

# Blom SDK Reference v2.8r0.0

## Blom ASA

---

Audience: **BLOM ASA partners and developers. Confidential**

Abstract: This reference describes low level functionalities and details of the Blom SDK.

Date	Blom Document
November 2011	Blom SDK Reference.doc

## **Notices**

Blom expressly retains all intellectual and other property rights with respect to this document and all matters set forth herein.

Some technical assertions of capability included herein are estimates based on limited information gathered from past experience.

The terms, conditions, specifications, and procedures described herein are subject to change in the sole discretion of Blom ASA and its affiliates. End-User is responsible for requesting and obtaining the latest release of these terms, conditions, specifications, and procedures prior to any purchase or deployment of the products described herein.

## **Confidentiality**

This document and the information contained herein is the proprietary and confidential information of Blom, ASA. It is provided under contract agreement, and may not be reproduced or used for purposes outside the scope of such agreement.

## **Trademarks**

Blom's logo is a registered trademark of Blom, ASA in the Kingdom of Norway and other countries. Other brands and their products are registered trademarks or trademarks of their respective holders and should be noted as such.

## **Copyrights**

© 2009, Blom, ASA • All Rights Reserved

## Revision History

Document Number	Issue Date	Editor	Reason for Change
BSDK_2800	November 2011	Blom	Update to Blom SDK
BU3D_DP_1610	September 2010	Blom	Update to library version 1.6r1.0a
BU3D_DP_1300	January 2010	Blom	Update to library version 1.0.0.0
BU3D_DP_1200	December 2009	Blom	Update to library version 0.5.0.0
BU3D_DP_1100	October 2009	Blom	Update to library version 0.4.1.0
BU3D_DP_1030	June 2009	Blom	Revision 3
BU3D_DP_1010	April 2009	Blom	Original Document

## Table of Contents

<b>1</b>	<b>Scope</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>2</b>
<b>3</b>	<b>BlomURBEX™</b>	<b>3</b>
<b>4</b>	<b>Blom SDK Reference</b>	<b>5</b>
4.1	System requirements	5
4.2	Global overview	5
4.2.1	Blom Datasets	5
4.2.2	Tiled structure	7
4.2.3	Multi-resolution DTM	7
4.2.4	Tile coordinates and quadtree keys	8
4.2.5	Blom SDK namespace	10
4.2.6	Classes overview	10
4.2.7	Data types and character encoding	13
4.2.8	General procedure of data retrieving	15
<b>5</b>	<b>Blom SDK global functions</b>	<b>17</b>
5.1	Initialization and destruction functions	17
5.2	Meta data functions	18
5.3	Data loader	19
5.4	Blom library licensing	19
5.5	GetLastDataSourceUsed	21
<b>6</b>	<b>Implementing BAttributesReceiver</b>	<b>22</b>
<b>7</b>	<b>Blom SDK event viewer</b>	<b>25</b>
<b>8</b>	<b>Blom SDK Proxy Settings</b>	<b>27</b>
<b>9</b>	<b>Data loader – Initializing and managing data sources</b>	<b>29</b>
9.1	Creating a data loader	29
9.2	Data sources	29
9.3	Adding BlomURBEX server as data source	30
9.4	Adding Blom library as data source	31
9.5	Get default latitude and longitude	31
<b>10</b>	<b>Ortho imagery loading</b>	<b>33</b>
10.1	Load ortho base and overlay	33

10.2	Load map portion	34
<b>11</b>	<b>Digital terrain model loading</b>	<b>37</b>
<b>12</b>	<b>Urban 3D models loading</b>	<b>40</b>
12.1	Overview	40
12.2	Model loading	41
12.3	Meshes loading	43
12.4	Shapes loading	44
12.5	Building texture loading	44
12.6	Models and shapes attributes	45
<b>13</b>	<b>Discrete imagery loading</b>	<b>47</b>
13.1	Find discrete images by extent	47
13.2	Find nearest discrete images to a location	48
13.3	Find all discrete images at a particular location	50
13.4	Load discrete image tile	52
13.5	Discrete images metadata	53
13.6	Discrete images calculations	53
<b>14</b>	<b>Calculations</b>	<b>54</b>
14.1	Calculate elevation	54
14.2	Calculate ground length	54
<b>15</b>	<b>LiDAR data loading</b>	<b>56</b>
15.1	LoadLidar	56
15.2	LoadLidarArea() [For Circle]	56
15.3	LoadLidarArea() [For Extent]	57
<b>16</b>	<b>Get available overlays</b>	<b>59</b>
<b>17</b>	<b>Geocoding with Blom SDK</b>	<b>61</b>
17.1	Geocoding	61
17.2	Reverse geocoding	62

## **1 Scope**

This document gathers the features and environmental circumstances of the Blom SDK. It aims to describe key features of such technology in order to understand its strengths and possibilities for mobile, navigation and web scenarios.

Blom SDK (which comes in C++ and .NET version) is a software library that enables fast development and integration of consumer and professional services and applications based on *Blom* data served by BlomURBEX server and Blom local library.

Items covered are functionalities, interface and delivered data models.

## 2 Introduction

BlomURBEX™ is a geographic information server (geoserver) designed to offer fast, simple access to geospatial models through an extensive set of standardized interfaces on which multiple value-added services can be offered.

With the popularization of the Internet, organisations are increasingly using and depending on geospatial data for their daily activities. However, the processing and management of this information rarely forms part of the company's *core business*. It is therefore necessary to be able to server geospatial data efficiently to applications, organizations, and consumers, so that it can be separated from:

- The location and negotiation of source data
- Upgrades and maintenance
- Problems intrinsic to formats, projections, metadata, etc.

BLOM is aware of this need and seeks to offer its clients a proposal that is clearly distinct from the other offers on the market, complementing it with value-added services that in turn distinguish it even more from the competition. With this in mind, BLOM launched its BlomURBEX™ geospatial service platform in 2008

BlomURBEX™ strengths and features enable incredibly fast development of consumer, professional services and applications enriched with Blom's datasets. This means that BlomURBEX™ includes not only basic cartography services, but also makes additional services possible that allow high added value to be incorporated into the customer site or applications.

When the data amount is huge, as it is the case with the Blom geodata models it is cumbersome for end applications to host its own server and data containers in most cases.

In other cases, such as mobile services, supporting on-board data for big areas is hard and expensive. It would require huge memory sizes, which would increase the device price considerably, and the distribution processes are quite complicated. In most cases, it is simply impossible to load all the data embedded on the device, so an on-line solution is the only viable approach.

There are also many cases where providing online access is not a viable option due to access or availability restrictions, such as in emergency services. To this end Blom has also generated a set of libraries for offline delivery of geodata with all the capabilities and features available on our online services.

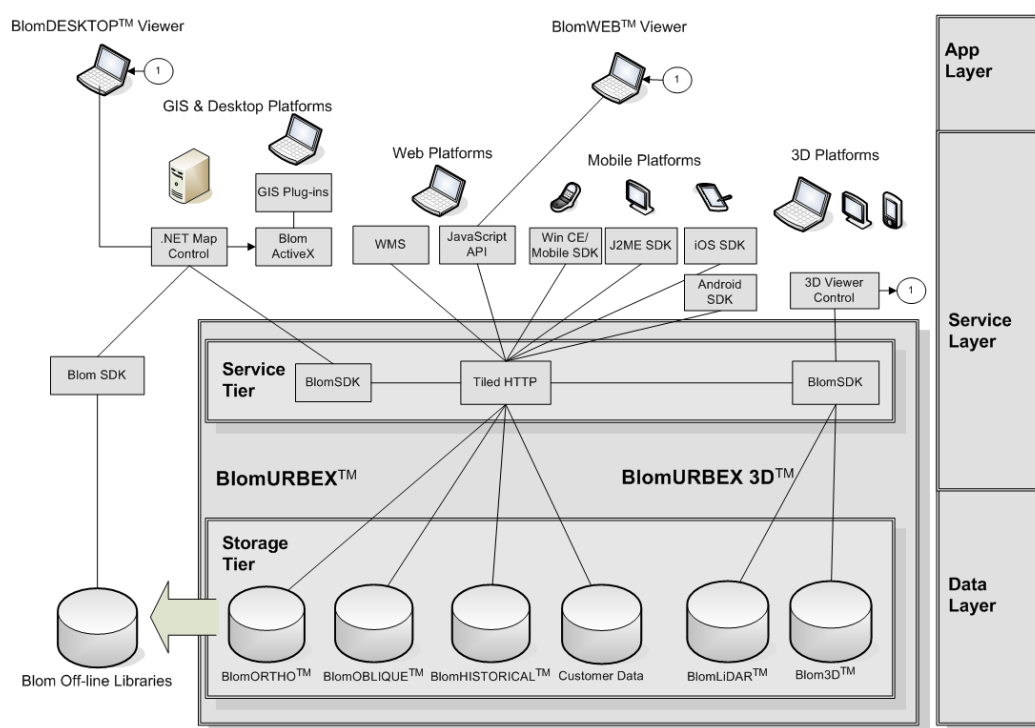
To optimally access all the available datasets, whatever the access method, Blom provides a proper Software Development Kit (SDK) that allows a fast integration and development of the final application.

This document describes that SDK.

### 3 BlomURBEX™

BlomURBEX™ is an online service providing the needed technology to host and deliver different datasets of spatial geodata with enough storage capability and process power to meet the needs of market and industry standards.

The service is arranged in three tiers: A data tier holding the data weight, a service tier making data accessible to users from BlomURBEX Servers or from delivered local Blom Libraries, and the Apps tier that consist in the different front end applications as seen below. There is also a set of capabilities to manage data offline in the form of Blom Libraries.



The service tier offers to final applications an access point to the different data components. The service methods are mainly tile-oriented, in order to make data referencing, search and delivery processes easier, avoiding processing at service time.

Transport is mainly relied on HTTP protocol, specifically optimized into manageable packets for the 3D streaming service.

For more information on BlomURBEX™, Blom Geoserver solution, please refer to the **BlomURBEX Product Description**.

Besides the easy and fast integration, by using an online solution, the client device is released from needing the big storage sizes required to handle real scenarios. The online viewers allow the user to navigate through all the areas covered by *Blom*, and to always access the most recent data.

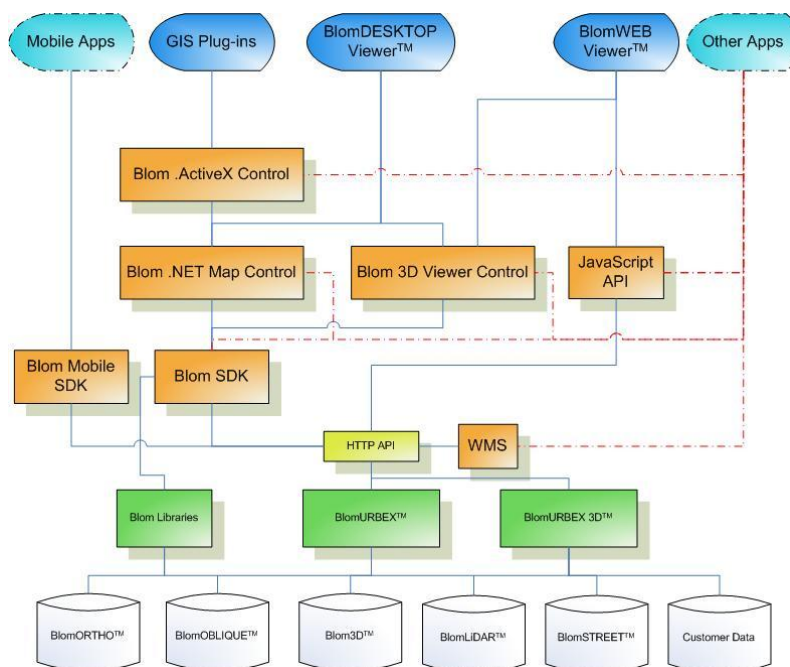


The solution is a progressive model, which avoids delivering large amounts of data in one bundle, providing fast response and saving bandwidth.

All available data (ortho and oblique images, DTM, 3D and LiDAR models, and rasterized vector datasets) are cut and packed into manageable tiles for easy delivery. Textures in 3D models are compressed into several quality levels in order to allow progressive degradation, keeping under manageable limits the size of the transmitted and rendered data.

In the case of Blom Libraries, they are composed of a set of Blom File System packages into a directory tree, packed and encrypted in search of the same goals of easy delivery and performance.

Blom SDK is a low level set of tools provided to make transparent the protocol details to the application developer. It provides several convenient capabilities to the programmer, such as local data caching and preloading allowing full control over all Blom datasets both online and offline. The following diagram details the family of Blom development tools and the location of Blom SDK in this family.



As seen, the Blom SDK is a low level product constructed over the core HTTP API. It provides developers full control over all aspects of the datasets, and it can be used to access data located in BlomURBEX™/BlomURBEX 3D™, as well as off-line data delivered via Blom Libraries. In addition, another set of high levels components is constructed on top of this SDK. These components are described in their own documents.

## **4 Blom SDK Reference**

### **4.1 System requirements**

- Blom SDK .NET version has been designed to run inside managed .NET applications.

The library can only be run under Microsoft Windows environment with .NET Framework 2.0 or higher installed.

- Blom SDK C++ version has been designed to run inside unmanaged C++ applications.

Target environments are Desktop Windows, Windows CE, Symbian and Sony Playstation platforms.

### **4.2 Global overview**

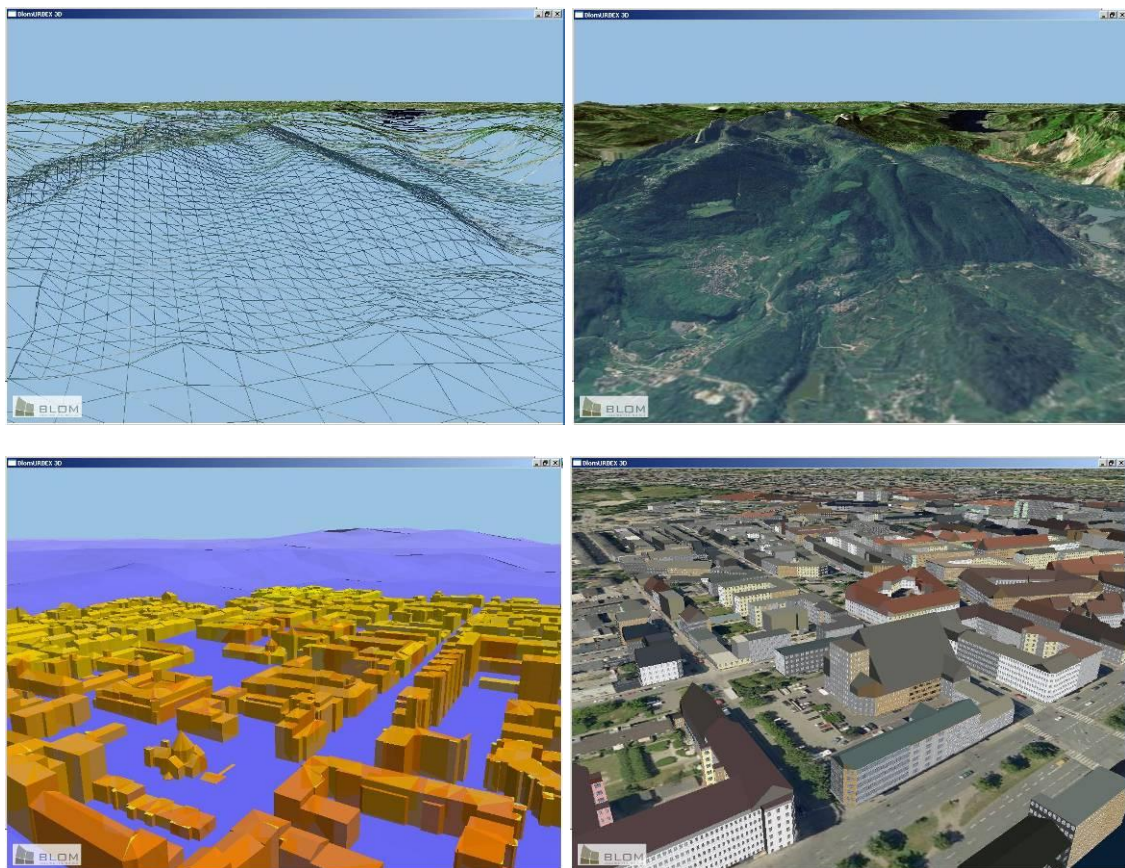
#### **4.2.1 Blom Datasets**

Blom datasets, available via BlomURBEX™ or Blom Libraries include:

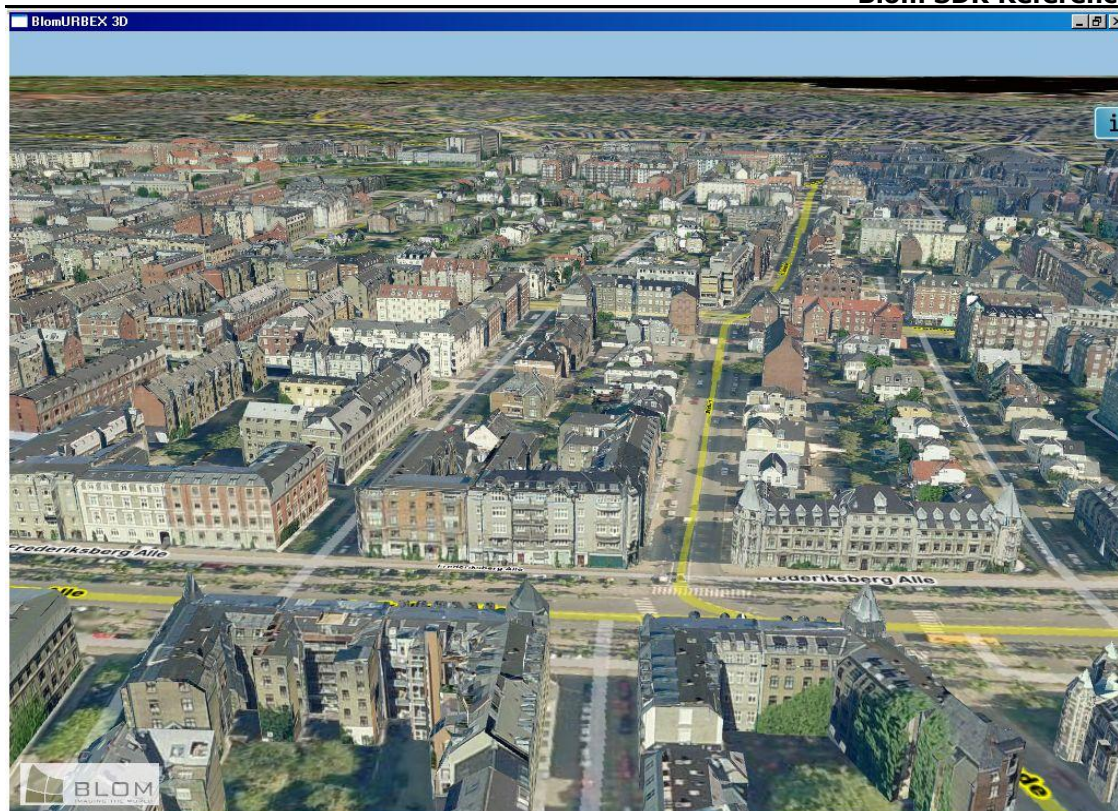
- **Vertical (Ortho) images:** continuous projected mosaic of vertical views. High resolution (0,1 to 0,5m) aerial imagery establishing a seamless image view of an area.
- **Oblique images:** discontinuous set of pictures composed of single images covering all spots from 5 different angles (North, South, East, West and vertical). Resolution 10-18 cm, accuracy 1-4m.
- **Ortho-rectified oblique:** continuous projected mosaic of oblique views providing a seamless oblique view from 4 angles, North, South, East and West.
- **Rasterized vector data:** An overlay of rasterized vector data that can be displayed on top of ortho and oblique layers. By default, TeleAtlas street maps are offered as an option, but also custom vector maps can be imported.
- **Digital Terrain Models**
- **Buildings**
  - Wire structures in several levels of detail.
  - Texture libraries with different detail levels
    - Opaque plain colour

- Parametric patterns
- Photorealistic images
- **LIDAR data:**
  - Raw data extracted directly from our LIDAR cameras that let the customer deal with the most accurate georeferenced data.
- **Third-party data:** In addition, BlomURBEX™ can also host third-party datasets and also server customer data with the same performance and privacy features as provided for Blom own data.

The following images show some examples of these elements:







### 4.2.2 Tiled structure

All the data available is cut into tiles and available at several zoom levels, to allow for fast data delivery, reducing process time on the server side.

Tiles are very useful to allow high performance and to reduce the information volume sent through the internet, since a smart client application will be able to request just the tiles needed in a particular moment to represent a particular extent.

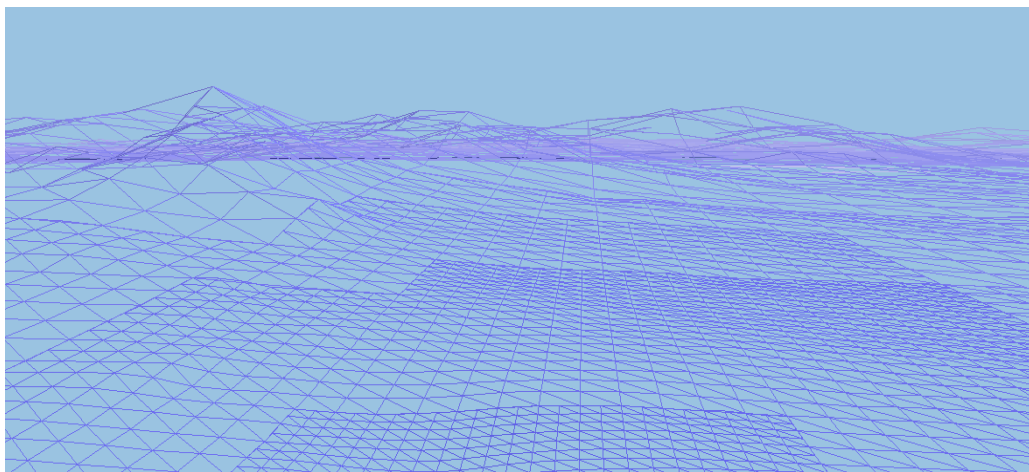
Applications can also do caching of these tiles, so when a user is panning through an area if he/she navigates back and forward, the same tiles won't be requested several times, but just once. This gives fast response times, better user experience, lower bandwidth consumption and places less stress on the servers.

Several tiling levels are arranged, so different detail level can be adjusted to each data type. This is specifically relevant for DTMs, ortho-photos and texture qualities.

### 4.2.3 Multi-resolution DTM

Digital Terrain Models are arranged as grids of heights, varying from low resolution to highly precise grids. The application can control which detail level to use depending on the camera elevation above the ground, the complexity of the model and the performance of the particular graphics library or hardware.

Several resolutions of DTM are suggested to be used in the same scenario, displaying high resolutions in the near ranges and allowing progressive degradation for distant areas, controlling the amount of data while preserving visual appearance.



#### 4.2.4 Tile coordinates and quadtree keys

All tiles from Blom SDK, including ortho images, DTM sections and urban 3D models, are served using Spherical Mercator projection (EPSG:3785). This projection is used because it can be used almost world-wide (poles are not covered) and does not alter the aspect of features, though it scales up areas far from the equator.

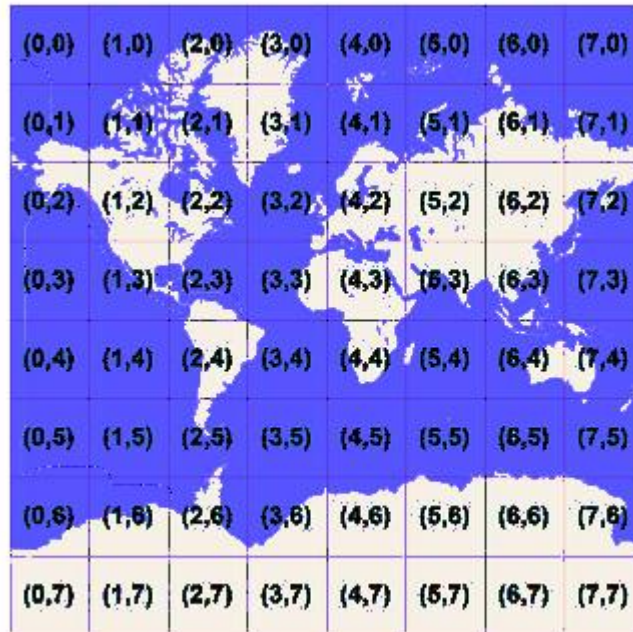
As mentioned, poles are not covered, the range of Spherical Mercator in latitude coordinate covers from  $-85.05112878^\circ$  up to  $85.05112878^\circ$  in WGS84 Lat/Long.

The Earth surface is considered as divided in grids, each one consisting on several squared sized tiles. Different zoom levels imply different grid sizes and so, different tiles.

Each individual tile has a unique code used to identify it uniquely, the quadtree key (qtreekey). Given one pair of coordinates and one zoom level only one tile will be covering that point.

For each zoom level, the whole world surface is divided in tiles. For every zoom level the earth surface is divided in  $(2^{\text{level}} \times 2^{\text{level}})$  tiles. Each tile is given XY coordinates ranging from (0, 0) in the upper left to  $(2^{\text{level}-1}, 2^{\text{level}-1})$  in the lower right.

**Example:** At level 3, tiles will have  $2^3 \times 2^3$ , so tile coordinates range from (0, 0) to (7, 7) as shown:



To optimize the indexing and storage of tiles, the two-dimensional tile XY coordinates are combined into one-dimensional strings called quadtree keys. Each quadtree key uniquely identifies a single tile at a particular zoom level.

To convert tile coordinates into a quadtree key, the bits of the Y and X coordinates are interleaved, and the result is interpreted as a base-4 number (with leading zeros maintained) and converted into a string.

**Example:** Given tile XY coordinates of (7, 5) at level 3, the quadtree key is determined as follows:

Tile X = 7 = **111** (binary)

Tile Y = 5 = **101** (binary)

Quadtree key = **110111** (binary) = 313 (base-4) = Quadtree key string = "313"

The length of a quadtree key (the number of digits) equals the level of detail (zoom level) of the corresponding tile.

The quadtree key of any tile starts with the quadtree key of its parent tile (the containing tile at the previous level).

**Example:** Tile 2 is the parent of tiles 20 through 23, and tile 11 is the parent of tiles 110 through 113:

### 4.2.5 Blom SDK namespace

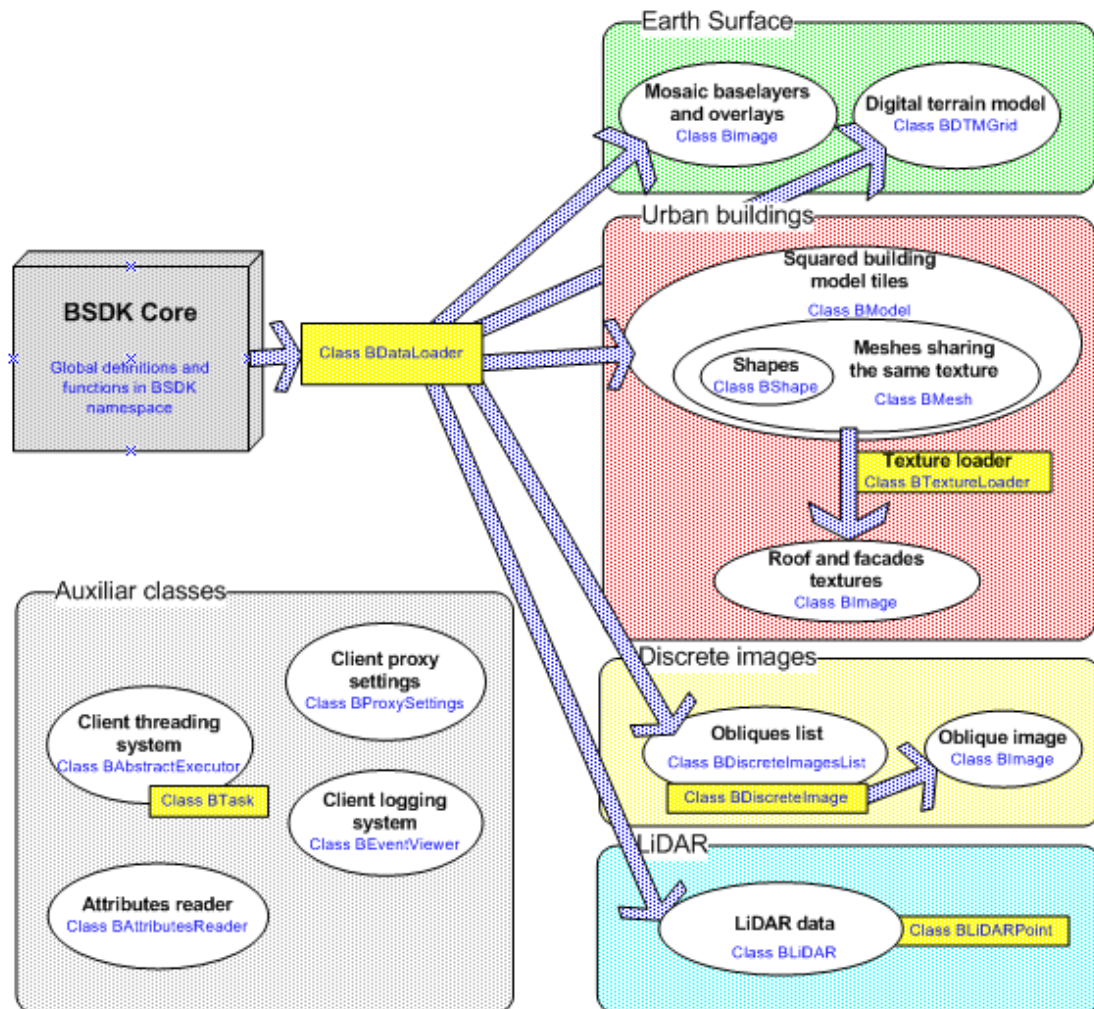
C++ applications: All Blom SDK interface members are encapsulated in [BSDK](#) namespace.

.NET applications: All Blom SDK interface members are encapsulated in [BSDKW](#) namespace. Never use any class, function or enumeration out of this namespace, they are for internal use even if they seems to be public, and they can be changed or even disappear in next versions.

### 4.2.6 Classes overview

This library provides client application all the necessary stuff to make a fast and easy BlomURBEX and BlomURBEX 3D Servers integration. The client application interface contains a set of classes as shown in the following graph and a set of global functions.





White ellipsoidal forms are abstract classes that the client application should implement, while yellow forms represent concrete classes implemented in the library that should not be overridden nor instanced by user.

`BEventViewer` is a class defined to retrieve information about BSDK internal functionality, currently it is ready to inform client about connectivity problems, also, it may be linked by BSDK client to a logger for general debugging purposes.

`BAbstractExecutor` must be implemented by client application to execute BSDK library heavy tasks in background or during idle activity, these tasks will be encapsulated in `BTask` objects and delivered to the client.

**Note:** class `BAbstractExecutor` is not available in .NET version of BSDK. Multi-Threading requirements are taken care internally.



[BProxySettings](#) must be implemented by client application to configure proxy settings. These proxy settings should be collected by the application, for example, searching system configuration or asking directly to the user for this information using pop-ups.

[BDataLoader](#) is a class implemented to let client application login and retrieving data from BlomURBEX server, add/remove datasources, fetch data from the added datasources, find discrete images, perform geocoding/reverse geocoding, find discrete image, load ortho overlays, etc.

Types of supported datasource are: BlomUrbex Server and local *Blom* libraries. To access data from *Blom* libraries, a valid license is needed. License can be for a single library or it may provide access to all *Blom* libraries. License can be for certain time period or they may be valid for always. Some Blom SDK features requires license to work: It may happen that a certain *Blom* library is licensed but the data measurement features are not licensed. So client applications with no license will not be able to access those data measurement features.

With the exception of [BTextureLoader](#), the rest of classes are used to represent BlomURBEX data.

- Earth surface objects: ortho-photos and road overlay images ([BImage](#)), and the digital terrain model ([BDTMGrid](#)).
- Building objects: tiles ([BModel](#)), meshes ([BMesh](#)), shapes ([BShape](#)) and building textures ([BImage](#)).
- Obliques imagery: obliques list ([BOblique](#))
- [BLiDAR](#) is class that is capable of receiving a set of LiDAR point. Client application may implement on this class to receive the LiDAR point and then can use the LiDAR point information as per their requirement.
- [BDiscreteImage](#) is a class let client download tiles and metadata from discrete images. This class provides many attributes of discrete images and calculations regarding discrete images.

[BTextureLoader](#) is an auxiliary class used to make possible load models and facade textures in background by the user.

## 4.2.7 Data types and character encoding

### 4.2.7.1 Orientation

Imaginary orientation in BlomUrbex 3D™ Data Provider Software Development Kit is represented by the enum `BLorientation`:

```
typedef enum [C++ Code]
{
    BL_ORIENTATION_NORTH      = 0x6E,      // 'n'
    BL_ORIENTATION_SOUTH      = 0x73,      // 's'
    BL_ORIENTATION_EAST        = 0x65,      // 'e'
    BL_ORIENTATION_WEST        = 0x77,      // 'w'
    BL_ORIENTATION_VERTICAL    = 0x76,      // 'v'
    BL_ORIENTATION_ANY         = 0x2A       // '*'
} BLorientation;
```

```
public enum BLorientation [C# Code]
{
    BL_ORIENTATION_NORTH      = 0x6E,
    BL_ORIENTATION_SOUTH      = 0x73,
    BL_ORIENTATION_EAST        = 0x65,
    BL_ORIENTATION_WEST        = 0x77,
    BL_ORIENTATION_VERTICAL    = 0x76,
    BL_ORIENTATION_ANY         = 0x2A
}
```

- `BL_ORIENTATION_ANY`: Any available orientation
- `BL_ORIENTATION_EAST`: east facing imaginary orientation
- `BL_ORIENTATION_NORTH`: north facing imaginary orientation
- `BL_ORIENTATION_SOUTH`: south facing imaginary orientation
- `BL_ORIENTATION_VERTICAL`: ortho imaginary
- `BL_ORIENTATION_WEST`: west facing imaginary orientation

### 4.2.7.2 Error codes

No Blom SDK method or function throws nor catches any exception. Also, user classes inheriting Blom SDK classes must be exception-safe. Errors are returned as `BLerror` error codes. A 0 (null) value should be considered success. Here follows the main error codes definitions:

```
typedef enum [C++Code]
{
    BL_SUCCESS = 0,
    BL_ERROR_UNSPECIFIED = -1,
    BL_ERROR_CRITICAL_ERROR = -2,
    BL_ERROR_NOT_FOUND = -3,
    BL_ERROR_BAD_INPUT = -4,
    BL_ERROR_DATA_SOURCE_NOT_FOUND = -5,
    BL_ERROR_BAD_LOGIN = -6,
    BL_ERROR_NOT_IMPLEMENTED = -7,
    BL_ERROR_NOT_INITIALIZED = -8,
    BL_ERROR_DATA_SOURCE_WAS_FOUND_CORRUPTED = -9,
    BL_ERROR_DATA_SOURCE_NOT_LICENSED = -10,
    BL_ERROR_BLOM_FEATURE_NOT_LICENSED = -11
} BLError;
```

```
public enum BLError [C# Code]
{
    BL_SUCCESS = 0,
    BL_ERROR_UNSPECIFIED = -1,
    BL_ERROR_CRITICAL_ERROR = -2,
    BL_ERROR_NOT_FOUND = -3,
    BL_ERROR_BAD_INPUT = -4,
    BL_ERROR_DATA_SOURCE_NOT_FOUND = -5,
    BL_ERROR_BAD_LOGIN = -6,
    BL_ERROR_NOT_IMPLEMENTED = -7,
    BL_ERROR_NOT_INITIALIZED = -8,
    BL_ERROR_DATA_SOURCE_WAS_FOUND_CORRUPTED = -9,
    BL_ERROR_DATA_SOURCE_NOT_LICENSED = -10,
    BL_ERROR_BLOM_FEATURE_NOT_LICENSED = -11
}
```

- **BL\_SUCCESS** (= 0): Operation was executed successfully.
- **BL\_ERROR\_UNSPECIFIED**: SDK found a generic error.
- **BL\_ERROR\_CRITICAL\_ERROR**: SDK found a critical exception that could not deal with.
- **BL\_ERROR\_NOT\_FOUND**: Requested data was unavailable or not found.
- **BL\_ERROR\_BAD\_INPUT**: User introduced bad arguments in current function call or returned a bad value in a feedback method call.
- **BL\_ERROR\_DATA\_SOURCE\_NOT\_FOUND**: Local file could not be found or opened or network connection could not be established.
- **BL\_ERROR\_BAD\_LOGIN**: Bad user or password.

`BL_ERROR_NOT_IMPLEMENTED`: A requested feature is not implemented in current version.

- `BL_ERROR_DATA_SOURCE_WAS_FOUND_CORRUPTED`: Data source was found corrupted.
- `BL_ERROR_DATA_SOURCE_NOT_LICENSED`: Data source was not licensed.
- `BL_ERROR_BLOM_FEATURE_NOT_LICENSED`: Feature is not licensed.

### 4.2.8 General procedure of data retrieving

For performance reasons, client application will be in charge of most of the memory allocation and objects creation according to the Blom SDK required interfaces. The general procedure to retrieve information from BlomURBEX is depicted below.

For instance, if a user demands an ortho-image for a certain location in the world:

Then, the following steps will be executed on runtime:

1. Client application creates an instance of class derived from `BImage` image or `Bitmap` image.
2. Client application requests the Blom SDK to download on this class the desired BlomURBEX ortho-image, using the function `LoadOrthoOverlay()`. This function is blocking.
3. When the download is successful, instance 'image' will be loaded with the requested image.
4. Once `LoadOrthoOverlay()` returns, client application can consume and deallocate the object as desired.

This schema is similar for all the other 3D objects served by BlomURBEX.

**Note:** In.NET interface, some methods have been made simpler, and sometimes they returns directly the data with high level objects.

For example this function returns a `system.drawing.Bitmap` object instead of requiring the user subclass any lower level object:

```
Public BError BDataLoader.LoadOrthoBaseLayer(out
system.drawing.Bitmap image, string qtreeKey, string baselayer)
```

**Important:** All functions are thread-safe, so several invocations can be done to any function from different threads concurrently. Even more, Blom SDK has been specifically designed to work with several threads, helping client application get maximum performance.

All loading functions are executed in the thread which they are called from, therefore they are blocking, and they will not return until it either completes its task or results in an error.

## 5 Blom SDK global functions

These functions can be organized in the following various groups:

### 5.1 Initialization and destruction functions

- `Initialize()` – This function initialize the Blom SDK environment and configure the disk and memory cache option. Below are the parameters of this function –

- `int maxMemCacheSizeInKB` – Maximum memory cache size in KB.
- `int maxDiskCacheSizeInKB` – Maximum disk cache size in KB. It will be automatically limited to 10 GB.
- `string diskCacheDirectory` – Local folder to store cache files.

e.g. below code initialize a Blom SDK by passing the memory cache, disk cache size and cache directory location.

```
int memCacheSizeInKB = 10000;
int diskCacheSizeInKB = 10000;
string cacheDirectory = "C:\\Temp\\BSDKCache";

BSDK.Initialize(diskCacheSizeInKB, memCacheSizeInKB, cacheDirectory);
```

[C# Code]

**Note:** C++ version of BSDK, requires a `BAbstractExecutor` object as parameter in `Initialize()`. Refer to 5.1.3 Classes Overview for more on `BAbstractExecutor`.

```
// Dummy implementation of BAbstractExecutor to demonstrate usage. [C++ Code]
class Executor : public BSDK::BAbstractExecutor
{
public:
    void Execute(BSDK::BTask &task)
    {
        task.Run();
    }

    void CancelAllTasks()
    {
    }
};

Executor executor;
BSDK::Initialize(200000, 2500000, "C:", executor, true);
```

**Note:** BSDK `Initialize()` function must be called before calling any other BSDK function.

- `Release()` Must be called at the end of execution, this method cleans up all the resources required in BSDK, such as internet connections, opened files and allocated memory. Afterwards, BSDK may be restarted using `Initialize()` function.

## 5.2 Meta data functions

- `GetCurrentVersion()` – This function will return the current version of Blom SDK.

```
// This will retrieve the BSDK version in string [C# Code]

string version = BSDK.GetCurrentVersion();
```

```
// This will get BSDK version [C++ Code]

const char* currentVersion = BSDK::GetCurrentVersion();
```

- `GetMachineID()` – This function will return the machine ID of Client terminal which is some combination of system hardware specific identifiers.

```
// This will get the machine ID [C# Code]

string machineID = BSDK.GetMachineID();
```

```
// This will get the machine ID [C++ Code]

const char* machineID = BSDK::GetMachineID();
```

- `GetDomainName()` – This function will return the string containing current domain name of machine.

```
// This will get the domain name [C# Code]

string domainName = BSDK.GetDomainName();
```

```
// This will get the domain name [C++ Code]

const char* domainName = BSDK::GetDomainName();
```

## 5.3 Data loader

- `CreateNewDataLoader()` – this function will create and return a new instance of `BDataLoader` class. `BDataLoader` will be used to login and retrieving data from BlomURBEX server and local Blom libraries. Refer 9 for more details.

```
//This will create the instance of BDataLoader [C# Code]

BDataLoader dataLoader = BSDK.CreateNewDataLoader();
```

```
// This will create the BDataLoader* [C++ Code]

BDataLoader* pDataLoader = BSDK::CreateNewDataLoader();
```

## 5.4 Blom library licensing

Data Loader can use BlomURBEX server or a local Blom Library as a data source. Before using a local Blom Library, it must be validated against a license. Blom SDK maintains a set of valid licenses which is used to when data loader adds a local Blom Library as a data source. For BLOM SDK a license is a one line string which contains information about the library/s licensed, validity period, type of license (License can be Machine Locked or Domain specific). A Blom License files (.bll) is a file which can contain one or multiple License strings. *Blom* SDK provides the following interface to manage the licenses:

- `AddLicenseFile()` – This method is used to add a license file to the list of *Blom* Library licenses kept by Blom SDK. The method accepts two arguments – a string containing the path to the license, and an instance of type `BAttributesReceiver(C#)/BAttributesReader(C++)`. `BAttributesReceiver(C#)/BAttributesReader(C++)` : is an abstract class, and it allows to receive all the attributes as Blom SDK performs certain operations. In this case, the operation is the reading of the license file which Blom SDK parses internally.

```
AttributesReceiverStdoutPrinter attributes = new AttributesReceiverStdoutPrinter [C# Code]

// initialize attributereciever with an instance of BAttributesReceiver

string pathToLicense = "D:\\BlomLicenses\\ESAVLA.bll";

BSDK.AddLicenseFile(attributes, pathToLicense);
```



```
// Output of above example will be
Attribute expiration-date Value 2001-01-01
Attribute library-name Value ESAVLA
Attribute library-type Value A
```

```
BSDK::BLError errorCode = BSDK::BL_SUCCESS; [C++ Code]

AttributeReaderStdoutPrinter objAttributeReader;
// AttributeReaderStdoutPrinter inherited from BAttributesReader

errorCode = BSDK::AddLicenseFile("D:\\BlomLicenses\\ESAVLA.bll",
objAttributeReader);
```

```
// Output of above example will be
Attribute expiration-date Value 2001-01-01
Attribute library-name Value ESAVLA
Attribute library-type Value A
```

**Note:** AttributesReaderStdoutPrinter and AttributesReceiverStdoutPrinter classes are defined later in the document. See section 4.3 for more details

- [RemoveLicenseFile\(\)](#) – Use this method to remove a license from the list of BLOM Library licenses maintained by BSDK. The method accepts a string containing the name to the license file.

```
BSDK.RemoveLicenseFile(@"ESAVLA.bll"); [C# Code]
```

```
BSDK::BLError errorCode = BSDK::BL_SUCCESS; [C++ Code]

errorCode = BSDK::RemoveLicenseFile("D:\\BlomLicenses\\ESAVLA.bll ");
```

- [ClearAllLicenses\(\)](#) – The call to the method simply clears the list of licenses being kept by Blom SDK.

```
BSDK.ClearAllLicenses(); [C# Code]
```

```
BSDK::BLError errorCode = BSDK::BL_SUCCESS; [C++ Code]

errorCode = BSDK::ClearAllLicenses();
```

- [GetLicense\(\)](#) – Use this method to retrieve a license string stored in BSDK by providing an `int` index. If the index is out of bounds, it will return NULL.

```

BSDK::BLError errorCode = BSDK::BL_SUCCESS;

//AttributesReader objAttributeReader;
AttributeReaderStdoutPrinter objAttributeReader;

errorCode = BSDK::AddLicenseFile("D:\\BlomLicenses\\ESAVLA.bll ",
objAttributeReader);

int i = 0;

while(const char* licenseString = BSDK::GetLicense(i))
{
    printf("License String at index %d -- %s\\n",i,licenseString);
    i++;
}
  
```

```

BSDK.AddLicenseFile(@"D:\\BlomLicenses\\ESAVLA.bll ", attributesReaderAssertor));

int i= 0;
string licenseString;
licenseString = BSDK.GetLicense(i);
while (licenseString != null)
{
    System.Console.WriteLine("License String {0}", licenseString);
    i++;
    licenseString = BSDK.GetLicense(i);
}
  
```

## 5.5 *GetLastDataSourceUsed*

BSDK provides API for getting name of the instance of datasource which was last accessed by DataLoader in current thread.

- `GetLastDataSourceUsed()` : This function will return the fully qualified name of the datasource last used in current thread or NULL if no datasource is used upto now.

Also in certain API's of BDataLoader, User can specify the name of the concrete datasource. These API's will then consider only the specified datasource. Following is the list of API's which support this behaviour.

- `GetDefaultLatitude()`
- `GetDefaultLongitude()`
- `GetDataSource()`
- `GetAvailableData ()`
- `GetDataInformation ()`

## 6 Implementing BAttributesReceiver

The abstract class `BAttributesReceiver(C#)` and `BAttributesReader(C++)` allows Blom SDK to communicate attributes being produced when certain operations are performed. The class provides an overloaded method `SetAttribute()`. Each overload has two parameters - attribute name and attribute values. Attribute name is a string and values are passed as an array of different data types and thus the overloads:

The class implementing the abstract class must know all the attributes Blom SDK will be providing values for and their types. It must then implement the required overloads and provide empty implementation for those not required.

```
class AttributesReceiverStdoutPrinter : BAttributesReceiver    [C# Code]
{
    public AttributesReceiverStdoutPrinter()
    {
    }
    override public void SetAttribute(string n, byte[] v)
    {
        foreach (byte b in v)
        {
            System.Console.WriteLine("Attribute{0} Value{1}", n, b);
        }
    }

    override public void SetAttribute(string n, bool[] v)
    {
        foreach (bool b in v)
        {
            System.Console.WriteLine("Attribute{0} Value{1}", n, b);
        }
    }

    override public void SetAttribute(string n, int[] v)
    {
        foreach (int i in v)
        {
            System.Console.WriteLine("Attribute{0} Value{1}", n, i);
        }
    }

    override public void SetAttribute(string n, double[] v)
    {
        foreach (double d in v)
        {
            System.Console.WriteLine("Attribute{0} Value{1}", n, d);
        }
    }

    override public void SetAttribute(string n, string[] v)
    {
        foreach (string s in v)
        {
            System.Console.WriteLine("Attribute{0} Value{1}", n, s);
        }
    }
}
```

Here is a simple sample implementation of class derived from BAttributesReader (for C++ users), which prints attributes on screen.

```
class AttributeReaderStdoutPrinter : public BSDK::BAttributesReader[C++ Code]
{
public:
    AttributeReaderStdoutPrinter()
    {
    }
    void SetAttribute(const BSDK::BLprintingchar *varname,
        BSDK::BLvarianttype datatype, BSDK::BLint32 attrCount, const
        BSDK::BLvariant *valuesarray)
    {
        switch (datatype)
        {
        case BSDK::BL_BOOL_TYPE:
        {
            for (int i = 0; i < attrCount; i++)
            {
                printf("Attribute %s Value %i\n",varname,valuesarray[i]);
            }
            break;
        }

        case BSDK::BL_UINT8_TYPE:
        {
            for (int i = 0; i < attrCount; i++)
            {
                printf("Attribute %s Value %i\n",varname,valuesarray[i]);
            }
            break;
        }

        case BSDK::BL_INT32_TYPE:
        {
            for (int i = 0; i < attrCount; i++)
            {
                printf("Attribute % Value %i\n",varname,valuesarray[i]);
            }
            break;
        }

        case BSDK::BL_FIXED32_TYPE:
        {
            for (int i = 0; i < attrCount; i++)
            {
                printf("Attribute %s of BOOL type has value
%i\n",varname,valuesarray[i]);
            }
            break;
        }
        }
    }
}
```

```

case BSDK::BL_FLOAT64_TYPE:
{
    for (int i = 0; i < attrCount; i++)
    {
        printf("Attribute %s Value %f\n",varname,valuesarray[i]);
    }
    break;
}

case BSDK::BL_STRING_TYPE:
{
    for (int i = 0; i < attrCount; i++)
    {
        printf("Attribute                %s                Value
%s\n",varname,valuesarray[i].strVal);
    }
    break;
}

case BSDK::BL_UNICODE_STRING_TYPE:
{
    for (int i = 0; i < attrCount; i++)
    {
        printf("Attribute                %s                Value
%ws\n",varname,valuesarray[i].strVal);
    }
    break;
}
}
}
};

```

## 7 *Blom SDK event viewer*

Blom SDK may be configured to notify about internal events using [BEventViewer](#). These notifications include important events, such as communication problems. In order to trace the problem easily, these events will be given using detailed codes for machine reading and messages written in natural English language for human understanding.

Client application should use [BEventViewer](#) to redirect the SDK events wherever it is necessary: Notifying to user, writing to output log files or databases, etc.

For users of BSDK .NET version, [BEventViewer](#) class should be instanced by client application and delivered to client using [BEventViewer.Notify](#) delegate.

```
// attach function [C# Code]
BEventViewer.NotifyEvent = NotifyEvent;

// Event handler
void NotifyEvent(BEventViewer.BLeventtype eventType,
BEventViewer.BLeventcode eventCode, string message)
{
    // Handle and log event
}
```

Before building a message to notify the client about an event, [BEventViewer](#) client object may be asked whether it wants to receive such event using [EventTypeEnabledFilter\(\)](#) delegate.

e.g. below example shows how can we configure it to notify only event of type `BL_EVENT_TYPE_IO_EXCEPTION` –

```
// attaching function [C# Code]
BEventViewer.EventTypeEnabledFilter = EventTypeEnabledFilter;

bool EventTypeEnabledFilter(BEventViewer.BLeventtype eventType)
{
    if (eventType == BEventViewer.BLeventtype.BL_EVENT_TYPE_IO_EXCEPTION)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

If user wants to receive current event, message will be built and [NotifyEvent\(\)](#) method will be called.

Users of BSDK C++ version need to subclass BEventViewer.

Below example shows how we can configure to notify only for event type BL\_EVENT\_TYPE\_IO\_EXCEPTION

```
class EventViewer : public BSDK::BEventViewer [C++ Code]
{
    int numMessages;

public:
    BSDK::BLbool IsEventTypeEnabled(BSDK::BEventViewer::BLeventtype eventType)
    const
    {
        if( eventType == BSDK::BEventViewer.BL_EVENT_TYPE_IO_EXCEPTION)
        {
            return true;
        }
        return false;
    }

    void NotifyEvent(BSDK::BEventViewer::BLeventtype eventType,
        BSDK::BEventViewer::BLeventcode eventCode,
        const BSDK::BLprintingchar *message)
    {
        numMessages ++;
    }
};

// now set the EventViewer in BSDK
EventViewer      eventViewer;
BSDK::SetEventViewer(&eventViewer);
```

## 8 *Blom SDK Proxy Settings*

Blom SDK allows you to specify your own implementation of how to handle proxy settings. Blom SDK provides an abstract class `BProxySettings`. In order to handle proxy settings, you need to create a class by inheriting from `BProxySettings` and pass its instance to `BSDK.SetProxySettings()` function. Blom SDK uses `GetProxyProxy()` property, if its value is null then that means it is disabled and enabled otherwise. If the proxy is enabled then it will query `BProxySettings` implementation username and password, where you can handle these as you want. E.g. you want to present a dialog for the user to enter proxy server username and password. Below is a minimal implementation of `BProxySettings` class –

```
public class MyProxySettings : BProxySettings [C# Code]
{
    // Query for the proxy server address, if you return a valid
    // address then it will query for the
    public override string GetProxyServer()
    {
        // Write your business logic here to control whether
        // proxy is enabled or not

        bool isProxyEnabled = true;

        if (isProxyEnabled)
        {
            // Write your business logic here to return a correct IP.
            return "192.168.1.10";
        }
        else
        {
            // If proxy is not enabled the return null
            return null;
        }
    }

    // If you provide a proxy server address in GetProxyServer(), then
    // this function will be called for retrieving username and password
    // for proxy login
    public override string GetProxyAuthorization(
        BLAuthenticationScheme scheme, string realm)
    {
        // Write your business logic here to return correct credentials.
        return "proxyUsername:proxyPassword";
    }
}
```



```
class MyProxySettings : public BSDK::BProxySettings [C++ Code]
{
    char proxyServer[260];

public:
    const char *GetProxyServer()
    {
        // Write your business logic here to control whether
        // proxy is enabled or not
        bool bIsProxyEnabled = true;
        if (bIsProxyEnabled )
        {
            return "192.168.1.10";
        }
        else
        {
            return 0;
        }

    }
    // If you provide a proxy server address in GetProxyServer(), then this
    // this function will be called for retrieving username and password
    // for proxy login.
    const char
*GetProxyAuthorization(BSDK::BProxySettings::BLAuthenticationScheme scheme,
const char *realm)
    {
        // write your business logic here to return correct credentials
        return "proxyusername:proxypassword";
    }

};
```

## 9 Data loader – Initializing and managing data sources

Data loader is responsible for retrieval of data from BlomURBEX server and Blom library. Before you start using its function you must specify a data source from where it will load the data. Data loader supports two types of data sources – BlomURBEX and Blom library.

### 9.1 Creating a data loader

You can create a data loader using `BSDKW.CreateNewDataLoader()` function.

```
BDataLoader DataLoader = BSDKW.CreateNewDataLoader();
```

[\[C# Code\]](#)

```
BDataLoader* pDataLoader = BSDK::CreateNewDataLoader();
```

[\[C++ Code\]](#)

```
// And when you are done:
delete pDataLoader;
```

### 9.2 Data sources

DataSources are used for defining a source providing GIS data. For eg. If Client application wishes to retrieve data online from BLOM URBEX server, then it needs to create a `BLOMURBEXDataSource`, else if Client application wants to retrieve data from a local BLOM library, then a `BLOMLibraryDataSource` should be created. Each DataSource is defined by a string. While adding datasource, Client can specify the priority for it. When calling DataLoader's various APIs, data is fetched from datasource based on their priority. Eg.

```
//Create a new dataloader
```

[\[C# Code\]](#)

```
BDataLoader dataLoader = BSDKW.CreateNewDataLoader();
```

```
// now add online BLOMURbexDataSource
dataLoader.AddDataSource(0, "BlomURBEX3DServer:username:password");
```

```
// now add a local library datasource
dataLoader.AddDataSource(0, "blomlib:D:\\ESAVLA");
```

```
// GetDefaultLongitude for BLOMURbexDataSource
```

```
Double longitude;
```

```
longitude=dataLoader.GetDefaultLongitude("BlomURBEX3DServer:username:password");
;
```

```
//Create a new dataloader [C++ Code]

BDataLoader* pDataLoader = BSDK::CreateNewDataLoader();

// now add online BLOMURbexDataSource
pDataLoader->AddDataSource(0, "BlomURBEX3DServer:username:password");

// now add a local library datasource
pDataLoader->AddDataSource(0, "blomlib:D:\\ESAVLA");

// GetDefaultLongitude for BLOMURbexDataSource
pDataLoader->GetDefaultLongitude("BlomURBEX3DServer:username:password");

// further use of DataLoader
//
.
.
.
// at the end, delete the BDataLoader object
delete pDataLoader;
```

As each datasource is defined by its string while adding it in DataLoader, Client can specify name (same string as used while calling AddDataSource) of a particular datasource in some API's of DataLoader. Then in those API's datasource with the specified name will only be considered. Client can get the string for datasource, which is accessed/used last time in current thread by calling BDataLoader's API: GetLastDataSourceUsed().

**Note:** In C++ you have to delete the object when you are done with the dataloader object.

```
BDataLoader* pDataLoader = BSDK::CreateNewDataLoader(); [C++ Code]

// And when you are done:
delete pDataLoader;
```

### 9.3 Adding BlomURBEX server as data source

To add BlomURBEX server as data sources you need either a username/password OR a user token to access the data.

If you have username password then this is how you add it as data source to the map:

```
BDataLoader dataLoader = BSDK.CreateNewDataLoader(); [C# Code]
```

```
dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");
```

```
BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader(); \[C++ Code\]
```

```
dataLoader1->AddDataSource(1, "blomURBEX3DServer:username:password");
```

If you have a user token then this is how you add the it as data source to data loader –

```
BDataLoader dataLoader = BSDK.CreateNewDataLoader(); \[C# Code\]
```

```
dataLoader.AddDataSource(1, "blomurbex3dserver:usertoken");
```

```
BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader(); \[C++ Code\]
```

```
dataLoader1->AddDataSource(1, "blomURBEX3DServer:usertoken");
```

## 9.4 Adding Blom library as data source

You can also add any local blom library as data source to data loader.

If you have a blom library on your file system then this is how to add it as a data source to your application:

```
BDataLoader dataLoader = BSDK.CreateNewDataLoader(); \[C# Code\]  
string path = "D:\\BlomLiraries\\ESAVLA";  
dataLoader.AddDataSource(0, string.Format("Blomlib:{0}", path));
```

```
BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader(); \[C++ Code\]
```

```
dataLoader1->AddDataSource(0, "blomlib:D:\\ESAVLA06-093-LIB");
```

Where path is the location of the root folder of the library. You can also add multiple blom libraries using same technique. Just keep on adding data source to the BDataLoader class. However, to use any blom library, you need a valid license in order to use it. For more details on how to use licenses, Refer 5.4 for more details.

## 9.5 Get default latitude and longitude

After adding datasource, you can request the default latitude and longitude for that user or blom library. Use [DefaultLatitude](#) and [DefaultLongitude](#) properties ro retrieve this information.

```
BDataLoader dataLoader = BSDK.CreateNewDataLoader(); \[C# Code\]  
dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");
```

```
double defaultLatitude = dataLoader.DefaultLatitude;  
double defaultLongitude = dataLoader.DefaultLongitude;
```

```
BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader(); \[C++ Code\]  
  
dataLoader1->AddDataSource(1, "BlomURBEX3DServer:usrname:password");  
  
double defaultLongitude = dataLoader1->GetDefaultLongitude();  
double defaultLatitude = dataLoader1->GetDefaultLatitude();
```

## 10 Ortho imagery loading

Ortho images loading follows the general procedure detailed in 4.2.8.

### 10.1 Load ortho base and overlay

You can use `BDataLoader` to retrieving ortho images. To retrieve an ortho image, client application must invoke `BDataLoader::LoadOrthoOverlay()` or `LoadOrthoBaseLayer()`, these methods require the following parameters:

- `out Bitmap image/ BImage image` : The client application object where the data will be loaded. In C++ version of BSDK, Client needs to pass an object derived from `BSDK::BImage` class, whereas in C# Client can pass a high level object of type `system.Drawing.Bitmap`.
- `string qtreeKey`: Ortho image location requested (as defined in 4.2.1).
- `string overlay`: Overlay / baselayer name requested, null in case of you are requesting base map.
- `BLorientation orientation`: Imagery orientation: North, South, East, West or Ortho.
- `string crs`: Coordinate reference system. By default: "epsg:3785", Spherical Mercator
- `string date`: Years range. Depending on the format, the date can be returned in the following years range:
  - A null value (0): Data returned will be the most recent available. This is the value by default.
  - `<year>`: Data returned will be the most recent available captured in the given year or older.
  - `<year1>-<year2>`: Data returned will be the most recent available captured between the given years.
  - `<year1>-<year1>`: Data returned will be the most recent available captured in the given year.

An image in BSDK is represented using the `BImage` class. The user needs to inherit `BImage` class and deliver its own object to loading method.

Finally, `LoadOrthoOverlay()` / `LoadOrthoBaseLayer()` will return.

Example –

```

BDataLoader dataLoader = BSDK.CreateNewDataLoader();
dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");

Bitmap image;
BLError result = dataLoader.LoadOrthoOverlay(out image, "120210001131000310",
    null, BLorientation.BL_ORIENTATION_VERTICAL, "epsg:3785");
if (result == BLError.BL_SUCCESS)
{
    // request image is successfully loaded in the "image" object
}
  
```

[C# Code]

```

BSDK::BDataLoader *pDataLoader = BSDK::CreateNewDataLoader();

BSDK::BLError errCode = BSDK::BL_SUCCESS;

pDataLoader->AddDataSource(0, "BlomURBEX3DServer:username:password");

MyImage image; // MyImage inherited from BImage

errCode=pDataLoader->LoadOrthoBaseLayer(image,"0331110012103211122","default",
BSDK::BL_ORIENTATION_NORTH, "epsg:3785", "2006");

if(errCode == BLError.BL_SUCCESS){

    // request image is successfully loaded in "image" object.}
  
```

[C++ Code]

**Note:** All orthophoto and road overlays images will be always 256 x 256 pixels each.

## 10.2 Load map portion

You can request a portion of map as a single image using **BDataLoader's** function **LoadOrthoArea()** function. It accepts following parameters –

- **out Bitmap image/ BImage image:** The client application object where the data will be loaded. In C++ version of BSDK, Client needs to pass an object derived from BSDK::BImage class, whereas in C# Client can pass a high level object of type system.Drawing.Bitmap.
- **double minX, minY, maxX, maxY:** The region requested
- **string crs:** Coordinate reference system. By default: "epsg:3785", Spherical Mercator

- `int` width, height: width and height of the output image of the map
- `BLorientation` orientation: Imagery orientation: North, South, East, West or Ortho.
- `string` baselayer: layer name requested, null in case of you are request base map
- `string` style: This parameter is reserved for future use.
- `string` date: Years range. Depending on the format, the date can be returned in the following years range:
  - A null value (0): Data returned will be the most recent available. This is the value by default.
  - `<year>`: Data returned will be the most recent available captured in the given year or older.
  - `<year1>-<year2>`: Data returned will be the most recent available captured between the given years.
  - `<year1>-<year1>`: Data returned will be the most recent available captured in the given year.

[\[C# Code\]](#)

```
//e.g. following code load a portion of map into the Bitmap object
imageBDataLoader dataLoader = BSDK.CreateNewDataLoader();
dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");

Bitmap image;
BLError result = dataLoader.LoadOrthoArea(out image,
    -523000, 4960000, -521500, 4961500, "EPSG:3785", 500, 500,
    BLorientation.BL_ORIENTATION_VERTICAL, "ortho");

if (result == BLError.BL_SUCCESS)
{
    // Bitmap image will be load with the request map portion
}
```



```
BSDK::BDataLoader *pDataLoader = BSDK::CreateNewDataLoader(); \[C++ Code\]

BSDK::BLError errCode = BSDK::BL_SUCCESS;

pDataLoader->AddDataSource(0, "BlomURBEX3DServer:username:password");

MyImage image; // MyImage inherited from BImage

errCode=pDataLoader->LoadOrthoArea(image,-523000, 4960000, -521500, 4961500,
"EPSG:3785", 550, 550, BLorientation.BL_ORIENTATION_VERTICAL,"ortho")

);

if(errCode == BLError.BL_SUCCESS){

// request image is successfully loaded in "image" object.}
```

## 11 Digital terrain model loading

**NOTE:** This functionality is not available in .NET version of BSDK. It will be available in future versions.

DTM loading follows the general procedure detailed in 4.2.8.

The client application must invoke `LoadDTM()` to retrieve a DTM grid section. This method contains the following parameters:

- `BDTMGrid dtmGrid`: The client application object where the data will be loaded. You create your own implementation of `BDTMGrid` class by inheriting from it.
- `string qtreeKey`: The DTM grid location requested (as defined in 4.2.1).
- `string layer`: The DTM layer requested.
- `string crs`: Coordinate reference system. By default: "epsg:3785", Spherical Mercator
- `string date`: Years range. Depending on the format, the date can be returned in the following years range:
  - A null value (0): Data returned will be the most recent available. This is the value by default.
  - `<year>`: Data returned will be the most recent available captured in the given year or older.
  - `<year1>-<year2>`: Data returned will be the most recent available captured between the given years.
  - `<year1>-<year1>`: Data returned will be the most recent available captured in the given year.

DTM grid objects are very small, so it is not efficient to request every single `qtreeKey`, from level 1 to 22. To reduce such a lot of grid invocations. DTM will be served only at certain levels. First level serving DTM will be at level 1 (i.e. quadtree keys '0', '1', '2' and '3'). The following valid level will be indicated as a response to this request in the `dtmCovering` parameter from `BDTMGrid.Initialize()` method.

**Example:**

A request with quadtree key "0" will return 1 as `dtmCovering`, and a grid with 6x6 heights. This indicates that for a more detailed DTM you should call with a quadtree key of length 2 with "0" as a prefix.

A request with quadtree key "03" will return 5 as `dtmCovering`, and a grid with 96x96 heights. This grid should be enough for the following 5 levels. This indicates that for a more detailed DTM you should call with a quadtree key of length 7 with "03" as a prefix.

A bad request with quadtree key "031" will return 0 as `dtmCovering`, and no grid at all. It is because previous level has indicated that this level should be skipped. It has "03" as a prefix but quadtree key contains only 3 characters!

A correct request may be with quadtree key "0313332", which will return 5 as `dtmCovering`, and a grid with 96x96 DTM values. This indicates that for a more detailed DTM you should call with a quadtree key of length 12 with "0313332" as a prefix.

A correct request with quadtree key "031333200000", may return 0 as `dtmCovering` and no grid. This result is given because, currently BlomURBEX does not have more DTM information in this area, but the request is absolutely correct and possibly BlomURBEX might respond correct DTM information in the future.

**D**

uring DTM loading procedure, BSDK will initialize `BDTMGrid` object calling `Initialize()` client application method with the following parameters:

- `int resolution`: DTM resolution. Indicates how many samples are given for this section. Client application will be given one additional sample at the right (sample longitude index = `resolution`) and at the top (sample latitude index = `resolution`), that will coincide with the adjacent DTM tile.
- `int dtmCovering`: DTM covering.

Afterwards, BSDK will call `GetWritableDTMRow()` and `ProcessDTMRow()` alternatively for each DTM latitude line (`resolution + 1` times) in order to let developer use and store independently each line of the DTM.

- `GetWritableDTMRow()` must return enough space to save a row: `(resolution + 1) int32`.

- BSDK will call `ProcessDTMRow()` to let client application postprocess each DTM latitude line. Always, after this call, BSDK will discard given memory.

Finally, `LoadDTM()` will return.

This DTM loading approach enables high performance for simple loading heights into a single array, and also, fast conversions on the fly such as down sampling, up sampling, projection conversions, etc.

**Note:** If a DTM section is not available at a certain level for a given area, it will not be available a higher resolution in the same area.

**Another note:** We are working on several more efficient DTM format, as QuadTIN data structures. Blom SDK may evolve in that direction in the future.

## 12 Urban 3D models loading

**NOTE:** This functionality is not available in .NET version of BSDK. It will be available in future versions.

### 12.1 Overview

Urban 3D models are complex objects comprising the following objects.

- 3D model: Represents a set of meshes located in a certain squared area. Models can only be requested at quadtree level 18. All the covered objects share the following attributes. A model is compound of:
  - A set of attributes. These attributes may contain information about model location, building names, shape type, landmarks identification, reference points to elevate the building on the client DTM, author, etc. Details will be depicted below in section 12.6.
  - A set of 3D coordinates and texture coordinates. 3D coordinates will contain (y, x) in Spherical Mercator projection and a height component in meters (depending on 3D layer height component may be given as height above ground level). Several vertexes in the same model may share any coordinate.
  - A set of meshes (as defined below).
- 3D mesh: represents a set of shapes of a 3D model sharing the same texture. A 3D mesh contains the following information:
  - A texture loader object, that lets the client application to load the corresponding texture in the future (if the shape is texturized).
  - A set of shapes (as defined below).
- 3D shape: represents a geometric shape, containing the following information:
  - A set of attributes. These properties may contain information about building identifier, shape identifier, shape type (roof, façade, pediment or socket), etc. Details will be depicted below in section 12.6.
  - A shape type: indicating that current shape is a triangles array.
  - A default RGBA color for this shape.
  - A set of vertex indexes referred to the array of model 3D coordinates and texture coordinates.

**Note:** It might be useful to make `BModel` implementation ready to aggregate several model layers in the same object. It will make easy in the future to load an Urban layer with high-resolution landmarks and afterwards, aggregate another Urban layer containing lower resolution buildings mixed in the same tile.

To complete this feature, `BModel` implementation should refuse inserting a new low-level building when an already inserted high-level building (i.e. a landmark) with the same building identifier had been added.

The tile configuration procedure will follow this sequence:

## 12.2 Model loading

Model tile loading follows the general procedure detailed in 4.2.8.

To retrieve a model tile, client application must invoke `BDataLoader.LoadModel()`. This method contains the following parameters:

- `BModel model`: The client application object where the data will be loaded. You create your own implementation of `BModel` class by inheriting from it.
- `string qtreeKey`: Model tile location requested (as defined in 4.2.1).
- `string layer`: Building model layer requested.
- `string crs`: Coordinate reference system. By default: "epsg:3785", Spherical Mercator
- `string date`: Years range. Depending on the format, the date can be returned in the following years range:
  - A null value (0): Data returned will be the most recent available. This is the value by default.
  - `<year>`: Data returned will be the most recent available captured in the given year or older.
  - `<year1>-<year2>`: Data returned will be the most recent available captured between the given years.
  - `<year1>-<year1>`: Data returned will be the most recent available captured in the given year.

**Note:** Current configuration of BlomURBEX stores model tiles only at level 18, so only quadtree keys with 18 characters will return 3D models.

```
BDataloader dataLoader = BSDK.CreateNewDataLoader();

dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");

MyModel model = new MyModel();

BLError result = dataLoader.LoadModel(model, "120210001131000210", null,
    "epsg:3785", null);

if (result == BLError.BL_SUCCESS)
{
    // requested model is loaded in the in MyModel instance
}
```

The tile configuration procedure will follow this sequence:

- Blom SDK calls `Initialize()` indicating the number of meshes and the number of vertexes coordinates. Also, it will be provided an offset in meters to be applied to all the 3D coordinate vertexes.
- Blom SDK calls `SetAttribute()` method as necessary setting all attributes. For more details about model attributes, see 12.6.
- Blom SDK calls `GetWritableVertexArray()`, which must return an array with enough memory for `3 * vertexCount int32`. This array will be filled up with `vertexCount` segments of three numbers, each segment with the following information.
  - `y`: Spherical Mercator Y coordinate.
  - `x`: Spherical Mercator X coordinate.
  - `z`: Height above ground level in meters.

To obtain the real value, these values should be converted to double value and then added to coordinate offset given in `Initialize()`.

- Blom SDK calls `ProcessVertexArray()`, to let client postprocess vertex array.
- Blom SDK calls `GetWritableTexCoordsArray()`, which must return an array with enough memory for `2 * vertexCount int32`. This array will be

- filled up with `vertexCount` segments of two numbers, each segment with the following information:
  - `x`: X texture coordinate.
  - `y`: Y texture coordinate.

To obtain the real value, these values should be converted to double value.

- Blom SDK calls `ProcessTexCoordsArray()`, to let client postprocess texture coordinates array.
- Afterwards, Blom SDK will call `GetWritableMesh()` and `ProcessMesh()` alternatively for each mesh in order to let developer use and store independently each one.
  - `GetWritableMesh()` invocation must return a `BMesh` object ready to be configured.
  - Given mesh will be configured as depicted in 12.3.
  - `ProcessMesh()` invocation will notify client application that given `BMesh` object is already written.
- Model tile will be considered configured when every mesh object had been configured (see below in 12.3).

Finally, `LoadModel()` will return.

## 12.3 Meshes loading

During model configuration, the meshes will be also configured.

The procedure to set up a `BMesh` will be the following:

- Blom SDK calls `Initialize()` indicating the number of shapes and will set a `BTextureLoader` if the mesh is texturized (`BTextureLoader` will be described afterwards in 12.5). `BTextureLoader` object will be automatically destroyed when `Initialize()` method ends, so client application should create a copy of this object to keep alive a reference to this object, using `clone()` method. Cloned object should be deleted by the client using `delete` operator.
- After, Blom SDK will call `GetWritableShape()` and `ProcessShape()` alternatively for each shape in order to let developer use and store independently each one.



- `GetWritableShape()` invocation must return a `BShape` object ready to be configured.
- Given shape will be configured as depicted in 12.4.
- `ProcessShape()` invocation will notify client application that given `BShape` object is already written.
- Mesh will be considered configured when every shape object had been configured (see below in 12.4).

## 12.4 Shapes loading

During mesh configuration, the shapes will be also configured.

The procedure to set up a `BShape` will be the following:

- Blom SDK calls `Initialize()` with the following parameters.
  - `shapeType`: Integer indicating that the shape is being defined is a triangles array.
  - `vertexCount`: Integer indicating how many vertex does this shape contain.
  - `colorComponent[4]`: Four bytes indicating the default RGBA color.
- Blom SDK calls `SetAttribute()` method as necessary setting all attributes.
- Blom SDK calls `GetWritableVertexesIndexes()` once. The client application must return an array with enough memory for `vertexCount` `int32`, these values will be referred to `BModel` coordinates vertex and texture coordinates given in `BModel.GetWritableVertexArray()` and `BModel.GetWritableTexCoordsArray()`.

## 12.5 Building texture loading

`BTextureLoader` class enables the loading of building textures on demand anytime. A `BTextureLoader` object is provided to a `BMesh` objects during its initialization.

`BTextureLoader` has the following methods:

- `Clone()`: This method lets user create a copy of this object, copy construction and assignment operator is explicitly disallowed.
- `HashCode()`: This method may be used by the user to compare two `BTextureLoader`s. If the result given in the output parameter of two `BTextureLoader`s is the same, they will download the same file. It should

be used by the client to download only once each different image. It is especially useful in pattern textures based models such as LOD3.

- `Multiresolution()`: This method returns whether this texture loader supports multiresolution. If this method returns true, different images with adequate quality will be given in `LoadTexture()` for each quality value requested. Otherwise, the same image will be returned despite different qualities are requested.

The client application must invoke `LoadTexture()` in order to load a building texture. This method contains the following parameters:

- `out Bitmap image`: The client application object where the data will be loaded.
- `int quality`: Requested quality; it should be in the range [0 (lowest quality) – 3 (highest quality)]. If the image has not multiresolution, this value will be ignored.

Building textures loading follows the same procedure defined in 10 for Earth surface images.

## 12.6 Models and shapes attributes

Models attributes lets associate meta-data to the model data. Currently the data provided as attributes are building and façade identification, models base points to elevate the different buildings on the digital terrain model correctly and a flag indicating whether this model shares the same texture with the any other model.

- Building codes: Models will be given an array of strings with the attribute tag `BModel.BLmodelattr.BL_MODEL_ATTR_BUILDING_BLOCK_ID_LIST`, this array values will be referenced by index in the shape attribute `BShape.BLshapeattr.BL_SHAPE_ATTR_BUILDING_BLOCK_ID_INDEX`.
- `BShape.BLbuildingpart.BL_SHAPE_ATTR_BUILDING_PART_TYPE` attribute will identify whether each shape is part of a façade, a roof, a pediment or a socket of the building it belongs to. The returned value will be one of the values in the `BShape.BLbuildingpart` enumeration.
- `BShape.BLshapeattr.BL_SHAPE_ATTR_BUILDING_PART_ID` will join the parts of the buildings that form a unity. For example, a building may have four facades, each one compound of two triangles. This number will indicate which pair of triangles forms part of the same façade.
- `BModel.BLmodelattr.BL_MODEL_ATTR_BUILDING_BASE_POINT_LIST` is an array of two `int32` numbers per building, containing pairs of latitude/longitude coordinates per building (in relative coordinates in Spherical Mercator as the rest of the 3D vertexes). Each building must be raised or lowered the same distance from the ground regardless the terrain digital model employed. The amount of elevation

that should be added or subtracted of each building is the value of the digital terrain model in the given latitude/longitude. Using the best quality of BlomURBEX 3D digital terrain model given through this BSDK, the buildings will fit perfectly in every terrain.

- `BModel.BLmodelattr.BL_MODEL_ATTR_SHARED_TEXTURES` will return a Boolean value giving a hint to the client application indicating whether the model tile is pattern based. When a model is pattern based several neighbor tiles will probably have the same textures, and therefore the textures can be loaded only once and then shared among them. It is particularly useful for LOD3 models, forest, traffic signs, etc. When this flag returns a true value, client application should check whether each two `BTextureLoaders` from different tiles reference the same textures to save up space and improve performance. It could be done using the `HashCode()` method.

## 13 Discrete imagery loading

To load discrete imagery, application has to find the discrete images that capture a certain point or region. Discrete listing follows the general procedure detailed in 4.2.8.

### 13.1 Find discrete images by extent

This function can be used to find list of discrete images in the provided earth region. Client application must invoke `FindDiscreteImagesByExtent()` to retrieve the discrete images in a certain Earth region. This method contains the following parameters:

- `List<BDiscreteImage> discreteList/BDiscreteImageList`: The client application object where the data will be loaded.

OR

`ReadDiscreteImageDelegate readDiscreteImageDelegate`: The delegate will notify all discrete images one by one to the registered function.

- `double minX, minY, maxX, maxY`: The region requested.
- `BLorientation orientation (optional)`: Imagery orientation: North, South, East, West or Ortho.
- `string crs (optional)`: Coordinate reference system. By default: "epsg:4326", WGS84 Geographics Coordinates.
- `string date (optional)`: Years range. Depending on the format, the date can be returned in the following years range:
  - A null value (0): Data returned will be the most recent available. This is the value by default.
  - `<year>`: Data returned will be the most recent available captured in the given year or older.
  - `<year1>-<year2>`: Data returned will be the most recent available captured between the given years.
  - `<year1>-<year1>`: Data returned will be the most recent available captured in the given year.

A discrete image in Blom SDK is represented using the class `BDiscreteImage`.

```

BDataLoader dataLoader = BSDK.CreateNewDataLoader();
dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");

List<BDiscreteImage> discreteImagesList = new List<BDiscreteImage>();
BLError result = dataLoader.FindDiscreteImagesByExtent(discreteImagesList,
    xmin, ymin, xmax, ymax, BLorientation.BL_ORIENTATION_NORTH, "EPSG:3785",
    null);
if (result == BLError.BL_SUCCESS)
{
    // discreteImagesList will contain all available discretized meta data
}
  
```

```

BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader();

dataLoader1->AddDataSource(1, "blomurbex3dserver:username:password");

MyDiscreteImageList  imageList;

// MyDiscreteImageList inherited from BDiscreteImageList

dataLoader1->FindDiscreteImagesByExtent(imageList,  xmin,  ymin,  ymax,  ymax,
    BSDK::BL_ORIENTATION_WEST);
  
```

Where xmin, ymin, xmax, ymax are the coordinates defining a certain region on earth.

## 13.2 Find nearest discrete images to a location

This function can be used to find list of discrete images which are nearest to a particular location. Client application must invoke `FindDiscreteImage()` to retrieve the available discrete in a certain Earth region. This method contains the following parameters:

- `List<BDiscreteImage>` discreteList/`BDiscreteImageList`: The client application object where the data will be loaded.

OR

`ReadDiscreteImageDelegate` readDiscreteImageDelegate: The delegate will notify all discrete images one by one to the registered function.

OR

`out BDiscreteImage` discreteImage: This parameter will contain the returned discrete image.

- `double` x, y: x and y coordinate of the location requested

- `BLorientation` orientation (optional): Imagery orientation: North, South, East, West or Ortho.
- `string` crs (optional): Coordinate reference system. By default: "epsg:4326", WGS84 Geographics Coordinates.
- `string` date (optional): Years range. Depending on the format, the date can be returned in the following years range:
  - A null value (0): Data returned will be the most recent available. This is the value by default.
  - `<year>`: Data returned will be the most recent available captured in the given year or older.
  - `<year1>-<year2>`: Data returned will be the most recent available captured between the given years.
  - `<year1>-<year1>`: Data returned will be the most recent available captured in the given year.

A discrete image in Blom SDK is represented using the class `BDiscreteImage`.

```
BDataLoader dataLoader = BSDK.CreateNewDataLoader();  
dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");  
  
double x = 1399999.7082705908D;  
double y = 7493990.1591216344D;  
  
List<BDiscreteImage> discreteImagesList = new List<BDiscreteImage>();  
BLError result = dataLoader.FindDiscreteImage(discreteImagesList, x, y,  
    BLorientation.BL_ORIENTATION_NORTH, "EPSG:3785", null);  
if (result == BLError.BL_SUCCESS)  
{  
    // discreteImagesList will contain all available discretess meta data  
}
```

```

BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader();

dataLoader1->AddDataSource(1, "blomurbex3dserver:username:password");

MyDiscreteImageList  imageList;

// MyDiscreteImageList inherited from BDiscreteImageList

double x = -4.6836912974;

double y = 40.6680158844;

if (BSDK::BL_SUCCESS==dataLoader1->FindDiscreteImage(imageList,x,y,
BSDK::BL_ORIENTATION_VERTICAL))

{

// imageList will contain all available discretized meta data

}
  
```

During oblique list loading procedure, BSDK will deliver to the `List<BDiscreteImage>` instance.

### 13.3 Find all discrete images at a particular location

This function can be used to find list of discrete images in the provided location. Client application must invoke `FindDiscreteImages()` to retrieve the available discrete in a certain Earth region. This method contains the following parameters:

- `List<BDiscreteImage>` discreteList/BDiscreteImageList: The client application object where the data will be loaded.

OR

`ReadDiscreteImageDelegate` readDiscreteImageDelegate: The delegate will notify all discrete images one by one to the registered function.

- `double` x, y: x and y coordinate of the location requested
- `BLorientation` orientation (optional): Imagery orientation: North, South, East, West or Ortho.
- `string` crs (optional): Coordinate reference system. By default: "epsg:4326", WGS84 Geographics Coordinates.

- `string` `date` (optional): Years range. Depending on the format, the date can be returned in the following years range:
  - A null value (0): Data returned will be the most recent available. This is the value by default.
  - `<year>`: Data returned will be the most recent available captured in the given year or older.
  - `<year1>-<year2>`: Data returned will be the most recent available captured between the given years.
  - `<year1>-<year1>`: Data returned will be the most recent available captured in the given year.

A discrete image in Blom SDK is represented using the class `BDiscreteImage`.

```

BDataLoader dataLoader = BSDK.CreateNewDataLoader();
dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");

double x = 1399999.7082705908D;
double y = 7493990.1591216344D;

List<BDiscreteImage> discreteImagesList = new List<BDiscreteImage>();
BLError result = dataLoader.FindDiscreteImages(discreteImagesList, x, y,
    BLorientation.BL_ORIENTATION_NORTH, "EPSG:3785", null);
if (result == BLError.BL_SUCCESS)
{
    // discreteImagesList will contain all available discretess meta data
}
  
```

```

BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader();
dataLoader1->AddDataSource(1, "blomurbex3dserver:username:password");

MyDiscreteImageList  imageList;

// MyDiscreteImageList inherited from BDiscreteImageList

double x = -4.6836912974;

double y = 40.6680158844;

if(BSDK::BL_SUCCESS == dataLoader1->FindDiscreteImages(imageList,x,y,
    BSDK::BL_ORIENTATION_VERTICAL))
{
    // imageList will contain all the availble discrete metadata
}
  
```



## 13.4 Load discrete image tile

When `BDiscreteImage` objects are delivered, they contain only the oblique ID attribute (`ID` property). In order to accede to the rest of the metadata, user has to invoke `LoadMetadata()` method. Once metadata is loaded, user can accede safely to all the getters methods of `BDiscreteImage`.

Discrete imagery can be requested in original size or downsampled, and BlomURBEX 3D Server will automatically tile this image in tiles of 256x256 pixels. User can accede to oblique imagery using `BDiscreteImage.LoadImageTile()`. This method contains the following parameters:

- `out Bitmap image/BImage image`: The client application object where the data will be loaded. In C++ version of BSDK, Client needs to pass an object derived from `BSDK::BImage` class, whereas in C# Client can pass a high level object of type `system.Drawing.Bitmap`.
- `int zoom`: Zoom level of the image:
  - 1: original size of the image
  - 2: image downsampled to a half size.
  - 3: image downsampled to a quarter size.
  - 4: image downsampled to an eighth size.
- `int x, int y`: Coordinates of the tile to request.

```
BDataloader dataLoader = BSDK.CreateNewDataLoader();  
dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");  
  
BDiscreteImage image;  
  
BLError result = dataLoader.FindDiscreteImage(out image, x, y,  
    BLorientation.BL_ORIENTATION_NORTH, "EPSG:3785", null);  
  
if (result == BLError.BL_SUCCESS)  
{  
    Bitmap bitmap;  
    image.LoadImageTile(out bitmap, 1, 2, 2);  
    // Tile is loaded into the bitmap object  
}
```

**[C# Code]**

## **13.5 Discrete images metadata**

Attributes of discrete image are provided. Some of which are listed below:

- `LoadImageTile()` : load a tile form discrete image.
- `LoadImageArea()` : load a certain area from discrete image.
- `LoadMetaData()` : load the meta data information contained in discrete image.
- `GetSelectedPixelX()/GetSelectedPixelY()` : get the selected pixel (x,y coordinate)of this photo when this photo was delivered using `FindDiscreteImage` and `FindDiscreteImages`.
- `GetOrientation()` : get the orientation of the image.
- `GetCenterLat()/GetCenterLon()` : get the center coordinates of the discrete image.
- `GetURy()/GetURx()/GetLLx()/GetLLy()` : get the corner coordinates of the discrete image.
- `GetCameraLat()/GetCameraLon()` : get the positon of the camera.
- `GetPFLHCOV()/GetPFLVCOV()` : gets the planned flight horizontal/vertical coverage for the discrete image.

## **13.6 Discrete images calculations**

Also, many calculations on discrete image are provided like:

- `TransformPoints()` : Transform a set of points from `discreteImage` pixel coordinates to World coordinates and viceversa.
- Calculations for the area of an extent defined by set, length, heights, bearing , elevation etc. Please, review the API reference to find them.

## 14 Calculations

Blom SDK provides certain functionalities for measurements.

### 14.1 Calculate elevation

Use the function CalculateElevation to measure the elevation from sea level of a given point of an ortho image. It requires following parameters:

- `out double elevation/double& elevation`: variable 'elevation' will be filled with the elevation in metres.
- `int x, int y`: Coordinates of the tile to request.
- `string crs (optional)`: Coordinate reference system. By default: "epsg:4326", WGS84 Geographics Coordinates.

```

BDataLoader urbexDataLoader = BSDK.CreateNewDataLoader();           [C# Code]

urbexDataLoader.AddDataSource(1, "blomurbex3dserver:username:password");

double urbexAlt;

if( BLError.BL_SUCCESS
== urbexDataLoader.CalculateElevation(out urbexAlt, -3.7, 40.416667))
{
    // variable urbexAlt conatins the elevation
}
  
```

```

BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader();       [C++Code]

dataLoader1->AddDataSource(1, "blomurbex3dserver:username:password");

double localAlt;

BSDK::BLError errorCode = BSDK::BL_SUCCESS;
errorCode = dataLoader1->CalculateElevation(localAlt, -4.699949, 40.654536);
if(BSDK::BL_SUCCESS == errorCode)
{
    // variable localAlt conatins the elevation
}
  
```

### 14.2 Calculate ground length

Use the function CalculateGroundLength to return the length between 2 points taking into account the ground's surface shape. It requires following parameters:

- `out double lengthInMeters/double& lengthInMeters`: variable 'lengthInMeters' will be filled with the length in metres.

- `double` numpoints, `double[]` points/`int` numpoints, `double*` points: number of output points of returned segment.
- `int` intx, `int` inty: Coordinates of the initial point.
- `int` destx, `int` desty: Coordinates of the initial point.
- `string` crs (optional): Coordinate reference system. By default: "epsg:4326", WGS84 Geographics Coordinates.

```

BDataLoader dataLoader = BSDK.CreateNewDataLoader();

dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");
double lengthInMeters;

double[] pointsArray;

if(BLError.BL_SUCCESS == dataLoader.CalculateGroundLength(out lengthInMeters,
out pointsArray, -3.7040, 40.4165240399, -3.7042, 40.4165240399) )
{
    // variable 'lengthInMeters' contains the ground length.
}
  
```

[C# Code]

```

BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader();

dataLoader1->AddDataSource(1, "blomurbex3dserver:username:password");
double lengthInMeters;
int numpoints;
BSDK::BLfloat64 *points = new BSDK::BLfloat64[22];

BSDK::BLError errorCode = dataLoader1->CalculateGroundLength( lengthInMeters,
numpoints, points, -3.7040, 40.4165240399, -3.7042, 40.4165240399);
if(BSDK::BL_SUCCESS == errorCode)
{
    // variable 'lengthInMeters' contains the ground length
}
  
```

[C++Code]

## 15 LiDAR data loading

LiDAR data loading follows the general procedure detailed in 4.2.8.

### 15.1 LoadLidar

The client application must invoke `LoadLidar()` to retrieve a LiDAR section. This method contains the following parameters:

- `BLiDAR lidar`: The client application object where the LiDAR data will be loaded. You create your own implementation of `BLiDAR` class by inheriting from it.
- `string qtreeKey`: The LiDAR location requested (as defined in 4.2.1).
- `string layer`: The LiDAR layer requested.
- `string crs`: Coordinate reference system. By default: "epsg:3785", Spherical Mercator
- `string date`: Years range. Depending on the format, the date can be returned in the following years range:
  - A null value (0): Data returned will be the most recent available. This is the value by default.
  - `<year>`: Data returned will be the most recent available captured in the given year or older.
  - `<year1>-<year2>`: Data returned will be the most recent available captured between the given years.
  - `<year1>-<year1>`: Data returned will be the most recent available captured in the given year.

```
BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader(); [C++code]

dataLoader1->AddDataSource(1, "blomurbex3dserver:username:password");

MyLidar lidar; // MyLidar derived from BSDK::BLiDAR

dataLoader1->LoadLiDar(lidar,
  "033111222301333330", "blom_lidar_10", "epsg:3785");
```

### 15.2 LoadLidarArea() [For Circle]

The client application must invoke `LoadLidarArea()` to retrieve a LiDAR section contained in a circle. This method contains the following parameters:

- **BLiDAR** lidar: The client application object where the LiDAR data will be loaded. You can create your own implementation of **BLiDAR** class by inheriting from it.
- **double** centerX, **double** centerY: Centre of the circle
- **double** radius: radius of the circle.
- **string** crs: Coordinate reference system. By default: "epsg:3785", Spherical Mercator.
- **string** date: Years range. Depending on the format, the date can be returned in the following years range:
  - A null value (0): Data returned will be the most recent available. This is the value by default.
  - **<year>**: Data returned will be the most recent available captured in the given year or older.
  - **<year1>-<year2>**: Data returned will be the most recent available captured between the given years.
  - **<year1>-<year1>**: Data returned will be the most recent available captured in the given year.

```

BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader(); [C++Code]

dataLoader1->AddDataSource(1, "blomurbex3dserver:username:password");

MyLidar  lidar; // MyLidar derived from BSDK::BLiDAR

dataLoader1->LoadLiDarArea(lidar,
centerX,centerY,1500,"blom_lidar_10","epsg:3785");
  
```

### 15.3 LoadLidarArea() [For Extent]

The client application must invoke **LoadLidarArea()** to retrieve a LiDAR section contained in an extent. This method contains the following parameters:

- **BLiDAR** lidar: The client application object where the LiDAR data will be loaded. You can create your own implementation of **BLiDAR** class by inheriting from it.
- **Double\*** points: array which contains the points which define extent.
- **int** numPoints: number of points which define the extent.

- `string` `crs`: Coordinate reference system. By default: "`epsg:3785`", Spherical Mercator
- `string` `date`: Years range. Depending on the format, the date can be returned in the following years range:
  - A null value (0): Data returned will be the most recent available. This is the value by default.
  - `<year>`: Data returned will be the most recent available captured in the given year or older.
  - `<year1>-<year2>`: Data returned will be the most recent available captured between the given years.
  - `<year1>-<year1>`: Data returned will be the most recent available captured in the given year.

```
BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader();  
  
dataLoader1->AddDataSource(1, "blomurbex3dserver:username:password");  
  
MyLidar lidar; // MyLidar derived from BSDK::BLiDAR  
  
dataLoader1->LoadLiDarArea(lidar, 5, pointsArray, "blom_lidar_10", "epsg:3785");
```

[\[C++Code\]](#)

## 16 Get available overlays

Apart from base map, BlomURBEX also offers a number of overlays which can be overlaid on map. To load overlays use `BDataLoader's` function: `GetAvailableData()` function. It requires following parameters –

- `out AvailableData[]` availableData/ `BAttributesReader` – arrays of AvailableData or object of class derived from `BAttributesReader` which will be filled after successful execution of function.
- `BLdatatype` datatype – types of overlays. Following are the possible types of overlays which can be requested –

```
public enum BLdatatype [C# Code]
{
    BL_DATA_DISCRETE_BASE_IMAGERY = 1,
    BL_DATA_DISCRETE_OVERLAY_IMAGERY = 2,
    BL_DATA_MOSSAIC_BASE_IMAGERY = 3,
    BL_DATA_MOSSAIC_OVERLAY_IMAGERY = 4,
    BL_DATA_URBAN_MODEL = 5,
    BL_DATA_TERRAIN_MODEL = 6,
    BL_DATA_SURFACE_MODEL = 7,
    BL_LIDAR_MODEL = 8,
}
```

```
typedef enum [C++ Code]
{
    BL_DATA_ANY = 0,
    BL_DATA_DISCRETE_BASE_IMAGERY = 1,
    BL_DATA_DISCRETE_OVERLAY_IMAGERY = 2,
    BL_DATA_MOSSAIC_BASE_IMAGERY = 3,
    BL_DATA_MOSSAIC_OVERLAY_IMAGERY = 4,
    BL_DATA_URBAN_MODEL = 5,
    BL_DATA_TERRAIN_MODEL = 6,
    BL_DATA_SURFACE_MODEL = 7,
    BL_DATA_LIDAR_MODEL = 8
} BLdatatype;
```



e.g. following is a sample implementation to request all mosaic imaginary overlays in a list of string –

```
public List<string> GetAvailableMosaicOverlays() [C# Code]
{
    List<string> layerSourceNameList = new List<string>();

    BDataLoader dataLoader = BSDK.CreateNewDataLoader();
    dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");

    BSDKW.AvailableData[] availableData = null;
    dataLoader.GetAvailableData(out availableData,
        BLdatatype.BL_DATA_MOSSAIC_OVERLAY_IMAGERY);

    foreach (BSDKW.AvailableData data in availableData)
    {
        layerSourceNameList.Add(data.Layer);
    }

    return layerSourceNameList;
}
```

```
BSDK::BDataLoader *pDataLoader = BSDK::CreateNewDataLoader(); [C++ Code]

BSDK::BLError errorCode = BSDK::BL_SUCCESS;

AttributesReader attrReader;

// AttributesReader inherited from BAttributesReader

pDataLoader->AddDataSource(0, "BlomURBEX3DServer:username:password");
errorCode = pDataLoader->GetAvailableData(attrReader,
    BSDK::BL_DATA_DISCRETE_BASE_IMAGERY);
if(errorCode == BLError.BL_SUCCESS)
{
    // Make use of AttributesReader object
}
```

## 17 Geocoding with Blom SDK

You can perform geocoding and reverse geocoding using BSDK as described below –

### 17.1 Geocoding

Geocoding is the process of finding associated geographic coordinates from other geographic data, such as street addresses, or zip codes etc. You can use `BDataLoader.PerformGeocoding()` function to achieve this. It accepts following parameters –

- `BAttributesReceiver` `attributesReceiver` – you will receive result in this attribute receiver class. Refer to 6 for more details on `BAttributesReceiver`.
- `string` `location` – the value for which geocoding needs to be performed.

e.g. Below is a sample implementation for perform geocoding of a city –

```
[C# Code]
BDataLoader dataLoader = BSDK.CreateNewDataLoader();
dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");

MyAttributesReceiver attributes = new MyAttributesReceiver();
BLError result = dataLoader.PerformGeocoding(attributes,
    "Tower of London, City of London, United Kingdom");
if (result == BLError.BL_SUCCESS)
{
    // MyAttributesReceiver will receive required data
}
```

```
[C++ Code]
BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader();
dataLoader1->AddDataSource(0, "BlomURBEX3DServer:london:paris");

BSDK::BLError errorCode = BSDK::BL_SUCCESS;

AttributeReaderStdoutPrinter attributeTest;
// AttributeReaderStdoutPrinter inherited from BAttributesReader

dataLoader1->PerformGeoCoding(attributeTest, "Tower of London, City of London,
United Kingdom");

if( errorCode == BLError.BL_SUCCESS){

    // AttributesReaderAssertor will receive require data.

}
```

Output of the above request will be received in attribute receiver and will be –

```
Attribute suggestion Value "Tower of London, The Queen's Steps, Bermondsey,
London Borough of Tower Hamlets, Greater London, England, SE1 2AA, United
Kingdom"

Attribute location Value -0.0760405941217587, 51.5080489730794

Attribute height Value -1.0
```

## 17.2 Reverse geocoding

Reverse geocoding is the opposite: finding an associated textual location such as a street address, from geographic coordinates. You can use `BDataLoader.PerformReverseGeocoding()` function to achieve this. It accepts following parameters –

- `BAttributesReceiver attributesReceiver` – you will receive result in this attribute receiver class. Refer to 6 for more details on `BAttributesReceiver`.
- `double x, y` – coordinate of the location for which you want to perform reverse geocoding.
- `int level` – the zoom level for which you want the reverse geocoding.

e.g. Below is a sample implementation for performing reverse geocoding for the same location we get from geocoding example above –

```
BDataLoader dataLoader = BSDK.CreateNewDataLoader(); [C# Code]
dataLoader.AddDataSource(1, "blomurbex3dserver:username:password");

AttributesReceiverStdoutPrinter attributes = new AttributesReceiverStdoutPrinter
();
BLError result = dataLoader.PerformReverseGeocoding(attributes,
    -0.0760405941217587D, 51.5080489730794D, 1);
if (result == BLError.BL_SUCCESS)
{
    // MyAttributesReceiver will receive required data
}
```

```
BSDK::BDataLoader *dataLoader1 = BSDK::CreateNewDataLoader();           [C++ Code]
dataLoader1->AddDataSource(0, "BlomURBEX3DServer:london:paris");

BSDK::BLError errorCode = BSDK::BL_SUCCESS;

AttributesReaderAssertor attributeTest;
// AttributesReaderAssertor inherited from BAttributesReader

dataLoader1->PerformReverseGeocoding(attributeTest, -0.0760405941217587D,
51.5080489730794D, 1);

if( errorCode == BLError.BL_SUCCESS){

// AttributesReaderAssertor will receive require data.

}
```

Output of the above request will be received in attribute receiver and will be –

```
address = "Tower of London, The Queen's Steps, Bermondsey, London Borough of
Tower Hamlets, Greater London, England, SE1 2AA, United Kingdom"
```



---

## **Blom SDK Reference**

For further information please contact Blom BIS Product Management at [BISproducts@blomasa.com](mailto:BISproducts@blomasa.com) or contact your local Blom sales representative.