# RDF BCF library

RDF BCF library is a tool to facilitate BCF exchange:

* Clear structures to create or read BCF data, uniform for all supported versions.
* Automatic populating data (creation/edition information, extending schema with users, stages, statuses, etc.)
* Automatic management for embedded (internal) files.
* Keep data consistent on changes.
* Data validation on saving.

# API

Library supports following types of APIs

* C++ Objects API in bcfAPI.h
* C API in bcfEngine.h can be used to easily interoperate with other languages.
* .Net (C#) Objects API in RDF.BCF assembly

# How to build, distribute and use

Build:

Open BCF.sln, it contains

* bcfEngine: core implementation and C/C++ APIs
* RDF.BCF: .Net APO
* CSExample: examples and smoke-tests

Build and run target BCSExample Release x64 – it include all targets and executes smoke tests.

Win32 target is not here but can be added.

Distribution:

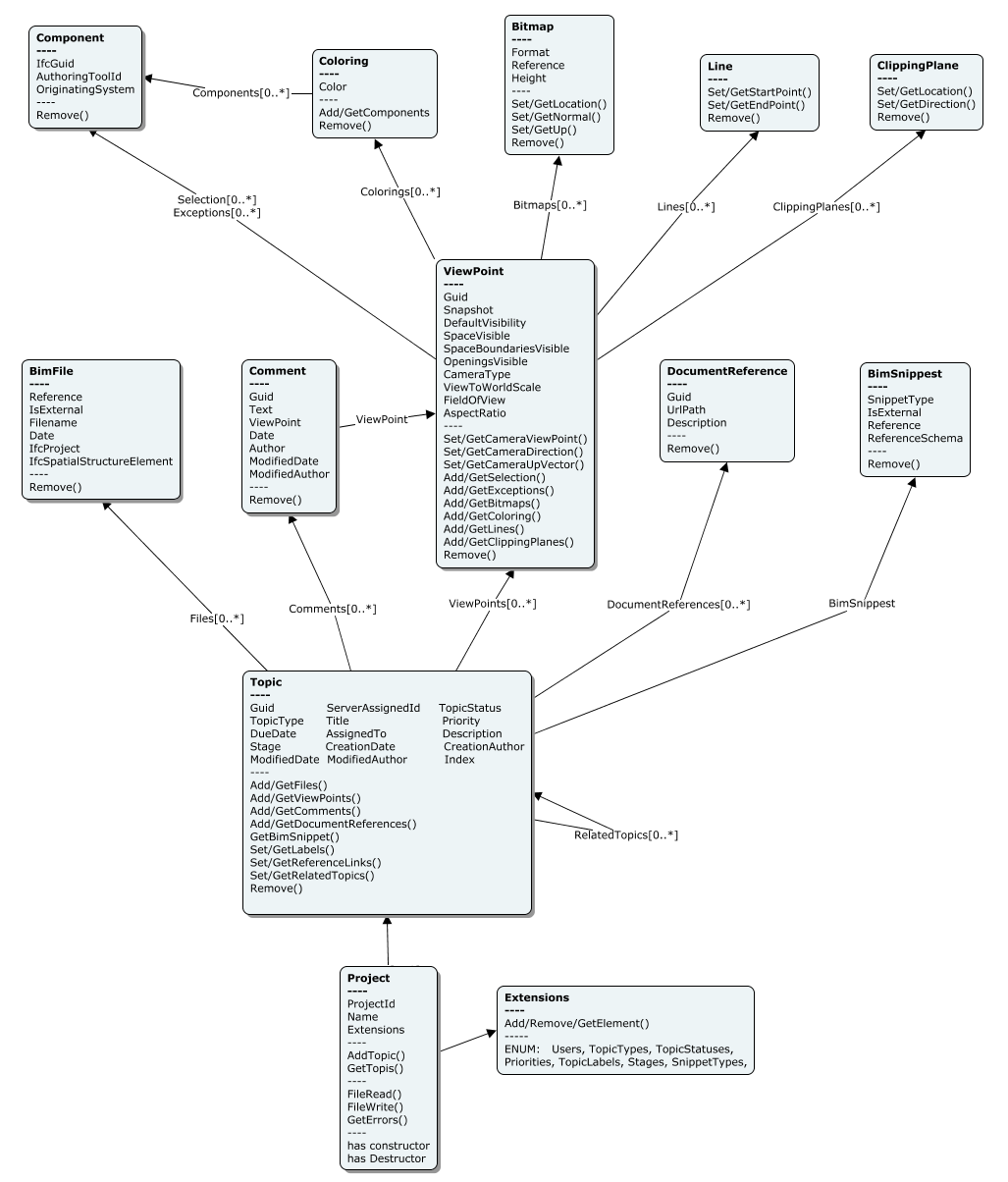
|  |  |
| --- | --- |
| output\bcfEngine.dll  output\zip.lib  output\zlib.dll | Always required |
| output\Release\net8.0\RDF.BCF.dll | Required for .Net applications |

Usage:

|  |  |
| --- | --- |
| C# | Add reference to RDF.BCF.dll |
| C | #include “bcfEngine.h”, link with bcfEngine.lib |
| C++ | #include “bcfAPI.h”, link with bcfEgine.lib |

See example of usage in CSExample\Example.cs

# RDF BCF Object Model



# Name conventions

In C# properties are implemented as property methods of classes

in C++ methods Get<PropName>() and Set<PropName>() are used.

In C API names of functions follow the template **bcf**<Class>**Get/Set**<PropName> and **bcf**<Class>**Add/Iterate**<Class2>

# Memory and Objects Lifetime Management

All resources are allocated in the context of Project, and they remain valid until Project object is closed. When the Project is closed all associated objects are disposed, pointers become invalid.

A caller should not allocate or create any object except Project, but use Add<Obj>(...) method of container object.

A caller should not delete or dispose any object except Project, but use Remove() method to abandon object from BCF data.

All objects must be used only until Project is closed/disposed.

If a method returns char pointer, the pointer is valid until next call to the method.

For C# it is recommended to use “using” statement.

# File management

RDF BCF library will automatically copy embedded (internal) files to/from filesystem.

After reading BCF data all Reference properties will point to real filesystem location.

When a caller sets Reference property it should be an existing file. The file must exist at the moment of setting and until Project.FileWrire() call. It will be copied to internal location on write.

Temporary files are cleaned when Project is closing.

# Error handling

When an error is happening, it is added to the project log and can by inquired with Project.GetErrors() method any time later.

Project.GetErrors() can be called after any method returns unsuccess (false, null or empty string) or after seria of calls.

Subject to discuss: We are trying to avoid exception in .Net and they are used only in Add<Obj> methods to avoid returning null. Maybe it is reasonable to return null and avoid exceptions at all, or use it also when property set fails.

# Partially implemented or not implemented yet (TODO)

* Cascade objects remove
* Embedded (internal) documents
* Read/write previous versions (actually, BCF 3.0 is implemented)
* Subject to discuss usage .Net exceptions
* Implement Linux support if needed
* Use ifcEngine to get BimFile properties and validate ifcGuids