openGPS®

Introducing the ISO 5436-2 software library in openGPS

Training at

NanoFocus AG

Oberhausen

Johannes Herwig

28-May-2008

This document gives a **short introduction for developers** who want to include openGPS into their applications.

www.opengps.eu

28-May-2008

Content

- The programming interface.
- The most important data types.
- How to access point data.
- How to obtain the ISO 5436-2 XML document.
- How to create a new file.
- How to implement vendorspecific extensions.
- How to handle errors.
- How to include the library in your application.
- Background information on the internal implementation.

C and C++ Programming Interfaces

openGPS®

- Both C and C++ interfaces are provided.
- Lead implementation is in C++, but...
- Each Class Type has its corresponding Abstract Data Type.
- With one exception: direct manipulation of an ISO 5436-2 XML schema instance is possible in C++ only.

- See the different principles in the next two code snippets!

Abstract Data Type vs. Class Type (1)

openGPS®

```
/* Abstract Data Type of an X3P file container. */
#include <opengps/iso5436_2.h>

/* Initially get a handle to an anonymous struct. */
OGPS_ISO5436_2Handle handle =
ogps_OpenISO5436_2(_T("C:\\filename.x3p"));

/* Make some useful changes - omitted here; save them. */
ogps_WriteISO5436_2(handle);

/* Release the obtained resource handle. */
ogps_CloseISO5436_2(&handle);
```

Abstract Data Type vs. Class Type (2)

openGPS®

```
/* Class model of an X3P file container. */
#include <opengps/cxx/iso5436_2.hxx>

/* Create an instance of the ISO 5436-2 model. */
OpenGPS::ISO5436_2 iso5436_2(_T("C:\\filename.x3p"));

/* Read an existing file (instead of creating it). */
iso5436_2.Open();

/* Make some useful changes - omitted here; save them. */
iso5436_2.Write();
```

Important Data Types

OGPS_ISO5436_2Handle **OpenGPS::ISO5436_2**

- Main data type when accessing an X3P file.

OGPS_DataPointPtr **OpenGPS::DataPoint**

- Typesafe handling of one single component of point data.
- *OGPS_Int16, OGPS_Int32, OGPS_Float, OGPS_Double*

OGPS_PointVectorPtr **OpenGPS::PointVector**

- Handles a three-dimensional vector of point data.

OGPS_PointIteratorPtr **OpenGPS::PointIterator**

- Iterates all point data stored within an X3P file.

Accessing Point Data

openGPS®

- X3P can handle point data of different data types:
- OGPS_Int16, OGPS_Int32, OGPS_Float, OGPS_Double.
- Also X3P can contain **line and area** data.
- **Area data** may address **invalid or missing** points.
- There are **many ways** in openGPS **to access point data**.
- On the one hand it is possible for you to **ignore all these complicated facts** when reading X3P **using an iterator**.
- On the other hand there are ways to **break it all down**!

Accessing Point Data – Iterator (1)

- Read only point data from X3P files.
- Do not care about the exact data type.
- Do not care whether it is line or area data.
- If it is area data, read row by column by depth.
- If a point is invalid or missing, it equals to null.
- Assume that all point data is of type double.

- See the full featured sample code in C hereafter!

Accessing Point Data – Iterator (2)

openGPS®

```
OGPS_PointVectorPtr vector = ogps_CreatePointVector();
OGPS_ISO5436_2Handle handle = ogps_OpenISO5436_2(_T("C:\\filename.x3p"));

OGPS_PointIteratorPtr iterator = ogps_CreateNextPointIterator(handle);
while(ogps_MoveNextPoint(iterator)) { /* Iterate all point data. */
  ogps_GetCurrentPoint(iterator, vector); /* Read point data into a buffer. */

  double x = 0.0, y = 0.0, z = 0.0;
  if(ogps_IsValidPoint(vector)) { /* Check whether this is valid point data. */
    ogps_GetXYZ(vector, &x, &y, &z); /* Get single components of point data. */
  }
}

ogps_FreePointIterator(&iterator);
ogps_CloseISO5436_2(&handle);
ogps_FreePointVector(&vector);
```

Accessing Point Data – Indexing (1)

- Using an **index based** method, you **must care** whether it is **line or area** data, due to different indexing schemes.
- Also you must know **list or matrix dimensions** for scope.
- Therefore you need to **access** the **XML structure** at first.
- **Hint:** This in turn **forces** you to **C++ linkage**, since **XML structure** is **C++ only**. The XML tree won't change very often, so the C API provided is still preferable.

- The following code examples sketch the overall procedure.

Accessing Point Data – Indexing (2)

- Obtain a pointer to the inner **XML document structure**.

```
/* Get the document using the C programming interface.
   However, the document itself is accessible as a C++ model only! */
using namespace OpenGPS::Schemas::ISO5436_2;

const ISO5436_2Type* const document = ogps_GetDocument(handle);
```

```
/* Get the document using the C++ programming interface. */
using namespace OpenGPS::Schemas::ISO5436_2;

const ISO5436_2Type* const document = iso5436_2.GetDocument();
```

Accessing Point Data – Indexing (3)

openGPS®

- Obtain either **list or matrix dimensions** to stay in scope.
- Get it from the **XML document structure** directly.

```
/* Is line or area data? */
if(document->Record3().ListDimension().present()) {
    unsigned long max = document->Record3().ListDimension().get();
    /* Index your point data - not shown here. */
} else if(document->Record3().MatrixDimension().present()) {
    unsigned long max_x, max_y, max_z;
    max_x = document->Record3().MatrixDimension().get().SizeX();
    max_y = document->Record3().MatrixDimension().get().SizeY();
    max_z = document->Record3().MatrixDimension().get().SizeZ();
    /* Index your point data - not shown here. */
}
```

Accessing Point Data – Indexing (4)

openGPS®

- Examples of indexing functions **accessing line data**:

```
/* 1.   Get the fully transformed value of a data point.
        Also do not care about the exact data type. */
double x, y, z;
ogps_GetListCoord(handle, index, &x, &y, &z);
```

```
/* 2.a Get the raw point data. Do not apply any point transformation. */
OGPS_PointVectorPtr vector = ogps_CreatePointVector();
ogps_GetListPoint(handle, index, vector);

/* 2.b Access the X component of the three-vector as Int32. */
int value = ogps_GetInt32X(vector);
ogps_FreePointVector(&vector);
```

Accessing Point Data – Indexing (5)

openGPS®

- Examples of indexing functions **accessing area data**:

```
/* 1.  Get the fully transformed value of a point data.
       Also do not care about the exact data type. */
double x, y, z;
ogps_GetMatrixCoord(handle, index_u, index_v, index_w, &x, &y, &z);
```

```
/* 2.a Get the raw point data. Do not apply any point transformation. */
OGPS_PointVectorPtr vector = ogps_CreatePointVector();
ogps_GetMatrixPoint(handle, index_u, index_v, index_w, vector);

/* 2.b Access the X component of the three-vector as Int32. */
int value = ogps_GetInt32X(vector);
ogps_FreePointVector(&vector);
```

Creating X3P Archives (1)

- Creating an X3P file comprises these two major steps:
  - Instantiate the **ISO 5436-2 XML document**.
  - Add **point data** conforming to that instance.
- Instantiating the ISO 5436-2 XML document comprises:
  - Setting **axes definitions** and their **coordinate transformations**.
  - Axes can be of either **absolute or implicit** type.
    - If **absolute**, data points **must explicitly provide values** for that dimension.
    - If **implicit**, data points **must not provide values** for that coordinate dimension. Instead these are derived from indexing techniques.
  - **Note:** the **Z axis** must **always** be of **absolute** type!
  - Also an axis defines the **data type of its coordinate values**:
    - ***I**nteger (OGPS_Int16), **L**ong (OGPS_Int32), **F**loat (OGPS_Float), **D**ouble (OGPS_Double)*.
  - Adding required **metainformation** about the **measurement process**.

Creating X3P Archives (2)

```
using namespace OpenGPS::Schemas::ISO5436_2;

/* Assemble the x axis; others are omitted here. */
AxisType xaxis_type(AxisType::I); /* Incremental */
DataType xdata_type(DataType::D); /* Double */
AxisDescriptionType xaxis(xaxis_type);
xaxis.DataType(xdata_type);

/* Assemble coordinate system. */
AxesType axes(xaxis, yaxis, zaxis);

/* Instantiate Record1 of the ISO 5436-2 XML specification. */
FeatureType surface(_T("SUR")); /* Area data */
Record1Type record1(_T("ISO5436 - 2000"), surface, axes);
```

Creating X3P Archives (3)

openGPS®

- Finish creation of a rudimentary ISO 5436-2 XML document.
- Having the XML document, obtain the handle to a new X3P file.
- Then proceed with adding the measured point data.

```
/* Setting dimensions of measured topology. */
MatrixDimensionType matrix(max_u, max_v, max_w);

/* Creating the X3P container. Use surface topology.
   (Instantiation of Record2 was omitted here.) */
OGPS_ISO5436_2Handle handle =
        ogps_CreateMatrixISO5436_2(
           _T("C:\\filename.x3p"), NULL, record1, record2, matrix
        );
```

Creating X3P Archives (4)

openGPS®

```
/* Create a buffer to shuffle point data. */
OGPS_PointVectorPtr vector = ogps_CreatePointVector();

/* Fill the three-vector with point data.
   Omit the X axis, because here it is an incremental axis. */
ogps_SetInt32Y(vector, 42);              /* Y axis has data type L. */
ogps_SetDoubleZ(vector, 192.168);    /* Z axis has data type D. */

/* Finally the data point is added at position u, v, w. */
ogps_SetMatrixPoint(handle, u , v, w, vector);

/* Write the newly created X3P file to disk. */
ogps_WriteISO5436_2(handle);

/* Release resources. */
ogps_CloseISO5436_2(&handle);
ogps_FreePointVector(&vector);
```

Vendorspecific Extensions (1)

- An X3P file is essentially a **ZIP archive**, therefore...
- Vendorspecific Extensions is simply a matter of **adding new files** containing arbitrary data of your choice!
- Additionally one must provide a **unique identifier** to find out if the X3P opened is actually an extended version of yours, ...
- e.g.: http://www.mycompany.com/myextension/version01.
- **Drawback**: No two extensions with different vendor URIs!

Vendorspecific Extensions (2)

- This example shows how to **add a vendorspecific file**.
- Then it is shown how to **extract that file** back from an X3P file **to a temporary working directory**.

```
/* Create a file containing vendorspecific data. */
std::wofstream ext_data(_T("C:\\extension.dat"));
ext_data << _T("Hello World.") << std::endl;
ext_data.close();
```

Vendorspecific Extensions (3)

```
/* Create an ISO5436_2 X3P container. */
OGPS_ISO5436_2Handle handle = ...;

/* Add some point data - omitted here;
then add your extension file data. */
ogps_AppendVendorSpecific(handle,
        _T("http://www.mycompany.com/myextension/version01"),
_T("C:\\extension.dat"));

/* Save and release resources. */
ogps_WriteISO5436_2(handle);
ogps_CloseISO5436_2(&handle);
```

Vendorspecific Extensions (4)

```
/* Open an ISO5436_2 X3P container. */
OGPS_ISO5436_2Handle handle = ...;

/* Try to read your vendorspecific extension file data. */
if(!ogps_GetVendorSpecific(handle,
        _T("http://www.mycompany.com/myextension/version01"),
        _T("extension.dat"), _T("C:\\tmp\\extension.dat"))) {
  std::cerr << "No vendorspecific extensions." << std::endl;
}

/* Release resources. */
ogps_CloseISO5436_2(&handle);
```

Error Handling (1)

openGPS®

- The **C++** interface provides an **exception mechanism**.
- The **C** interface provides **ogps_GetErrorId()** instead.
- Error conditions are **subdivided into categories**, each category has its **unique identifier**:
- *OGPS_ExNone, OGPS_ExGeneral, OGPS_ExOverflow, ...*
- Every message features a **brief text** naming its category and a **detailed description** with helpful instructions.
- Exceptions of category ***OGPS_ExWarning*** **can safely be ignored**. However there must have been a try-catch-clause.

- Exemplary source code follows.

Error Handling (2)

- Error handling in C++ and C programming interface:

```
try {
    /* Insert your code here. */
} catch(const OpenGPS::Exception& ex) {
    std::cerr << ex.text() << std::endl << ex.details() << std::endl;
}

/* Insert your code here. */
if(ogps_HasError()) {
    const OGPS_ExceptionId id = ogps_GetErrorId();
    std::cerr << ogps_GetErrorMessage() << std::endl <<
                 ogps_GetErrorDescription() << std::endl;
}
```

Usage Within Your Project - Binary Linkage

openGPS®

- **Static** and **shared** library versions are readily available.
- Also there are both **release** and **debug** builds included.
- Throughout the library **unicode strings** are used only.

- How to **link** the **static** version **with your application**:
  - Link with **ISO5436_2_XML.lib** (release build) resp. **ISO5436_2_XML_D.lib** (debug build).

- How to **link** the **shared** version **with your application**:
  - Define the preprocessor macro **SHARED_OPENGPS_LIBRARY**.
  - Link with **ISO5436_2_XML_S.lib** (release build) resp. **ISO5436_2_XML_DS.lib** (debug build).
  - Use this strategy when linking with properitary software!

Usage Within Your Project - Runtime Issues

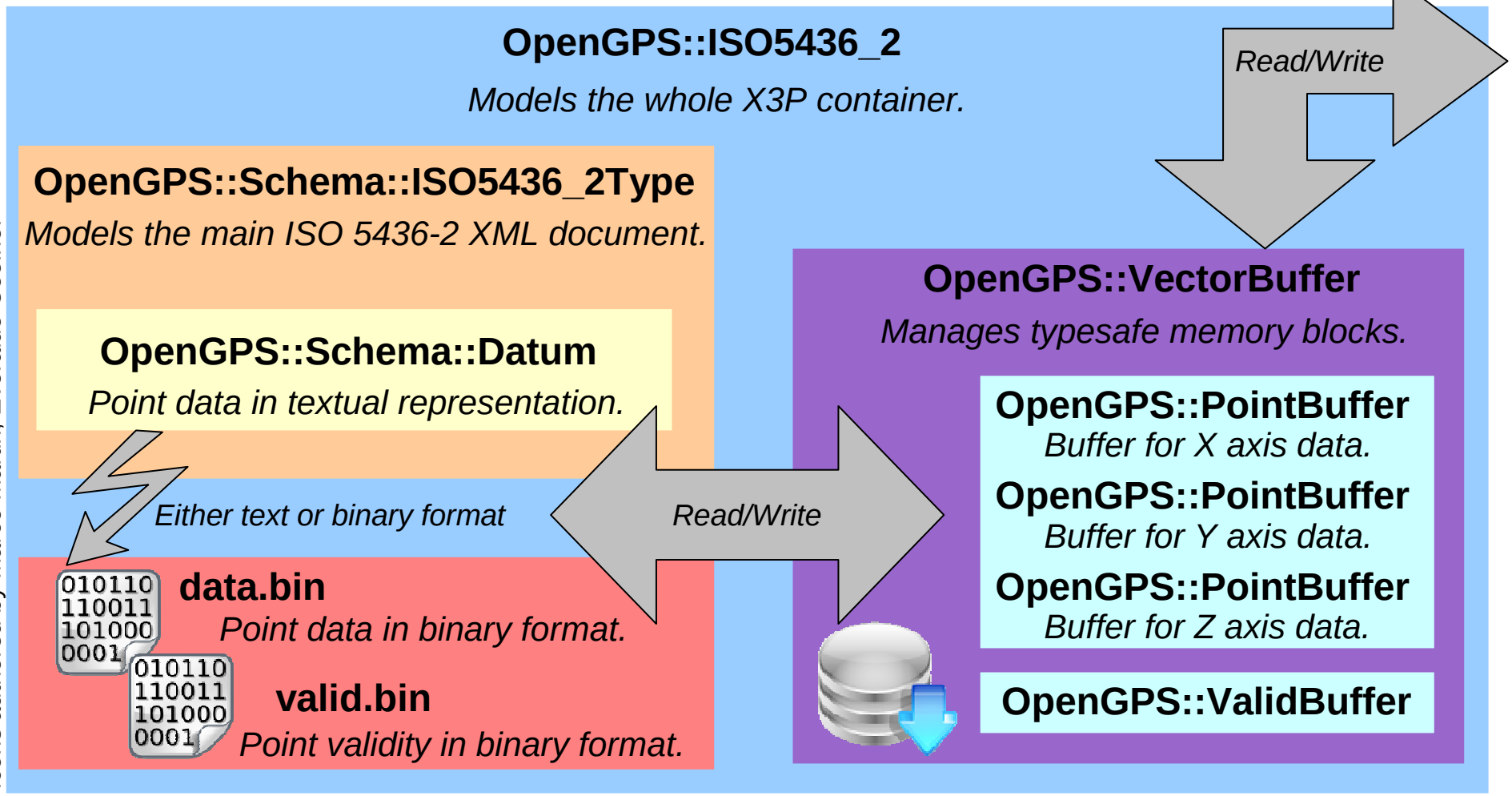
- The library needs runtime access to its **configuration files**.
- Therefore an **environment variable** pointing to the local openGPS installation is questioned when the application is running.
- Set **OPENGPS_LOCATION="<Path to openGPS>"**.
- When using openGPS with your customers, do not forget to **extend your setup routine** accordingly.
- On shared linkage also make sure **ISO5436_2_XML.dll** resp. **ISO5436_2_XML_D.dll** is within your **PATH.**

Behind the Scenes - Goals, Problems, Solutions

openGPS®

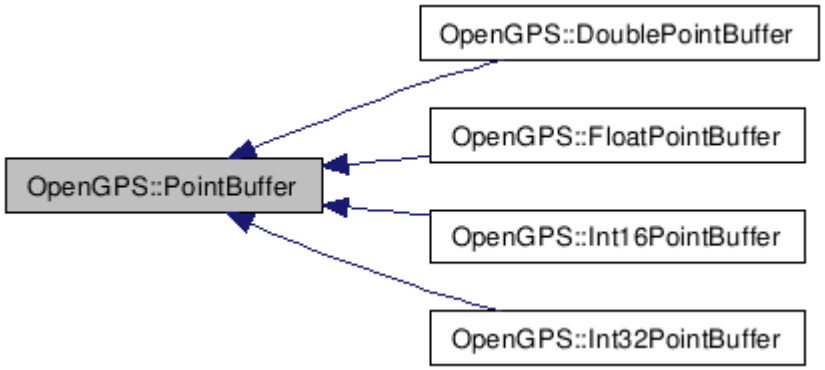
- Possibly large amount of point data.
  - Store point data in an external binary file rather than in XML directly.
  - For small amount of data XML is perfectly OK for readability, though.
- Need for efficient and transparent read/write access.
  - Access XML (text processing) or a binary file both in the same way.
  - Address the problem of Endianess on different architectures.
- Combination of possible axes data types is huge.
- But typesafe access to point data is required.

Behind the Scenes - Point Data in Memory

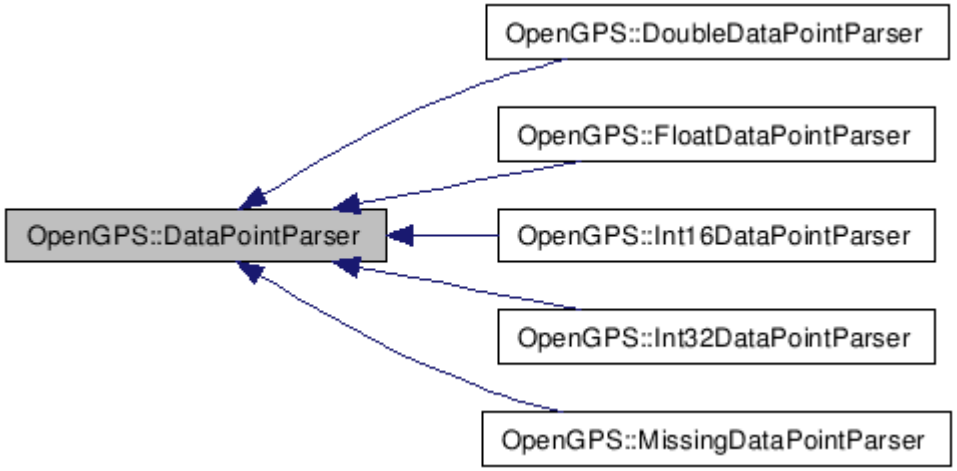


Icons authored by Marco Martin, Everaldo Coelho.

Behind the Scenes - Store and Parse Point Data (1)

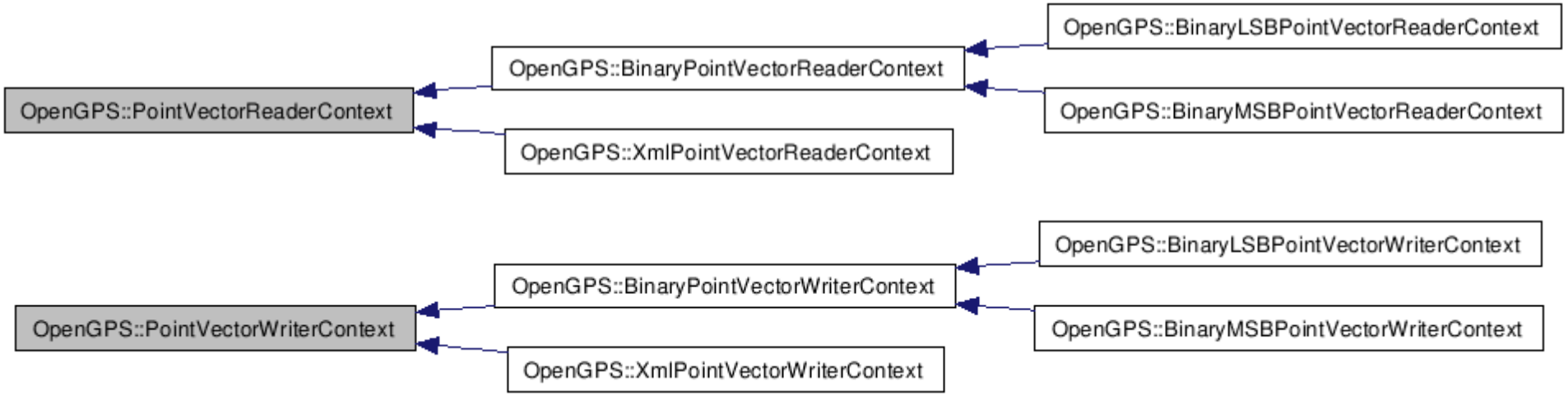


- Specific implementations of typed point buffers for every possible data type.



- Generic methods for reading and writing typed point data to resp. from arbitrary storage media.

Behind the Scenes - Store and Parse Point Data (2)



- A *Context ties the somewhat loosely DataPointParser (in all three dimensions) to specific storage media.
- Here the media are:
  - **Datum Tags** within an ISO 5436-2 XML.
  - External binary file **data.bin**.

Acknowledgement

- The implementation of ISO5436-2 X3P data format and this training document have been gratefully sponsored by NanoFocus AG, Germany

- www.nanofocus.de

openGPS®

www.opengps.eu

End

28-May-2008