

a struct representing a NODE object, and wrapping a complex XML fragment like the following:

```
@verbatim
```

<node id="12345" lat="6.66666" lon="7.77777" version="1" changeset="54321" user="some-user" uid="66" timestamp="2005-02-28T17:45:15Z"> <tag key="created_by" value="JOSM"> <tag key="tourism" value="camp-_site"> </node>

Examples:

test_osm1.c, test_osm2.c, and test_osm3.c.

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

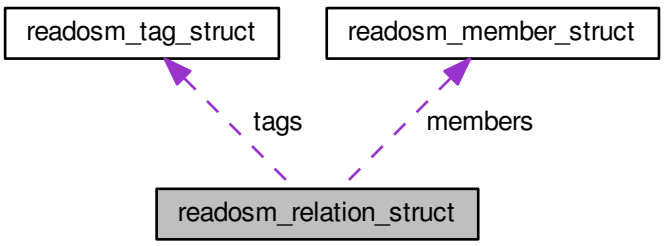
- headers/readosm.h

7.3 readosm_relation_struct Struct Reference

a struct representing a RELATION object, and wrapping a complex XML fragment like the following:

```
#include <readosm.h>
```

Collaboration diagram for readosm_relation_struct:



Data Fields

- const long long id

  RELATION-ID (expected to be a unique value)
- const int version

  object version
- const long long changeset

  ChangeSet ID.
- const char * user

  name of the User defining this RELATION
- const int uid

  corresponding numeric UserID
- const char * timestamp

  when this RELATION was defined
- const int member_count

  number of associated MEMBERs (may be zero)
- const readosm_member * members

  array of MEMBER objects (may be NULL)
- const int tag_count

  number of associated TAGs (may be zero)
- const readosm_tag * tags

  array of TAG objects (may be NULL)

7.3.1 Detailed Description

a struct representing a RELATION object, and wrapping a complex XML fragment like the following:

```
@verbatim
```

<relation id="12345" version="1" changeset="54321" user="some-user" uid="66" timestamp="2005-02-28T17:45-:15Z"> <member type="way" ref="12345" role="outer"> <member type="way" ref="12346" role="inner"> <tag key="created_by" value="JOSM"> <tag key="tourism" value="camp_site"> </relation>

Examples:

test_osm1.c, test_osm2.c, and test_osm3.c.

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

- headers/readosm.h

7.4 readosm_tag_struct Struct Reference

a struct representing a **key:value** pair, and wrapping an XML fragment like the following:

```
#include <readosm.h>
```

Data Fields

- const char * key
  the KEY
- const char * value
  the VALUE

7.4.1 Detailed Description

a struct representing a **key:value** pair, and wrapping an XML fragment like the following:

```
@verbatim
```

<tag key="key-value" value="some-value">

Examples:

test_osm1.c.

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

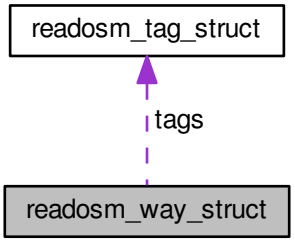
- headers/readosm.h

7.5 readosm_way_struct Struct Reference

a struct representing a WAY object, and wrapping a complex XML fragment like the following:

```
#include <readosm.h>
```

Collaboration diagram for readosm_way_struct:



Data Fields

- const long long id
  WAY-ID (expected to be a unique value)
- const int version
  object version
- const long long changeset
  ChangeSet ID.
- const char * user
  name of the User defining this WAY
- const int uid
  corresponding numeric UserID
- const char * timestamp
  when this WAY was defined
- const int node_ref_count
  number of referenced NODE-IDs (may be zero)
- const long long * node_refs
  array of NODE-IDs (may be NULL)
- const int tag_count
  number of associated TAGs (may be zero)
- const readosm_tag * tags
  array of TAG objects (may be NULL)

7.5.1 Detailed Description

a struct representing a WAY object, and wrapping a complex XML fragment like the following:

```
@verbatim
```

<way id="12345" version="1" changeset="54321" user="some-user" uid="66" timestamp="2005-02-28T17:45:15-Z"> <nd ref="12345"> <nd ref="12346"> <nd ref="12347"> <tag key="created_by" value="JOSM"> <tag key="tourism" value="camp_site"> </way>

Examples:

test_osm1.c, test_osm2.c, and test_osm3.c.

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

- headers/readosm.h

Chapter 8

File Documentation

8.1 headers/readosm.h File Reference

Function declarations and constants for ReadOSM library.

Data Structures

- struct readosm_tag_struct

  *a struct representing a **key:value** pair, and wrapping an XML fragment like the following:*
- struct readosm_node_struct

  a struct representing a NODE object, and wrapping a complex XML fragment like the following:
- struct readosm_way_struct

  a struct representing a WAY object, and wrapping a complex XML fragment like the following:
- struct readosm_member_struct

  a struct representing a RELATION-MEMBER, and wrapping an XML fragment like the following:
- struct readosm_relation_struct

  a struct representing a RELATION object, and wrapping a complex XML fragment like the following:

Macros

- #define READOSM_UNDEFINED -1234567890

  information is not available
- #define READOSM_MEMBER_NODE 7361

  MemberType: NODE.
- #define READOSM_MEMBER_WAY 6731

  MemberType: WAY.
- #define READOSM_MEMBER_RELATION 3671

  MemberType: RELATION.
- #define READOSM_OK 0

  No error, success.
- #define READOSM_INVALID_SUFFIX -1

  not .osm or .pbf suffix
- #define READOSM_FILE_NOT_FOUND -2

  .osm or .pbf file does not exist or is not accessible for reading
- #define READOSM_NULL_HANDLE -3

  Null OSM_handle argument.
- #define READOSM_INVALID_HANDLE -4

Invalid OSM_handle argument.

- #define READOSM_INSUFFICIENT_MEMORY -5

  some kind of memory allocation failure
- #define READOSM_CREATE_XML_PARSER_ERROR -6

  cannot create the XML Parser
- #define READOSM_READ_ERROR -7

  read error
- #define READOSM_XML_ERROR -8

  XML parser error.
- #define READOSM_INVALID_PBF_HEADER -9

  invalid PBF header
- #define READOSM_UNZIP_ERROR -10

  unZip error
- #define READOSM_ABORT -11

  user-required parser abort

Typedefs

- typedef struct readosm_tag_struct readosm_tag

  Typedef for TAG structure.
- typedef struct readosm_node_struct readosm_node

  Typedef for NODE structure.
- typedef struct readosm_way_struct readosm_way

  Typedef for WAY structure.
- typedef struct readosm_member_struct readosm_member

  Typedef for MEMBER structure.
- typedef struct readosm_relation_struct readosm_relation

  Typedef for RELATION structure.
- typedef int(∗ readosm_node_callback)(const void ∗user_data, const readosm_node ∗node)

  callback function handling NODE objects
- typedef int(∗ readosm_way_callback)(const void ∗user_data, const readosm_way ∗way)

  callback function handling WAY objects
- typedef int(∗ readosm_relation_callback)(const void ∗user_data, const readosm_relation ∗relation)

  callback function handling RELATION objects

Functions

- READOSM_DECLARE int readosm_open (const char ∗path, const void ∗∗osm_handle)

  Open the .osm or .pbf file, preparing for future functions.
- READOSM_DECLARE int readosm_close (const void ∗osm_handle)

  Close the .osm or .pbf file and release any allocated resource.
- READOSM_DECLARE int readosm_parse (const void ∗osm_handle, const void ∗user_data, readosm_node_callback node_fnct, readosm_way_callback way_fnct, readosm_relation_callback relation_fnct)

  Close the .osm or .pbf file and release any allocated resource.

8.1.1 Detailed Description

Function declarations and constants for ReadOSM library.

8.1.2 Typedef Documentation

8.1.2.1 typedef struct readosm_member_struct readosm_member

Typedef for MEMBER structure.

See also

readosm_member_struct

8.1.2.2 typedef struct readosm_node_struct readosm_node

Typedef for NODE structure.

See also

readosm_node_struct

8.1.2.3 typedef struct readosm_relation_struct readosm_relation

Typedef for RELATION structure.

See also

readosm_relation_struct

8.1.2.4 typedef struct readosm_tag_struct readosm_tag

Typedef for TAG structure.

See also

readosm_tag_struct

8.1.2.5 typedef struct readosm_way_struct readosm_way

Typedef for WAY structure.

See also

readosm_way_struct

8.1.3 Function Documentation

8.1.3.1 READOSM_DECLARE int readosm_close (const void * *osm_handle*)

Close the .osm or .pbf file and release any allocated resource.

Parameters

osm_handle	the handle previously returned by readosm_open()

Returns

READOSM_OK will be returned on success, otherwise any appropriate error code on failure.

Note

After calling readosm_close() any related resource will be released, and the handle will no longer be valid.

Examples:

test_osm1.c, test_osm2.c, and test_osm3.c.

8.1.3.2 READOSM_DECLARE int readosm_open (const char * *path*, const void ** *osm_handle*)

Open the .osm or .pbf file, preparing for future functions.

Parameters

path	full or relative pathname of the input file.
osm_handle	an opaque reference (handle) to be used in each subsequent function (return value).

Returns

READOSM_OK will be returned on success, otherwise any appropriate error code on failure.

Note

You are expected to readosm_close() even on failure, so as to correctly release any dynamic memory allocation.

Examples:

test_osm1.c, test_osm2.c, and test_osm3.c.

8.1.3.3 READOSM_DECLARE int readosm_parse (const void * *osm_handle*, const void * *user_data*, readosm_node_callback *node_fnct*, readosm_way_callback *way_fnct*, readosm_relation_callback *relation_fnct*)

Close the .osm or .pbf file and release any allocated resource.

```
\param osm_handle the handle previously returned by readosm_open()
```

Parameters

user_data	pointer to some user-supplied data struct
node_fnct	pointer to callback function intended to consume NODE objects (may be NULL if processing NODEs is not an interesting option)
way_fnct	pointer to callback function intended to consume WAY objects (may be NULL if processing WAYs is not an interesting option)
relation_fnct	pointer to callback function intended to consume RELATION objects (may be NULL if processing RELATIONs is not an interesting option)

Returns

READOSM_OK will be returned on success, otherwise any appropriate error code on failure.

Note

After calling readosm_close() any related resource will be released, and the handle will no longer be valid.

Examples:

test_osm1.c, test_osm2.c, and test_osm3.c.

Chapter 9

Example Documentation

9.1 test_osm1.c

test_osm1.c is a simple demonstration tool for OSM file formats.This sample code provides an example of:

- opening the OSM file
- parsing the OSM file, then printing an XML-like notation on the standard output.
- closing the OSM file when no further operations are required

Here is an example of a typical run:

```
./test_osm1 italy.osm >italy-from-xml
or
./test_osm1 italy.osm.pbf >italy-from-pbf
```

Please note: the output produced by test_osm1 is usually verbose, so redirecting the standard output to a disk file is strongly recommended.

```
/*
/ test_osm1.c
/
/ libreadosm Sample code
/
/ Author: Sandro Furieri a.furieri@lqt.it
/
/
      ------------------------------------------------------------------------
/
/ Version: MPL 1.1/GPL 2.0/LGPL 2.1
/
/ The contents of this file are subject to the Mozilla Public License Version
/ 1.1 (the "License"); you may not use this file except in compliance with
/ the License. You may obtain a copy of the License at
/ http://www.mozilla.org/MPL/
/
/ Software distributed under the License is distributed on an "AS IS" basis,
/ WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License
/ for the specific language governing rights and limitations under the
/ License.
/
/ The Original Code is the ReadOSM library
/
/ The Initial Developer of the Original Code is Alessandro Furieri
/
/ Portions created by the Initial Developer are Copyright (C) 2012
/ the Initial Developer. All Rights Reserved.
/
/ Contributor(s):
/
/ Alternatively, the contents of this file may be used under the terms of
/ either the GNU General Public License Version 2 or later (the "GPL"), or
/ the GNU Lesser General Public License Version 2.1 or later (the "LGPL"),
/ in which case the provisions of the GPL or the LGPL are applicable instead
```

```
/ of those above. If you wish to allow use of your version of this file only
/ under the terms of either the GPL or the LGPL, and not to allow others to
/ use your version of this file under the terms of the MPL, indicate your
/ decision by deleting the provisions above and replace them with the notice
/ and other provisions required by the GPL or the LGPL. If you do not delete
/ the provisions above, a recipient may use your version of this file under
/ the terms of any one of the MPL, the GPL or the LGPL.
/
*/

#include <stdio.h>

#include "readosm.h"

static int
print_node (const void *user_data, const readosm_node * node)
{
/*
* printing an OSM Node (callback function)
*
* this function is called by the OSM parser for each
* NODE object found
*
* please note well: the passed pointer corresponds to
* a READ-ONLY object; you can can query any node-related
* value, but you cannot alter them.
*
***********************************************
*
* this didactic sample will simply print the node object
* on the standard output adopting the appropriate OSM XML
* notation
*/
    char buf[128];
    int i;
    const readosm_tag *tag;

#if defined(_WIN32) || defined(__MINGW32__)
    /* CAVEAT - M$ runtime doesn't supports %lld for 64 bits */
    sprintf (buf, "%I64d", node->id);
#else
    sprintf (buf, "%lld", node->id);
#endif
    printf ("\t<node id=\"%s\"", buf);

/*
* some individual values may be set, or may be not
* unset values are identified by the READOSM_UNDEFINED
* conventional value, and must be consequently ignored
*/
    if (node->latitude != READOSM_UNDEFINED)
        printf (" lat=\"%1.7f\"", node->latitude);
    if (node->longitude != READOSM_UNDEFINED)
        printf (" lon=\"%1.7f\"", node->longitude);
    if (node->version != READOSM_UNDEFINED)
        printf (" version=\"%d\"", node->version);
    if (node->changeset != READOSM_UNDEFINED)
      {

#if defined(_WIN32) || defined(__MINGW32__)
          /* CAVEAT - M$ runtime doesn't supports %lld for 64 bits */
          sprintf (buf, "%I64d", node->changeset);
#else
          sprintf (buf, "%lld", node->changeset);
#endif
          printf (" changeset=\"%s\"", buf);
      }

/*
* unset string values are identified by a NULL pointer
* and must be consequently ignored
*/
    if (node->user != NULL)
        printf (" user=\"%s\"", node->user);
    if (node->uid != READOSM_UNDEFINED)
        printf (" uid=\"%d\"", node->uid);
    if (node->timestamp != NULL)
        printf (" timestamp=\"%s\"", node->timestamp);

/*
* the Node object may have its own tag list
* please note: this one is a variable-length list,
* and may be empty: in this case tag_count will be ZERO
*/
    if (node->tag_count == 0)
        printf (" />\n");
    else
```

```
      {
          printf (">\n");
          for (i = 0; i < node->tag_count; i++)
            {
                /* we'll now print each <tag> for this node */
                tag = node->tags + i;
                printf ("\t\t<tag k=\"%s\" v=\"%s\" />\n", tag->key,
                        tag->value);
            }
          printf ("\t</node>\n");
      }
    return READOSM_OK;
}

static int
print_way (const void *user_data, const readosm_way * way)
{
/*
* printing an OSM Way (callback function)
*
* this function is called by the OSM parser for each
* WAY object found
*
* please note well: the passed pointer corresponds to
* a READ-ONLY object; you can can query any way-related
* value, but you cannot alter them.
*
**********************************************
*
* this didactic sample will simply print the way object
* on the standard output adopting the appropriate OSM XML
* notation
*/
    char buf[128];
    int i;
    const readosm_tag *tag;

#if defined(_WIN32) || defined(__MINGW32__)
    /* CAVEAT - M$ runtime doesn't supports %lld for 64 bits */
    sprintf (buf, "%I64d", way->id);
#else
    sprintf (buf, "%lld", way->id);
#endif
    printf ("\t<way id=\"%s\"", buf);

/*
* some individual values may be set, or may be not
* unset values are identified by the READOSM_UNDEFINED
* conventional value, and must be consequently ignored
*/
    if (way->version != READOSM_UNDEFINED)
        printf (" version=\"%d\"", way->version);
    if (way->changeset != READOSM_UNDEFINED)
      {
#if defined(_WIN32) || defined(__MINGW32__)
          /* CAVEAT - M$ runtime doesn't supports %lld for 64 bits */
          sprintf (buf, "%I64d", way->changeset);
#else
          sprintf (buf, "%lld", way->changeset);
#endif
          printf (" changeset=\"%s\"", buf);
      }

/*
* unset string values are identified by a NULL pointer
* and must be consequently ignored
*/
    if (way->user != NULL)
        printf (" user=\"%s\"", way->user);
    if (way->uid != READOSM_UNDEFINED)
        printf (" uid=\"%d\"", way->uid);
    if (way->timestamp != NULL)
        printf (" timestamp=\"%s\"", way->timestamp);

/*
* the Way object may have a noderefs-list and a tag-list
* please note: these are variable-length lists, and may
* be empty: in this case the corresponding item count
* will be ZERO
*/
    if (way->tag_count == 0 && way->node_ref_count == 0)
        printf (" />\n");
    else
      {
          printf (">\n");
          for (i = 0; i < way->node_ref_count; i++)
            {
```

```
                /* we'll now print each <nd ref> for this way */
#if defined(_WIN32) || defined(__MINGW32__)
                /* CAVEAT - M$ runtime doesn't supports %lld for 64 bits */
                sprintf (buf, "%I64d", *(way->node_refs + i));
#else
                sprintf (buf, "%lld", *(way->node_refs + i));
#endif
                printf ("\t\t<nd ref=\"%s\" />\n", buf);
            }
          for (i = 0; i < way->tag_count; i++)
            {
                /* we'll now print each <tag> for this way */
                tag = way->tags + i;
                printf ("\t\t<tag k=\"%s\" v=\"%s\" />\n", tag->key,
                        tag->value);
            }
          printf ("\t</way>\n");
      }
    return READOSM_OK;
}

static int
print_relation (const void *user_data, const readosm_relation *
       relation)
{
/*
* printing an OSM Relation (callback function)
*
* this function is called by the OSM parser for each
* RELATION object found
*
* please note well: the passed pointer corresponds to
* a READ-ONLY object; you can can query any relation-related
* value, but you cannot alter them.
*
************************************************
*
* this didactic sample will simply print the relation object
* on the standard output adopting the appropriate OSM XML
* notation
*/
    char buf[128];
    int i;
    const readosm_member *member;
    const readosm_tag *tag;

#if defined(_WIN32) || defined(__MINGW32__)
    /* CAVEAT - M$ runtime doesn't supports %lld for 64 bits */
    sprintf (buf, "%I64d", relation->id);
#else
    sprintf (buf, "%lld", relation->id);
#endif
    printf ("\t<relation id=\"%s\"", buf);

/*
* some individual values may be set, or may be not
* unset values are identified by the READOSM_UNDEFINED
* conventional value, and must be consequently ignored
*/
    if (relation->version != READOSM_UNDEFINED)
        printf (" version=\"%d\"", relation->version);
    if (relation->changeset != READOSM_UNDEFINED)
      {
#if defined(_WIN32) || defined(__MINGW32__)
          /* CAVEAT - M$ runtime doesn't supports %lld for 64 bits */
          sprintf (buf, "%I64d", relation->changeset);
#else
          sprintf (buf, "%lld", relation->changeset);
#endif
          printf (" changeset=\"%s\"", buf);
      }

/*
* unset string values are identified by a NULL pointer
* and must be consequently ignored
*/
    if (relation->user != NULL)
        printf (" user=\"%s\"", relation->user);
    if (relation->uid != READOSM_UNDEFINED)
        printf (" uid=\"%d\"", relation->uid);
    if (relation->timestamp != NULL)
        printf (" timestamp=\"%s\"", relation->timestamp);

/*
* the Relation object may have a member-list and a tag-list
* please note: these are variable-length lists, and may
* be empty: in this case the corresponding item count
```

```
* will be ZERO
*/
    if (relation->tag_count == 0 && relation->member_count
       == 0)
        printf (" />\n");
    else
      {
          printf (">\n");
          for (i = 0; i < relation->member_count; i++)
            {
                /* we'll now print each <member> for this way */
                member = relation->members + i;
#if defined(_WIN32) || defined(__MINGW32__)
                /* CAVEAT - M$ runtime doesn't supports %lld for 64 bits */
                sprintf (buf, "%I64d", member->id);
#else
                sprintf (buf, "%lld", member->id);
#endif
                /* any <member> may be of "node", "way" or "relation" type */
                switch (member->member_type)
                  {
                  case READOSM_MEMBER_NODE:
                      printf ("\t\t<member type=\"node\" ref=\"%s\"", buf);
                      break;
                  case READOSM_MEMBER_WAY:
                      printf ("\t\t<member type=\"way\" ref=\"%s\"", buf);
                      break;
                  case READOSM_MEMBER_RELATION:
                      printf ("\t\t<member type=\"relation\" ref=\"%s\"", buf);
                      break;
                  default:
                      printf ("\t\t<member ref=\"%s\"", buf);
                      break;
                  };
                if (member->role != NULL)
                    printf (" role=\"%s\" />\n", member->role);
                else
                    printf (" />\n");
            }
          for (i = 0; i < relation->tag_count; i++)
            {
                /* we'll now print each <tag> for this way */
                tag = relation->tags + i;
                printf ("\t\t<tag k=\"%s\" v=\"%s\" />\n", tag->key,
                        tag->value);
            }
          printf ("\t</relation>\n");
      }
    return READOSM_OK;
}

int
main (int argc, char *argv[])
{
    const void *osm_handle;
    int ret;

    if (argc != 2)
      {
          fprintf (stderr, "usage: test_osm1 path-to-OSM-file\n");
          return -1;
      }

/*
* STEP #1: opening the OSM file
* this can indifferently be an OSM XML encoded file (.osm)
* or an OSM Protocol Buffer encoded file (.pbf)
* the library will transparently perform any required
* action in both cases.
*/
    ret = readosm_open (argv[1], &osm_handle);
    if (ret != READOSM_OK)
      {
          fprintf (stderr, "OPEN error: %d\n", ret);
          goto stop;
      }

/*
* STEP #2: parsing the OSM file
* this task is unbelievably simple
*
* you are simply required to pass the appropriate
* pointers for callback funtions respectively intended
* to process Node-objects, Way-objects and Relation-objects
*
* the library will then parse the whole input file, calling
* the appropriate callback handling function for each OSM object
```

```
* found: please see the callback functions implementing code
* to better understand how it works
*
* important notice: in this first example we'll not use at
* all the USER_DATA pointer. so the second arg will simply
* be (const void *)0 [i.e. NULL]
*/
    ret =
        readosm_parse (osm_handle, (const void *) 0, print_node,
      print_way,
                       print_relation);
    if (ret != READOSM_OK)
      {
          fprintf (stderr, "PARSE error: %d\n", ret);
          goto stop;
      }

    fprintf (stderr, "Ok, OSM input file successfully parsed\n");

  stop:
/*
* STEP #3: closing the OSM file
* this will release any internal memory allocation
*/
    readosm_close (osm_handle);
    return 0;
}
```

9.2 test_osm2.c

test_osm2.c is another simple demonstration tool for OSM file formats.This sample code provides an example of:

- opening the OSM file
- parsing the OSM file, thus collecting and printing simple statistics about NODEs, WAYs and RELATIONs
- error handling
- closing the OSM file when no further operations are required

Here is a typical usage example, parsing an OSM XML file (.osm):

```
./test_osm2 test.osm
Longitude    range: 8.7889611 / 9.4145124
Latitude     range: 41.3870658 / 42.8070090

Nodes             : 1060
               tags: 1052

Ways              : 112
              ndref: 785
               tags: 241

Relations         : 13
  member.nodes    : 16
  member.ways     : 44
  member.relations: 6
              tags: 199
```

Here is another example, this time parsing a .pbf (Protocol Buffer) OSM file:

```
./test_osm2 test.osm
Longitude    range: 8.5856726 / 10.2898441
Latitude     range: 41.3332843 / 43.5406952

Nodes             : 8000
               tags: 3162

Ways              : 12336
              ndref: 221627
               tags: 24904
```

```
Relations         : 1520
  member.nodes    : 2952
  member.ways     : 2741
  member.relations: 30
              tags: 10081
```

```
/*
/ test_osm2.c
/
/ libreadosm Sample code
/
/ Author: Sandro Furieri a.furieri@lqt.it
/
/
      ------------------------------------------------------------------------
/
/ Version: MPL 1.1/GPL 2.0/LGPL 2.1
/
/ The contents of this file are subject to the Mozilla Public License Version
/ 1.1 (the "License"); you may not use this file except in compliance with
/ the License. You may obtain a copy of the License at
/ http://www.mozilla.org/MPL/
/
/ Software distributed under the License is distributed on an "AS IS" basis,
/ WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License
/ for the specific language governing rights and limitations under the
/ License.
/
/ The Original Code is the ReadOSM library
/
/ The Initial Developer of the Original Code is Alessandro Furieri
/
/ Portions created by the Initial Developer are Copyright (C) 2012
/ the Initial Developer. All Rights Reserved.
/
/ Contributor(s):
/
/ Alternatively, the contents of this file may be used under the terms of
/ either the GNU General Public License Version 2 or later (the "GPL"), or
/ the GNU Lesser General Public License Version 2.1 or later (the "LGPL"),
/ in which case the provisions of the GPL or the LGPL are applicable instead
/ of those above. If you wish to allow use of your version of this file only
/ under the terms of either the GPL or the LGPL, and not to allow others to
/ use your version of this file under the terms of the MPL, indicate your
/ decision by deleting the provisions above and replace them with the notice
/ and other provisions required by the GPL or the LGPL. If you do not delete
/ the provisions above, a recipient may use your version of this file under
/ the terms of any one of the MPL, the GPL or the LGPL.
/
*/

#include <stdio.h>

#include "readosm.h"

struct osm_statistics
{
/* a struct intended to collect general OSM statistics */
    int node_count;
    int node_tag_count;
    int way_count;
    int way_ndref_count;
    int way_tag_count;
    int relation_count;
    int relation_member_node_count;
    int relation_member_way_count;
    int relation_member_relation_count;
    int relation_tag_count;
    double min_longitude;
    double max_longitude;
    double min_latitude;
    double max_latitude;
};

static int
node_stats (const void *user_data, const readosm_node * node)
{
/* updating OSM Node stats (callback function)*/

    /* casting the USER_DATA pointer to osm_statistics */
    struct osm_statistics *stats = (struct osm_statistics *) user_data;

    stats->node_count++;
    stats->node_tag_count += node->tag_count;
    if (node->latitude != READOSM_UNDEFINED)
      {
```

```
          if (node->latitude > stats->max_latitude)
              stats->max_latitude = node->latitude;
          if (node->latitude < stats->min_latitude)
              stats->min_latitude = node->latitude;
      }
    if (node->longitude != READOSM_UNDEFINED)
      {
          if (node->longitude > stats->max_longitude)
              stats->max_longitude = node->longitude;
          if (node->longitude < stats->min_longitude)
              stats->min_longitude = node->longitude;
      }
    return READOSM_OK;
}

static int
way_stats (const void *user_data, const readosm_way * way)
{
/* updating OSM Way stats (callback function)*/

    /* casting the USER_DATA pointer to osm_statistics */
    struct osm_statistics *stats = (struct osm_statistics *) user_data;

    stats->way_count++;
    stats->way_ndref_count += way->node_ref_count;
    stats->way_tag_count += way->tag_count;
    return READOSM_OK;
}

static int
relation_stats (const void *user_data, const readosm_relation *
       relation)
{
/* updating OSM Relation stats (callback function)*/
    int i;
    const readosm_member *member;

    /* casting the USER_DATA pointer to osm_statistics */
    struct osm_statistics *stats = (struct osm_statistics *) user_data;

    stats->relation_count++;
    for (i = 0; i < relation->member_count; i++)
      {
          member = relation->members + i;
          switch (member->member_type)
            {
            case READOSM_MEMBER_NODE:
                stats->relation_member_node_count++;
                break;
            case READOSM_MEMBER_WAY:
                stats->relation_member_way_count++;
                break;
            case READOSM_MEMBER_RELATION:
                stats->relation_member_relation_count++;
                break;
            };
      }
    stats->relation_tag_count += relation->tag_count;
    return READOSM_OK;
}

int
main (int argc, char *argv[])
{
    const void *osm_handle;
    int ret;
    struct osm_statistics infos;

/* initializing the statistics struct */
    infos.node_count = 0;
    infos.node_tag_count = 0;
    infos.way_count = 0;
    infos.way_ndref_count = 0;
    infos.way_tag_count = 0;
    infos.relation_count = 0;
    infos.relation_member_node_count = 0;
    infos.relation_member_way_count = 0;
    infos.relation_member_relation_count = 0;
    infos.relation_tag_count = 0;
    infos.min_longitude = 180.0;
    infos.max_longitude = -180.0;
    infos.min_latitude = 90.0;
    infos.max_latitude = -90.0;

    if (argc != 2)
      {
          fprintf (stderr, "usage: test_osm2 path-to-OSM-file\n");
```

```
          return -1;
      }

/*
* STEP #1: opening the OSM file
* this can indifferently be an OSM XML encoded file (.osm)
* or an OSM Protocol Buffer encoded file (.pbf)
* the library will transparently perform any required
* action in both cases.
*/
    ret = readosm_open (argv[1], &osm_handle);
    if (ret != READOSM_OK)
      {
          fprintf (stderr, "OPEN error: %d\n", ret);
          goto stop;
      }

/*
* STEP #2: parsing the OSM file
* this task is unbelievebly simple
*
* you are simply required to pass the appropriate
* pointers for callback funtions respectively intended
* to process Node-objects, Way-objects and Relation-objects
*
* the library will then parse the whole input file, calling
* the appropriate callback handling function for each OSM object
* found: please see the callback functions implementing code
* to better understand how it works
*
* important notice: this second example is mainly focused on
* using the USER_DATA pointer. in this example we'll pass the
* address of the osm_statistics struct so to gather some
* general infos.
*/
    ret =
        readosm_parse (osm_handle, &infos, node_stats, way_stats,
                       relation_stats);
    if (ret != READOSM_OK)
      {
          fprintf (stderr, "PARSE error: %d\n", ret);
          goto stop;
      }

/* printing OSM statistics */
    printf ("Longitude    range: %1.7f / %1.7f\n", infos.min_longitude,
            infos.max_longitude);
    printf ("Latitude     range: %1.7f / %1.7f\n\n", infos.min_latitude,
            infos.max_latitude);
    printf ("Nodes              : %d\n", infos.node_count);
    printf ("              tags: %d\n\n", infos.node_tag_count);
    printf ("Ways               : %d\n", infos.way_count);
    printf ("             ndref: %d\n", infos.way_ndref_count);
    printf ("              tags: %d\n\n", infos.way_tag_count);
    printf ("Relations          : %d\n", infos.relation_count);
    printf ("  member.nodes     : %d\n", infos.relation_member_node_count);
    printf ("  member.ways      : %d\n", infos.relation_member_way_count);
    printf ("  member.relations: %d\n", infos.relation_member_relation_count);
    printf ("              tags: %d\n", infos.relation_tag_count);

  stop:
/*
* STEP #3: closing the OSM file
* this will release any internal memory allocation
*/
    readosm_close (osm_handle);
    return 0;
}
```

9.3 test_osm3.c

test_osm3.c shows how to intentionally abort the parser.Here is a typical usage example, parsing an OSM XML file (.osm):

```
./test_osm3 test.osm 10
node#1
node#2
node#3
node#4
node#5
node#6
```

```
node#7
node#8
node#9
node#10
PARSING ABORTED
```

Here is another example, this time parsing a .pbf (Protocol Buffer) OSM file:

```
./test_osm3 test.osm 5
node#1
node#2
node#3
node#4
node#5
PARSING ABORTED
```

```
/*
/ test_osm3.c
/
/ libreadosm Sample code
/
/ Author: Sandro Furieri a.furieri@lqt.it
/
/
      ------------------------------------------------------------------------
/
/ Version: MPL 1.1/GPL 2.0/LGPL 2.1
/
/ The contents of this file are subject to the Mozilla Public License Version
/ 1.1 (the "License"); you may not use this file except in compliance with
/ the License. You may obtain a copy of the License at
/ http://www.mozilla.org/MPL/
/
/ Software distributed under the License is distributed on an "AS IS" basis,
/ WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License
/ for the specific language governing rights and limitations under the
/ License.
/
/ The Original Code is the ReadOSM library
/
/ The Initial Developer of the Original Code is Alessandro Furieri
/
/ Portions created by the Initial Developer are Copyright (C) 2012
/ the Initial Developer. All Rights Reserved.
/
/ Contributor(s):
/
/ Alternatively, the contents of this file may be used under the terms of
/ either the GNU General Public License Version 2 or later (the "GPL"), or
/ the GNU Lesser General Public License Version 2.1 or later (the "LGPL"),
/ in which case the provisions of the GPL or the LGPL are applicable instead
/ of those above. If you wish to allow use of your version of this file only
/ under the terms of either the GPL or the LGPL, and not to allow others to
/ use your version of this file under the terms of the MPL, indicate your
/ decision by deleting the provisions above and replace them with the notice
/ and other provisions required by the GPL or the LGPL. If you do not delete
/ the provisions above, a recipient may use your version of this file under
/ the terms of any one of the MPL, the GPL or the LGPL.
/
*/

#include <stdio.h>
#include <stdlib.h>

#include "readosm.h"

struct osm_helper
{
/* an user defined struct */
    int read_count;
    int stop_limit;
};

static int
eval_abort (struct osm_helper *helper)
{
/* testing the stop limit */
    if (helper->read_count > helper->stop_limit)
        return 1;
    return 0;
}

static int
```

```
parse_node (const void *user_data, const readosm_node * node)
{
/* parsing a Node (callback function)*/

    /* casting the USER_DATA pointer to osm_helper */
    struct osm_helper *helper = (struct osm_helper *) user_data;

    helper->read_count++;
    if (eval_abort (helper))
        return READOSM_ABORT;
    printf ("Node#%d\n", helper->read_count);
    return READOSM_OK;
}

static int
parse_way (const void *user_data, const readosm_way * way)
{
/* parsing a Way (callback function)*/

    /* casting the USER_DATA pointer to osm_helper */
    struct osm_helper *helper = (struct osm_helper *) user_data;

    helper->read_count++;
    if (eval_abort (helper))
        return READOSM_ABORT;
    printf ("Way#%d\n", helper->read_count);
    return READOSM_OK;
}

static int
parse_relation (const void *user_data, const readosm_relation *
       relation)
{
/* parsing a Relation stats (callback function)*/

    /* casting the USER_DATA pointer to osm_helper */
    struct osm_helper *helper = (struct osm_helper *) user_data;

    helper->read_count++;
    if (eval_abort (helper))
        return READOSM_ABORT;
    printf ("Relation#%d\n", helper->read_count);
    return READOSM_OK;
}

int
main (int argc, char *argv[])
{
    const void *osm_handle;
    int ret;
    struct osm_helper helper;

/* initializing the helper struct */
    helper.read_count = 0;
    helper.stop_limit = 0;

    if (argc != 3)
      {
          fprintf (stderr, "usage: test_osm3 path-to-OSM limit\n");
          return -1;
      }

/* setting the stop limit */
    helper.stop_limit = atoi (argv[2]);

/*
* STEP #1: opening the OSM file
* this can indifferently be an OSM XML encoded file (.osm)
* or an OSM Protocol Buffer encoded file (.pbf)
* the library will transparently perform any required
* action in both cases.
*/
    ret = readosm_open (argv[1], &osm_handle);
    if (ret != READOSM_OK)
      {
          fprintf (stderr, "OPEN error: %d\n", ret);
          goto stop;
      }

/*
* STEP #2: parsing the OSM file
* this task is unbelievably simple
*
* you are simply required to pass the appropriate
* pointers for callback funtions respectively intended
* to process Node-objects, Way-objects and Relation-objects
*
```

```
* the library will then parse the whole input file, calling
* the appropriate callback handling function for each OSM object
* found: please see the callback functions implementing code
* to better understand how it works
*
* important notice: this second example is mainly focused on
* using the USER_DATA pointer. in this example we'll pass the
* address of the osm_statistics struct so to gather some
* general infos.
*/
    ret =
        readosm_parse (osm_handle, &helper, parse_node, parse_way,
                       parse_relation);
    if (ret != READOSM_OK)
      {
          fprintf (stderr, "PARSE error: %d\n", ret);
          goto stop;
      }

  stop:
/*
* STEP #3: closing the OSM file
* this will release any internal memory allocation
*/
    readosm_close (osm_handle);
    return 0;
}
```

Index