

Connector with full GUI Implementation Guide

Version 7.0.5

The BIMcollab SDK

BIMcollab is a cloud-based issue management, BIM coordination, model checking, quantity takeoff, document management and asset management service for the AEC industry. The BIMcollab ecosystem consists of the cloud service (Nexus) and a wide variety of connected applications such as BIMcollab Zoom and the BIMcollab BCF Managers.

The BCF Managers are issue management plug-ins created by BIMcollab, that integrate into 3rd party CAD and BIM authoring software (e.g. Revit, AutoCAD etc.) that facilitate bi-directional data transfer with BIMcollab Nexus.

By deploying the BIMcollab SDK, 3rd party developers can also create their own plug-in to the authoring tool of their choice.

The BIMcollab SDK has two types:

- Connector with GUI included
- Connection API (no GUI)

This manual explains the possibilities and the usage of the Connector with GUI.

Content

	1
Connector with full GUI	1
1	
The BIMcollab SDK	2
Content	3
Introduction	4
Requirements	5
Package Contents	6
Change Log	7
Getting Started	9
Development	11
Vendor specific values	11
Initialize()	11
SaveOnExit()	11
AppInterface	12
Properties	12
Methods	14
Settings	20
Viewpoint usage	21
Setup ViewPoint	21
Components	21
Creating a component set	22
ClippingPlanes	22
Modeless Dialogs and Multi-Threading	23

Introduction

The BIMcollab Connector is an ASP.net user control which provides the full functionality of the BIMcollab BCF Manager for integration with 3rd party applications.

To allow the user control to communicate with the 3rd party application (e.g. setting or retrieving the scene setup and taking snapshots) a glue layer is provided, the AppInterface, which needs to be implemented by the 3rd party application developer.

Any application will require an app-token to be allowed to connect to BIMcollab spaces. During development the generic developer token 460C8E9C-20E4-4188-BCA2-8E34F464C026 can be used. This token will give you access to the development space: playground.bimcollab.com.

Once development has been completed, you can submit your software for testing and receive an application specific app-token which will allow your application to connect to all BIMcollab spaces.

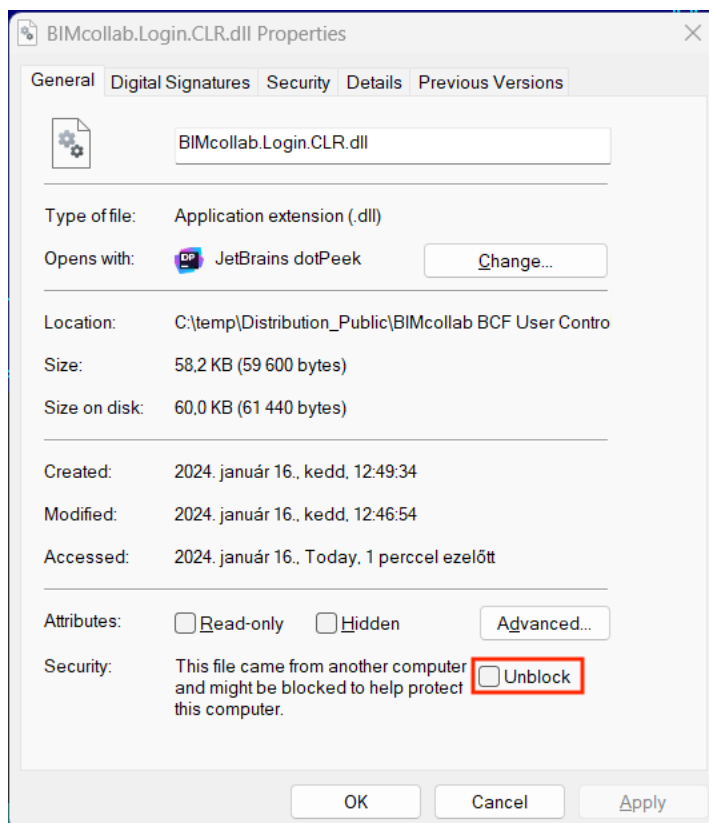
The BIMcollab Connector has been limited to work online only. The ability to support an offline workflow using BCF files with the Connector is available only on request.

Any end user wishing to use any BIMcollab application, including 3rd party applications using the user control, must activate the software using their BIMcollab ID.

Requirements

The following requirements must be met to make use of the BIMcollab Connector:

- The 3rd party application must be capable of integrating a UserControl built for .NET Framework 4.8
- The 3rd party application must provide methods to retrieve and set camera settings, clipping planes as well as the visible and/or selected components
- The 3rd party application must be able to run with the 'Prefer 32-bit' flag in the 'Build – General settings' switched off
- The Microsoft Visual C++ 2010 Redistributable Package must be added as a prerequisite for your application.
- Right click on the following files and open the Properties panel to unblock the security measures that are in place for all files downloaded from the internet:
BIMcollab.Login.CLR.dll, BIMcollab.Login.Webview.dll and 'BCFLicensemanger.dll'



Package Contents

The BIMcollab Connector package contains the following files:

The Source folder contains the project and source files which can be used to implement the ApplInterface:

- `App_Interface.cs`.
A template for the implementation of the ApplInterface.
- `ViewPoint.cs`
Contains the definitions of the viewpoint data structures used to communicate between the ApplInterface and the UserControl.
- `AssemblyInfo.cs`
Basic assembly information for the library (e.g. version, copyright).
- `AppInterface.csproj`
A Visual Studio project for building the ApplInterface.

The Redistributable folder contains the files which need to be distributed with the 3rd party application:

- `BCFManagerControlLib.dll`
The class library that defines the user control.
- `BIMcollab_cs.dll`
A support library for handling the communication with BIMcollab servers.
- `BCF LicenseManager.exe`
A support executable required for handling the activations.
- `BIMcollab.Login.Webview.dll`
A support library required for activation and login via our identity server.
- `BIMcollab.Login.CLR.dll`
A support library required login via our identity server
- `Microsoft.IdentityModel.JsonWebTokens.dll`
`Microsoft.IdentityModel.Logging.dll`
`Microsoft.IdentityModel.Tokens.dll`
`Newtonsoft.Json.dll`
A support library required for activation via our identity server.
- `BCFMCORE.dll`
A support library that is used by the `BCFManagerControlLib.dll`.

The Resources folder contains official icons and images which can be used within the 3rd party application.

Change Log

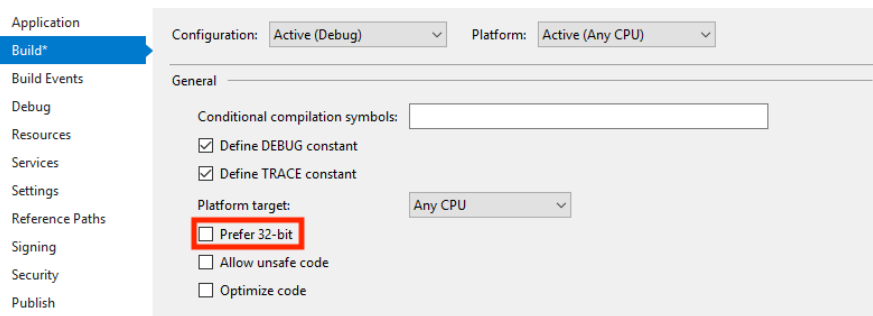
Version	Release date	Changes
5.0.8.361	30.08.2018	Initial release
5.0.9.185	30.11.2018	Pre R3 release <ul style="list-style-type: none"> - components in viewpoint feature now configurable - added Toolkit folder to avoid Visual Studio issues with 64bit components - updated setup instructions
5.0.9.253	24.01.2019	R3 release <ul style="list-style-type: none"> - updated setup instructions
5.2.10.182	17.06.2019	5.2 release <ul style="list-style-type: none"> - removed Toolkit folder as is no longer needed - updated setup instructions and commentary in source code - limited to on-line workflow (offline available only on request)
5.2.10.226	31.07.2019	5.2 update <ul style="list-style-type: none"> - added visibility to issue structure (see GetDictionariesDelegate)
5.3.11.28	22.04.2020	5.3 release <ul style="list-style-type: none"> - added support for custom fields - added support for mandatory fields - added variable to store client
5.4.11.314	25.02.2021	5.4 release
5.5.36.0	27.10.2021	5.5 release <ul style="list-style-type: none"> - Dutch & Portuguese languages support - Community translation support - Issues can now set without assigned team member - Improved add/edit issue dialog - Improved interaction while trying to edit someone else's comment
5.6.34.0	29.07.2022	5.6 release <ul style="list-style-type: none"> - More secure activation and login via identity server
6.0.10	29.08.2022	6.0 release <ul style="list-style-type: none"> - Redesigned user interface - public const bool / hasButtonQuickZoom is discontinued - public const bool / hasButtonToggleColor is discontinued

Version	Release date	Changes
6.1.18	12.01.2023	6.1 release - BCFMCore.dll has been added to Redistributables - All DLLs are digitally signed - Features: Save filters, Auto-sync and Set viewpoint as default have been added
6.2.16	10.03.2023	6.2 release - Features: Approval and relational operators on Filter dialog
7.0.5		7.0.5 release - Features: IDS palette

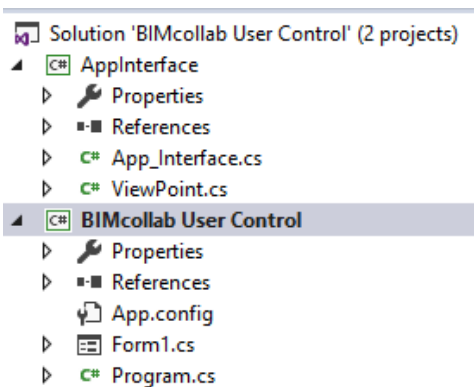
Getting Started

To quickly get started with adding the user control to your application, please follow the following steps in order:

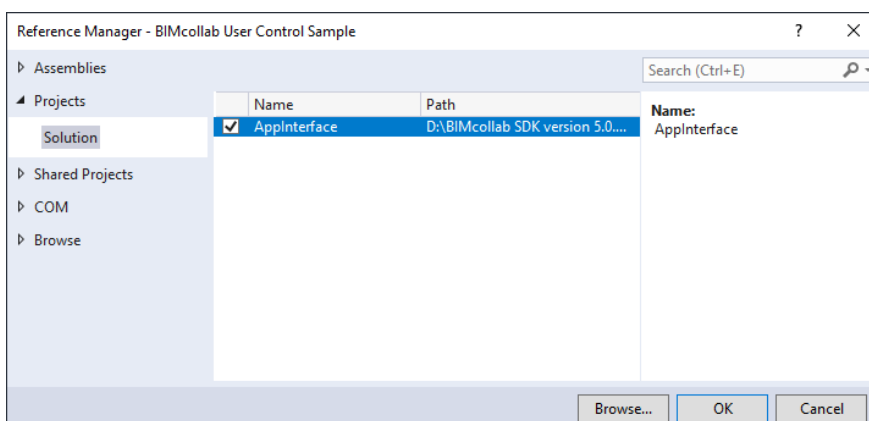
1. In your project's properties, ensure that the 'Prefer 32-bit' checkbox under Build>General is not checked.



2. Add the AppInterface project to your solution. This can be found in the Source folder. Add a reference to

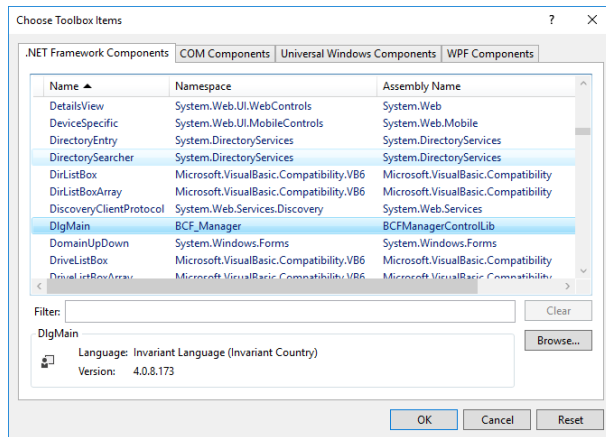


3. Projects > Solution > Appinterface to your project

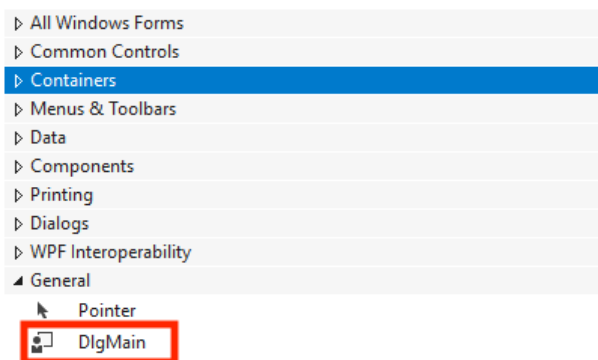


4. Build the solution
5. Open the Form you wish to add the Custom Control to in the form designer*

6. Add the user control to your toolbox:
 1. In the toolbox select “choose items”
 2. In the “choose toolbox items” dialog browse to the “BCFManagerControlLib.dll” found in the Redistributable folder.



7. Add the control to a form by dragging it from the ToolBox (General > DlgMain)



8. Implement the AppInterface (see section AppInterface)
9. Implement a call to the control’s “Initialize” method
10. Implement a call to the control’s “SaveOnExit” method
11. Copy all redistributable from the appropriate Redistributables folder to the project’s output folder
12. Build and run your application.

* You may need to restart Visual Studio before the visual editor will find the appinterface dll.

Development

When making use of the user control there are four required development steps that must be taken:

1. Set your vendor specific values
2. Call the control's "Initialize" method
3. Call the control's "SaveOnExit" method
4. Add the necessary functionality to the AppInterface
5. Add installation path to the registry

Vendor specific values

Every vendor will receive from KUBUS a document containing several vendor/application specific values which must be set in the App_Interface.cs file:

public static string developersToken

The app-token will uniquely identify the application when interacting with the BIMcollab eco-system.

public static string developersRegValPrefix

This string will allow settings to be stored in the registry uniquely for this application.

public static string developersClientID

This string is used internally by BIMcollab to identify the client used to create an issue.

Furthermore, every vendor will be required to create a registry entry which will store the full installation path of the custom control binaries so that these can be found and updated by update installers created by KUBUS. The registry key is defined in the in the same document as mentioned above.

Initialize()

Calling this method will initialize the control and make it ready for use.
Prior to calling this method, all controls within the user control will be disabled.

SaveOnExit()

When the 3rd party application closes, it is best practice to provide the end user the opportunity to save any changes made within the user control.

This can be done by calling the SaveOnExit method.

If it is possible to abort the closing sequence, then SaveOnExit can be called with showCancelButton = true. This will add a "Cancel" button to the message dialog.

AppInterface

To fully integrate user control with the 3rd party application, it is necessary to implement functionality for most of the AppInterface's methods.

The AppInterface also has defined several properties that can be used to hide / show specific elements on the user control.

Properties

public const bool

hasModel

Set this property to 'true' when the 3rd party application supports interaction with (IFC) models such as getting and setting camera position, manipulating component visibility, and creating snapshots.

public const bool

hasButtonBoundingBox

Set this property to true when the 3rd party application supports bounding boxes (a.k.a. clipping planes or section planes). With the button the user can instruct the 3rd party application to set a bounding box around the part of the model visible at that moment.

See also the method SetBoundingBox.

public const bool

hasButtonImportIssues

Set this property when the 3rd party application supports the creation of multiple issues (e.g clash detection, saved viewpoints, test results) which can be imported directly into the issue list.

See also the method ImportIssues.

When enabled, you can define the text used for the import button by defining the property importIssuesToolTip.

public const bool

hasButtonMinMax

Set this property to allow the user control to provide a button with will toggle the user control between its normal and a minimal layout.

public static bool

hasComponentsInViewpointOptionVisible

Set this property to allow the "Visible" option to be added to the Component in Viewpoint feature in the Add Issue dialog.

This feature can be disabled when your application does not add components to viewpoints or if your application only adds with selected components.

public static bool

hasComponentsInViewpointOptionSelected

Set this property to allow the “Selected” option to be added to the Component in Viewpoint feature in the Add Issue dialog.

This feature can be disabled when your application does not Set this property to allow the “Visible” option to be added to the Component in Viewpoint feature in the Add Issue dialog.

This feature can be disabled when your application does not add components to viewpoints or if your application does not work with selected components.

public static bool

hasSaveSelectionAsSelectedComponentsOption

Set this property to enable the “Save selection as...” feature in the Add Issue dialog.

This feature can be disabled when your application does not add components to viewpoints or if your application does not work with selected components.

Methods

public static void

Set...Delegate()

public static void

These 'Delegate' methods can be used by the user control to specify the delegates for various helper functions.

Do not modify these methods.

The helper function can be used by the 3rd party developer:

void AddZoomMessage(string message)

In the case where the 3rd party application is not able to recreate the view during a zoom to (see ZoomTo3D method) an error message can be defined using this helper function.

The error message can be composed of one or more messages; each added with a call to AddZoomMessage.

void ShowZoomMessage()

In the case where the 3rd party application is not able to recreate the view during a zoom to (see ZoomTo3D method) the error message icon can be made visible on the user control using this helper function.

Clicking on the message icon will result in the message being shown defined using the AddZoomMessage helper function.

string AddIssues(List<Issue> issues)

This helper function will allow the 3rd part application to define a list of issues to be pushed to the user control. See sample code for more information.

**string GetDictionaries(ref Dictionary<uint, Entry> statuses,
ref Dictionary<uint, Entry> types,
ref Dictionary<uint, Entry> priorities,
ref Dictionary<uint, Entry> labels,
ref Dictionary<uint, Entry> assignedTos,
ref Dictionary<uint, Entry> areas,
ref Dictionary<uint, Entry> milestone,
ref Dictionary<uint, Entry> visibilities,
ref Dictionary<uint, Entry> customselects1,
ref Dictionary<uint, Entry> customselects2)**

This helper function will allow the 3rd party application to retrieve the values currently used by the BIMcollab project it is connected to.

These values should be used when creating a list of issues to be pushed to the user control.

See sample code for more information.

```
string GetMandatoryFields(ref bool bArea,  
                          ref bool bAssignedTo,  
                          ref bool bDeadLine,  
                          ref bool bDescription,  
                          ref bool bLabels,  
                          ref bool bMileStone,  
                          ref bool bPriority,  
                          ref bool bType,  
                          ref bool bCustomSelect1,  
                          ref bool bCustomSelect2,  
                          ref bool bCustomField1,  
                          ref bool bCustomField2)
```

This helper function will allow the 3rd party application to identify which fields are set as mandatory in the BIMcollab project it is connected to.

These values should be considered when creating a list of issues to be pushed to the user control.

```
string GetEnabledFields(ref bool bArea,  
                        ref bool bAssignedTo,  
                        ref bool bDeadLine,  
                        ref bool bDescription,  
                        ref bool bLabels,  
                        ref bool bMileStone,  
                        ref bool bPriority,  
                        ref bool bType,  
                        ref bool bCustomSelect1,  
                        ref bool bCustomSelect2,  
                        ref bool bCustomField1,  
                        ref bool bCustomField2)
```

This helper function will allow the 3rd party application to identify which fields are available in the BIMcollab project it is connected to. The availability of fields can vary depending on the subscription level of the BIMcollab Nexus.

These values can be considered when creating a list of issues to be pushed to the user control (e.g. disable dialog controls).

```
public static string  
GetLanguageFilesPath()
```

This method shall return the full path to the folder containing the languages sub-folder including the final backslash.

Returning an empty string will result in the custom control looking for the language files in the current working directory.

public static string

GetApplicationName()

This method shall return a string contains the name of the application using the custom control.

The name will be used for various error/message dialogs.

public static IntPtr

GetApplicationMainWindowHandle()

This method shall return the handle value (HWND) of the application's main window.

The sample application shows a possible implementation.

public static string

GetToken()

This method shall return the application specific token needed to connect to a BIMcollab space in the form of a string.

This token will have been provided to you by KUBUS.

public static string

GetRegistryValuePrefix()

This method shall return a prefix which will be used by the user control to store application specific settings in the registry.

The prefix used must be unique and will be provided to you by KUBUS.

public static bool

CreateSnapshot(String snapshotPath)

This method shall create a snapshot of the current view in the 3rd party application and store it in the folder specified by snapshotPath.

The snapshot must be stored in .png format and be named "snapshot.png".

Return 'true' when the method succeeds.

public static bool

OkToSetView(ref String message, ref Boolean is2D, Boolean resetView = false)

This method shall return whether it is allowed to receive viewpoint information from 3rd party application's current view.

The parameter message is deprecated and is no longer used by the user control.

Parameter is2D is used to indicate to the user control whether the current view is a 2D view (true) or a 3D view (false).

Parameter resetView is used to indicate if the user is updating an existing viewpoint (true) or a new viewpoint is being created (false).

public static void

SetView2D(ViewPoint viewpoint, Settings settings)

This method will allow the 3rd party application to provide 2D viewpoint information to the user control based on the current view.

The parameter viewpoint will contain the viewpoint information.

The parameter settings provide information about the currently selected settings in the user control. See 'Settings' for more detailed information.

See 'Viewpoint Usage' for details about working with viewpoints.

public static bool

SetView3D(ViewPoint viewpoint, Settings settings)

This method will allow the 3rd party application to provide 3D viewpoint information to the user control based on the current view.

The parameter settings provide information about the currently selected settings in the user control. See 'Settings' for more detailed information.

Return 'true' when the method succeeds.

See 'Viewpoint Usage' for details about working with viewpoints.

public static void

ZoomTo2D(ViewPoint viewpoint, Settings settings)

This method will allow the 3rd party application to recreate a 2D view based on the provided viewpoint information.

The parameter settings provide information about the currently selected settings in the user control. See 'Settings' for more detailed information.

Return 'true' when the method succeeds.

See 'Viewpoint Usage' for details about working with viewpoints.

public static void

ZoomTo3D(ViewPoint viewpoint, Settings settings, bool fullZoom, bool controlKeyPressed = false)

This method will allow the 3rd party application to recreate a 2D view based on the provided viewpoint information.

The parameter settings provide information about the currently selected settings in the user control. See 'Settings' for more detailed information.

The parameter fullZoom, relates to the current status of the QuickZoom button in the user control.

- When true the 3D view should be recreated taking all viewpoint attributes into account.
- When false the 3D view should be recreated by changing the camera position only.

The parameter controlKeyPressed is set to true when the control key was pressed when the action to trigger the ZoomTo3D was performed. The 3rd party application can implement an alternative zoom action based on this parameter (e.g. change the projection mode).

Return 'true' when the method succeeds.

See 'Viewpoint Usage' for details about working with viewpoints.

public static bool

CreateComponentCache(ViewPoint viewpoint)

This method shall implement adding the viewpoints components to the component data cache. This cache can be used to prevent superfluous retrieval of component data from the 3rd party application. How and which data to cache is up to the developer of the AppInterface.

Can be left empty when caching is not needed.

public static Boolean

SelectComponent(String ifcGuid)

This method will allow the 3rd party application to select the component identified by the provided IFC GUID.

This will be called when a component is selected in the 'viewpoint info' dialog.

public static String

GetComponentIfcType(String ifcGuid)

This method will allow the 3rd party application to provide the user control with the IFC type of the component identified by the provided IFC GUID.

This will be called when the 'check components' button is clicked in the 'viewpoint info' dialog.

public static void

SetBoundingBox()

This method will allow the 3rd party application to create a bounding box around the visible components in the current 3D view.

This will be called when the 'create section box' button is clicked in the user control.

public static void

ImportIssues()

This method will allow the 3rd party application to create a list of issues to be imported from for instance a clash check.

This will be called when the 'Import Issues' button is clicked in the user control.

See sample code for more information.

Settings

The settings object received will indicate which options are selected in the user control when creating or recreating a view.

bool showComponentColor

This value indicates if the 'use override colors' button is enabled. When true then the 3rd party application should use any color information in the provided viewpoint when recreating the view.

uint maxNumberOfComponents

This value indicates the maximum number of components allowed in a viewpoint. If more than the maximum is added, then the component list will be ignored while creating the issue.

bool linkVisibleObjects

This value indicates if 'visible' is selected in the 'components in viewpoint' pulldown in the 'add Issue' dialog.

When true then all visible components in the current view in the 3rd party application should be added to the viewpoint.

bool linkSelectedObjects

This value indicates if 'selected' is selected in the 'components in viewpoint' pulldown in the 'add Issue' dialog.

When true then only the selected components in the current view in the 3rd party application should be added to the viewpoint.

bool linkNoObjects

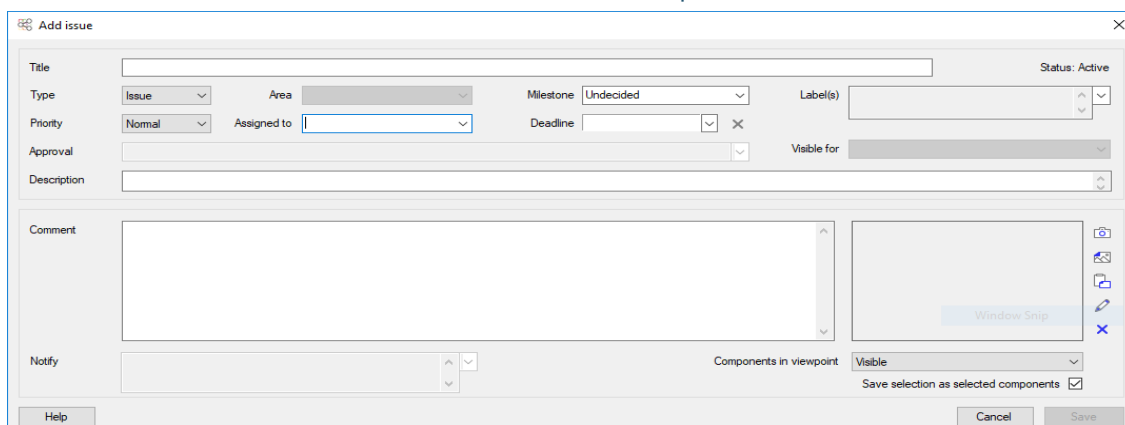
This value indicates if 'none' is selected in the 'components in viewpoint' pulldown in the 'add Issue' dialog.

When true then no components should be added to the viewpoint.

bool saveSelected

This value indicates the 'Save selection as selected components' checkbox is checked in the 'add Issues' dialog.

When true then any components being added to the viewpoint which are selected in the 3rd party application's current view should also be marked as selected in the viewpoint.



Viewpoint usage

The ViewPoint class is an empty container for cameras, settings, components, and clipping planes.

Setup ViewPoint

To create a ViewPoint a camera must be created.

For 3D viewpoints, the CameraPerspective and the CameraOrthogonal can be used.

For 2D viewpoints, the CameraDrawing is available.

Create one of those cameras and store it in their respective viewpoint class members.

3D cameras use 3 parameters to define them: location, direction and upvector. All dimensions are in meters.

- 'location' is a Point3D defining the camera position
- 'direction' is a Vector3D defining the direction the camera looks into
- 'upVector' is a Vector3D that defines the rotation of the camera around its looking direction
- 'fieldOfView' (perspective) relates to the camera's view angle
- 'viewToWorldScale' (orthogonal) defines the relation between the actual area in view and the camera's image

A 2D camera has 4 doubles that define the 2d area the camera sees.

- viewLeftTop_X
- viewLeftTop_Y
- viewRightBottom_X
- viewRightBottom_Y
- viewName is used to give the view a name
- viewPortClient is used to identify the application used to create the CameraDrawing.

Once the camera is defined, other elements such as components and clipping planes can be added to their respective collections.

Components

Components are defined by an ifc guid and some viewing properties:

- color (color overlay)
- selected (show the component in selected state)
- visibility (component should be shown (true) or hidden (false)).

Depending on the visibility of the components in the set, it shall be used to hide the components or hide the rest of the model(s).

All components with a color or that are selected shall not be considered for the visibility calculation, they are visible by default.

If all the other components are `visibility == false` the components shall be hidden from the model.

When there is at least one component with `visibility == true`, all models must be hidden and all components with `visibility == true` shall be shown.

After the setup of the view, the components with colors or selection must be shown in a matching state.

Creating a component set

The `Settings` class in the `ViewPoint` contains a number of parameters that must be taken into account when using a component collection:

`Settings.showComponentColor` is set to `false` when the user has chosen to `ZoomTo` without using the color option.

`Settings.maxNumberOfComponents` is set to the max number of components allowed in the component set.

`Settings.linkVisibleObjects` is set to `true` when the component collection must be created from only the visible components.

`Settings.linkSelectedObjects` is set to `true` when the component collection must be created from the selected components.

`Settings.saveSelected` is set to `true` when the selected state must be saved with the selected components.

ClippingPlanes

Clipping planes are specified by a location (`Point3D`) and a direction (`Vector3D`). The location pins the clipping plane to a location in the model, the direction defines the normal of the clipping plane. The side of the clipping plane the direction points into the area of the model that should be clipped away.

Modeless Dialogs and Multi-Threading

When using the custom control in a modeless dialog it is important to note that the apartment state of the thread running that dialog is set to 'single-threaded apartment'. Not doing so will result in the control not being able to make use of standard windows resources such as Open, Save and Select dialogs.

The following code snippet demonstrates the correct way to initialize the thread:

```
private void button1_Click(object sender, EventArgs e)
{
    Thread modelessThread = new Thread(StartBCFManager);
    modelessThread.IsBackground = true;
    modelessThread.SetApartmentState(ApartmentState.STA);
    modelessThread.Start(this);
}

private static void StartBCFManager(object here)
{
    Form2 form2 = new Form2();
    form2.ShowDialog();
}
```