BIMcollab

Connection library
Implementation guide

Version 5.0

KUBUS

The BIMcollab SDK

BIMcollab is a cloud based issue management service for the AEC industry. The BIMcollab ecosystem consists of the cloud service itself and a wide variety of connected applications such as BIMcollab ZOOM and the BIMcollab BCF Managers.

These connected applications make use of the *BIMcollab SDK* in order to transfer data between themselves and the BIMcollab service.

The BIMcollab SDK has currently three parts:

- Connector with GUI included
- Connection Library (no GUI)
- BCF-API (beta)

This manual explains the possibilities and the usage of the BIMcollab Connection Library. The library is available for development in C# and Java.

Content

General concepts

We will refer to the library as 'BIMcollab Connection library' from here.

Operation mode

The BIMcollab Connection library can only operate in online mode, meaning that a connection to a project on a BIMcollab space needs to be set up first thing during run-time.

In online mode the BIMcollab Connection library manages the connection with the project on a BIMcollab server, retrieves data from it and takes care of the local datamanagement.

In online mode the BIMcollab Connection library automatically retrieves the minimum amount of data needed for the operations requested. Therefore it is possible that accessing data takes a bit of time. However once the data is retrieved from the server it is automatically cached and instantly available on any following requests.

Collections

In many cases, data is stored in collections. Data in these collections can be accessed by their index number (starting with 0). Of course the number of items available in a collection can also be retrieved.

It is important to note that these indexes are not fixed. After a data reload, a specific item from a collection may end up with a different index.

The index numbers are used for setting properties, so make sure you are always using the correct indexes and do not rely on indexes being fixed or values from previous sessions.

Indexes can change on reloading or refreshing of the BIMcollab Connection library, when selecting a project and when removing elements from collections (e.g., issue labels, viewpoint components).

Active project

The BIMcollab Connection library contains a collection of projects, but all operations on issues require an active project.

There can be only one project active at any time and it must be selected from the projects available. You cannot create a new project in BIMcollab via the library. Once the project is selected, a reference to it can be retrieved for further use.

Operations done on projects that are not the active project may fail and data will not be able to be published or saved. However the project properties can be retrieved from these projects for display purposes.

Please note that the active project is a copy of one of the projects in the BIMcollab Connection library and indices retrieved from other projects must not be used on the active project.

BIMcollab user

The BIMcollab Connection library needs user information. This user is the author of any changes within the BIMcollab Connection library and his/her credentials are used for making the connection with a BIMcollab server.

App-token

The BIMcollab Connection library uses a token in the communication with a BIMcollab server. For each app you build you need a different token. Also apps running as add-ons in BIM tools need separate tokens for each version of the BIM tool. This way we can track which apps connect to our servers track of how many times a certain It is required that you make use of the token(s) provided to you.

For more information about tokens, or to get additional tokens, please contact us at developer@bimcollab.com.

Errors and Exceptions

Errors are communicated through exceptions, so when retrieving data from the BIMcollab Connection library, the data retrieved will be valid, unless an exception occurred.

In the example code below the exception handling is left out for a more clear view on the BIMcollab Connection library calls.

Getting Started

The entry point for the BIMcollab Connection library is the *BC_ProjectManager* class, an instance of it can be retrieved with the static GetInstance method.

Example code:

```
Java / C#:
BC_DataModel pm = BC_DataModel.GetInstance();
```

How to start working with a project

The general steps are:

1. Specifying the user
2. Connecting to a BIMcollab server
3. Inspecting the list of projects available to you
4. Select the required project

Example code:

```
Java / C#:
pm.SetActiveUser("", "", "<password>", "<email>", "");
pm.Connect("<BIMcollab space>", "<app-token>");

long nProjects = pm.GetNumberOfProjects();

for (long p = 0; p < nProjects; p++)
{
    pm.GetProjectByIndex(p);

    // E.g. Show the project data (title, description) to the user
}

// User wants to use project with index x
pm.SelectProjectByIndex(x);
BC_Project pProject = pm.GetActiveProject();
```

In *SetActiveUser* only the email address and password are relevant. The other arguments are not used and should be kept blanc.

In *Connect* the <BIMcollab space> is the full domain name of the targeted BIMcollab space (e.g., "join.bimcollab.com"). The token is the developers app-token you received from BIMcollab to identify your application.

The active user will have a specific role in a project. Depending on that role the user is allowed or disallowed to execute certain operations:

- The roles 'Project leader' and 'Editor' are allowed any operation.
- The role 'reviewer' can retrieve any value but is only allowed to 'Approve' or 'Disapprove' issues and to add a comment.
- The role 'Viewer' can only retrieve any value and cannot make any change.

Special connection settings

When the use of a proxy server is required, the proxy server address, port and credentials need to be specified before connecting to the BIMcollab server. The settings can be tested using the TestConnection method of the *BC_ProjectManager*.

Example code:

```
Java / C#:
pm.SetProxy(L"<proxy address>", <proxy port>, L"<proxy username>", L"<proxy password>");
pm.TestConnection(L"join.bimcollab.com");
```

Getting the properties

Issue properties can be retrieved from a *BC_Project*. Some of the property values can be inactive and should not be used when creating new issues. Those property values are needed for display purposes only. Whether a value is 'active' can of course be retrieved from the *BC_Project*.

Milestones	GetNumberOfMilestones( )	GetMilestoneLabelByIndex( )	GetMilestoneActiveByIndex( )
Areas	GetNumberOf Areas( )	GetAreaLabelByIndex( )	GetAreaActiveByIndex( )
Labels	GetNumberOfIssueLabels( )	GetIssueLabelLabelByIndex( )	GetIssueLabelActiveByIndex( )
Types	GetNumberOfIssueTypes( )	GetIssueStatusLabelByIndex( )	GetIssueStatusActiveByIndex( )
Priorities	GetNumberOfIssuePriorities( )	GetIssuePriorityLabelByIndex( )	GetIssuePriorityActiveByIndex( )
Status	GetNumberOfIssueStatuses( )	GetIssueStatusLabelByIndex( )	GetIssueStatusActiveByIndex( )

Please carefully read the section ‘Special note on working with property or user collections’.

Example code:

```
Java / C#:
long nrStatuses = pm.GetNumberOfIssueStatuses();

for (long i = 0; i < nrStatuses; i++)
{
    String status = pm.GetIssueStatusLabelByIndex(i);
}

unsigned int nrLabels = pPm->GetNumberOfIssueLabels();

for (i = 0; i < nrLabels; i++)
{
    std::wstring label = pPm->GetIssueLabelLabelByIndex(i);
    boolean active = pm.GetIssueLabelActiveByIndex(i);
}

long nrMilestones = p.GetNumberOfMilestones();

for (long i = 0; i < nrMilestones; i++)
{
    String milestone = project.GetMilestoneLabelByIndex(i);
    boolean active = project.GetMilestoneActiveByIndex(i);
}
```

Getting the users

Users are special Issue properties. The user collection is in fact a collection of all team members in the selected project.

This collection also contains the removed project team members. Again these are provided for diaplay purposes only as some issues may still refer to them.

Only the owner (team member to which the issue is assigned) of an issue can be modified through the BIMcollab Connection library.

Issues can only be assigned to team members that are assignable. This can be checked using the *IsAssignable* method

Please carefully read the section ‘Special note on working with property or user collections’.

Example code:

```
Java / C#:
long nrUsers = p.GetNumberOfUsers();

for (long i = 0; i < nrUsers; i++)
{
    BC_User user = project.GetUserByIndex(i);
    boolean assignable = user.IsAssignable();
}
```

Working with issues

Issues can be created in and retrieved from the active project. Once you have a reference to an issue, you can read and modify its properties and add or access comments.

Example code:

```
Java / C#:
long nrIssues = project.GetNumberOfIssues();

BC_Issue issue = project.GetIssueByIndex(x);

// Issue: getting and setting data
String title = issue.GetTitle();
BC_User user = issue.GetOwner();
BC_User otherUser = project.GetUserByIndex(2);
issue.SetOwner(otherUser);

long nrComments = issue.GetNumberOfComments();
BC_Comment comment = issue. GetCommentByIndex ();
BC_Comment newComment = issue.CreateComment();

// Create a new issue
BC_Issue* pNewIssue = project.CreateIssue();
```

Setting the state of an issue is bound to rules. For instance it is not allowed to change the state from active to closed, an issue must be published in resolved state first. To find out if a transition is allowed, the *BC_Issue* provides the IsOperationAllowed method.

Example code:

```
Java / C#:
if (issue.IsOperationAllowed(0, x))
{
    issue.SetStatus(x);
}
```

Issue message queue

Each issue maintains its own message queue. This queue contain information regarding operations performed for this issue with the BIMcollab server.

Messages are implemented in the *BC_Message* class and have a code and a text explaining in more detail the reason for setting the message.

A special type of message is the error message, which is any message with a code greater than 0.

Error messages can be active. This is the case for errors that occurred since the last operation with the BIMcollab server.

Any new operation will deactivate all errors before starting the operation. Methods that can trigger such operations are *Reload*, *Refresh*, *Publish*. Data loading operations that work in the background can also set error messages. Using the message queue it is therefore possible to review what happened with the issue.

Example code:

```
Java / C#:
unsigned int nrMessages = issue.GetNumberOfMessages();

BC_Message message = issue.GetMessageByIndex(x);

unsigned int code = message.GetCode();
std::wstring text = message.GetText();

if (issue.HasActiveError())
{
    message = issue.GetActiveError();
}
```

Working with Comments

Comments contain a comment text and some properties. Besides that, they can have a link to a viewpoint. The viewpoint related to a comment can be retrieved and a viewpoint can be created or set to an existing viewpoint.

Creating a viewpoint or setting a reference to an existing viewpoint replaces the existing link, the previous viewpoint is not lost.

Example code:

```
Java / C#:
long nrComments = issue.GetNumberOfComments();
BC_Comment comment = issue.GetCommentByIndex(3);
BC_ViewPoint viewpoints = comment.CreateViewPoint();

or

BC_ViewPoint newViewPoint = issue.CreateViewPoint();
comment.SetViewPoint(pNewViewPoint);
```

Working with Viewpoints and Snapshots

Viewpoints contain all of the information required to recreate a view within a BIM application.

A viewpoint consists of:

- A camera with it's position and direction
- A list of components identified by their IFC GUID and stored with their visible state, selected state and override color
- A set of sectioning planes

Please note that a viewpoint does not require components or sectioning planes.

Example code:

```
Java / C#:
BC_Comment comment = issue.GetCommentByIndex(3);

if (comment->HasViewPoint())
{
    BC_ViewPoint viewPoint = comment->GetViewPoint();

    // Properties
    unsigned int projectionType viewPoint->GetProjectionType();
    BC_3DPoint cameraDirection = GetOrthogonalCameraDirection();

    BC_3DPoint newCameraDirection(10, 20, 30);
    viewPoint.SetOrthogonalCameraDirection(newCameraDirection);

    // Components
    unsigned int nrComponents = viewPoint.GetNumberOfComponents();
    BC_Component component = viewPoint.GetComponentByIndex(5);
    viewPoint.RemoveComponentByIndex(2); // Changes the
    BC_Component newComponent = viewPoint.CreateComponent("<ifcGuid>", "red", true, true);
}
```

Viewpoints can have a snapshot, but that is not mandatory.

BC_SnapShots contains the image data in 2 variants: image and thumbnail, where the thumbnail is a resized version of the image.

Example code:

```
Java / C#:
BC_SnapShot snapShot;

if (viewPoint->HasSnapShot())
{
   snapShot = viewPoint.GetSnapShot();
}
else
{
    snapShot = viewPoint.CreateSnapShot();
}

long imageSize = snapShot.GetImageSize();

if (snapShot.HasImage())
{
    byte[] image = new byte[(int)imageSize];
    snapShot.GetImage(image, imageSize);

    // Do something with the raw image data
}
else
{
    // Get raw image data (preferably PNG), size is x
    byte[] image = new byte[(int)imageSize];
    snapShot.SetImage(image, x);
}
```

Publishing your work

In online mode the way to store your work is by publishing the issues.

Basic publish workflow

You can indicate which issues to publish by setting a flag. Only issues that were modified and have their publish flag set are actually published.

Example code:

```
Java / C#:
project.GetIssueByIndex(2).IncludeInPublish(true);
project.GetIssueByIndex(8).IncludeInPublish(true);
bool result = project.Publish();
```

After a successful publish of an issue, it is marked as unchanged. The publish flag is not changed.

Publish error handling

When the *Publish* operation fails, the individual issues that failed to publish have their error set. Please refer to 'Issue message queue' for information on how to use the message issue queue.

One special case of publish failure needs to be treated different from the general failure, which cannot be resolved. That special failure happens when the issue was modified on the BIMcollab server after it was retrieved with the BIMcollab Connection library. In this case the issue active error code is 3. The publish can be repeated while forcing the BIMcollab Connection library version to be published.

It is advised to only do so after the user has evaluated what was changed on the server!

To enable that, the issue provides a link to itself on BIMcollab.

Example code:

```
Java / C#:
if (issue.HasActiveError())
{
    BC_Message message = issue.GetActiveError();

    if (message.GetCode() == 3)
    {
        string url = issue.GetBIMcollabUrl();

        // Use url to show issue and interact with user about overwriting the changes

        if (overwrite) // User chose to overwrite changes on server
        {
            issue.OverwriteServerChanges();

            issue.Publish();
        }

        if (reload) // User chose to discard own changes
        {
            issue.Reload();
        }
    }
}
```

Publish progress monitoring

During the *Publish* the progress can be monitored.

Example code:

```
Java / C#:
// Start Publish on another thread

// action = 1 indicates publish in progress
// action = 2 indicates publish is finished

BC_Progress progress = project.GetProgress();

while (progress.action < 2)
{
    progress = project.GetProgress()
}
```

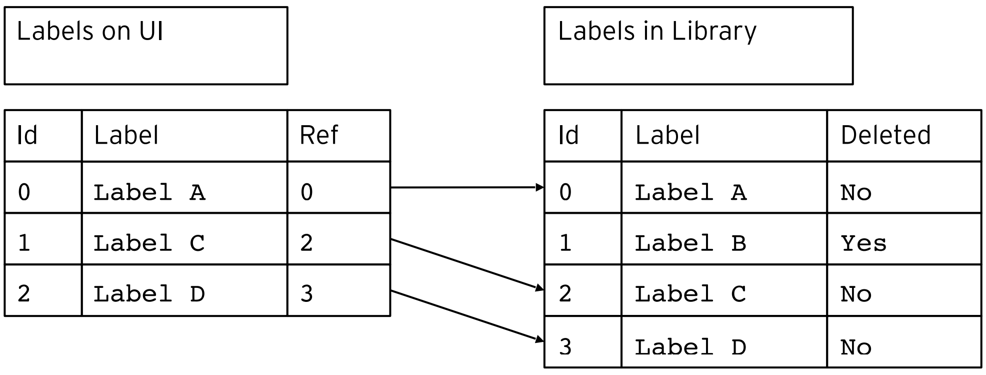
Special note on working with property or user collections

When working with mentioned collections, please take into account that these collections can contain state information. Under certain conditions, values with a specific state should not be presented to the users of the BIMcollab Connection library.

The collections always contain all values that are used at any time during the lifetime of the project. During a project team members get removed and new member are added, milestones are created and deleted and so on.

When presenting lists of properties to a user to select from, only the active properties must be used. The integrator of the BIMcollab Connection library must take care of a proper mapping of the properties presented to the user and the properties in the library.

Example:



Labels on UI

Id	Label	Ref
0	Label A	0
1	Label C	2
2	Label D	3

Labels in Library

Id	Label	Deleted
0	Label A	No
1	Label B	Yes
2	Label C	No
3	Label D	No

When presenting a list of team members to whom an issue is to be assigned the same principles apply, but then based on the 'IsAssignable' property of the *BC_User*.

Property	Usage	Check
Issue status	Selection options: only active	Project.GetIssueStatusActiveByIndex
Issue type	Selection options: only active	Project.GetIssueTypeActiveByIndex
Issue priority	Selection options: only active	Project.GetIssuePriorityActiveByIndex
Issue label	Selection options: only active	Project.GetIssueLabelActiveByIndex
Issue visibility	Selection options: only active	Project.GetIssueVisibilityActiveByIndex
Favorites set membership	Selection options: only active	Project.GetFavoritesSetActiveByIndex
Issue area	Selection options: only active	Project.GetAreaActiveByIndex
Issue milestone	Selection options: only active	Project.GetMilestoneActiveByIndex,
Issue owner	Selection options: only assignable	User.IsAssignable
Issue approver list	Selection options: users that can become approver	User.CanBecomeApprover
Issue notification list	Selection options: users that can be notified	User.CanBeNotified