**Steel Connections .NET API**

**Developer Training Guide**

**Contents**

[Chapter 1 Steel Connections joints technology and philosophy 5](#_Toc114654073)

[1.1 Introduction to the Steel Connections API 6](#_Toc114654074)

[1.2 Requirements 6](#_Toc114654075)

[1.3 Installation 6](#_Toc114654076)

[1.4 .NET vs COM Technology for Joints 6](#_Toc114654077)

[1.5 Joint Rules, Behaviors, and Best Practices 7](#_Toc114654078)

[Chapter 2 Creating .NET Joints 9](#_Toc114654079)

[2.1 Creating a .NET API joint 10](#_Toc114654080)

[2.2 Making the joint available in Revit 11](#_Toc114654081)

[2.3 Joint input objects 13](#_Toc114654082)

[2.4 Default attribute state for objects controlled by connections 13](#_Toc114654083)

[2.5 Steel Connections with Ranges of Applicability 14](#_Toc114654084)

[Chapter 3 Configure AstorRules.mdf for Filtering Steel Connections in Revit 15](#_Toc114654085)

[3.1 Database tables 16](#_Toc114654086)

[3.2 Quick Configuration 17](#_Toc114654087)

[3.3 Detailed Configuration 19](#_Toc114654088)

[3.3.1 AutoConnectionProfiles 19](#_Toc114654089)

[3.3.2 AutoConnectionCategories 19](#_Toc114654090)

[3.3.3 AutoConnectionInputSet 21](#_Toc114654091)

[3.3.4 AutoConnectionInputObjectCondSet 22](#_Toc114654092)

[3.3.5 AutoConnectionObjectsOrderForJoints 24](#_Toc114654093)

# Chapter 1 Steel Connections joints technology and philosophy

## Introduction to the Steel Connections API

The Steel Connections API allows you to program with any .NET compliant language like Visual Basic.NET or C#. Before using the API, learn to use Steel Connections features so that you can better understand this API.

## Requirements

* Understanding Steel Connections
* Programming language like C#
* Microsoft Visual Studio 2022 or Visual Studio Community Edition 2022 (if applicable) with the “.NET Desktop development” workload
* Revit 2025

## Installation

The Steel Connections API is installed with Revit. Microsoft Visual Studio Community Edition 2022 can be downloaded from <https://visualstudio.microsoft.com/downloads/>

Use the information in this document to help you build your own steel joints and discover some  
more nonobvious joint rules and behaviors that you should consider when you design a new joint.

## .NET vs COM Technology for Joints

|  |  |  |
| --- | --- | --- |
|  | .NET | COM |
| Solution Maturity | .NET is the golden standard for creating Addons for Revit. As such, Steel Connections has much of its functionality exposed to .NET API. | This technology has existed for longer than .NET and covers some parts of Steel Connections functionality that the .NET API might not. |
| UI Flexibility | In this case, the UI was meant to be implemented in a descriptive manner, looking very plain in the end, and missing a few important features, like the ability to show, hide, or replace UI controls, or the ability to make one value depend and change based on another. | With this technology, you have full UI flexibility - you can implement almost anything. However, it is still recommended to maintain some consistency with the existing UI, and there may be controls that may not make any sense - they would be too difficult to populate or to be considered by Steel Connections afterwards. |
| Languages Available for Writing Joints | All .NET-based languages are available; however, Steel Connections has been tested with C# | Any language that can work with COM (C++, C#). |
| Ease of Use or Coding | .NET languages are found to be easier and more natural to work with by most programmers. |  |
| Ability to do things outside the normal joints scope. | Generally, very flexible. However, it is not recommended to stray too far outside the rules of what can be done inside a joint, as you could risk breaking the Steel Connections automatic change handling. Read more in the next section of this document, [Rules, Behaviors, and Best Practices](#_Joint_Rules,_Behaviors,). | Offers access to some functionality that is not present in the .NET API. All required and useful data fields and methods should be covered and available in this API. However, access to some functionalities or properties may, at times, be restricted by the absence of very specific API. |
| Ability to create joints that instantiate other joints. | Limited | Yes |

## Joint Rules, Behaviors, and Best Practices

Connections are smart parts of a building that automatically adapt to changes in the environment (changes in the position of input elements, orientation, size, length, changes in loads placed on elements, or parameters modified by users) in order to:

* ensure the construction is as correct as possible.
* help the engineer avoid mistakes.
* save time by automatically propagating changes throughout the model.

Another term we use in some cases to refer to “Joints” is “Steel Connections”.

A joint has:

* Input elements (sometimes referred to as “drivers”).
* Parameters.
* Output elements (sometimes referred to as “driven”).

Steel Connections for Revit has a mechanism that checks for changes in the model to determine what other elements are impacted by the changes, and to properly sort and propagate changes throughout the model. Joints are an important part of that system, and here are some notes about how that works:

* Input elements are very important. Usually when an input element changes, the joint will update.
* Although the joints can modify their input elements, it is strongly discouraged because such operations are typically not deterministic enough and it can become impossible to achieve model stability (e.g., a joint moving the input beam - every time the joint updates, it moves the beam again and again, and as a result, the joint is not up to date and needs to be ran again).
* Joints can be cascaded: the output element of one joint can be an input element for another joint. So, if the input element of one joint changes, a sequence of joint updates will often occur.  
  Therefore it is important to properly define the input elements of a joint.Using elements that are not declared as input, may result in the joint not being updated when necessary.  
  If you declare objects that are not actually used as input by the joint, the joint might end up being updated more often than necessary, eventually leading to performance issues.
* Usually, joint logic is run only once, the moment you create them. Later, the joints will update automatically without any user input. Similarly, the joints will update automatically when you change their parameters.
* Every time a joint updates, all the created elements will be erased, new ones will be created in their place and the properties will be transferred. Steel Connections will automatically identify new elements created, replacing the old ones, and transfer properties and options from the old to the new.  
  These include user-defined properties that are not controlled by the joint, features created by the user in addition to what the joint does and others. For Steel Connections to be able to properly match the old objects to the new ones, the newly created object set needs to be reasonably similar.  
  Therefore, it is recommended for joints to have limited responsibility and handle a few cases very well, rather than handle a lot of cases poorly. It is not ideal that by changing one joint parameter, you will change the created element set from 3 plates and a weld to 5 beams and 6 bolts – that would be difficult to match, and your changes may be lost.
* Joints have a rule for what to do about the parameters of created elements. When you create a joint, you can specify what to do about these parameters globally, and the rule used to be that all the parameters of created elements are controlled by the joint and the user is not allowed to change them. A while ago, the default behavior was changed to be that all the parameters of created elements are NOT controlled by the joint, and the user is allowed to change them. Now, you have full control over these parameters, the default behavior coming into play only if you do not explicitly set a different rule. This implies that you must either enumerate the parameters that the joint controls so they are blocked for users or enumerate the parameters that are NOT controlled by the joint so that they can be made accessible to users. To clarify, the parameters controlled by the joint are expected to be set by the joint during element creation, and not be transferred from the old objects to the new objects after the joint updates.
* When joints run into an error, it is recommended that they throw an exception. This allows Steel Connections to notify the user through a warning message. The joint will display a warning mark and a connection handler object, pointing out a connection exists, instead of the actual joint element.  
  This way, users can easily see that something is wrong and that the joint needs manual intervention.
* Steel Connections for Revit automatically detects such simple cases and acts on them; for example, if one of the input elements disappears, even if it is a direct user action (e.g., the user deletes an element that is input to one or more joints), the joint or joints will also be deleted.
* There are various scenarios when joints will be run fully automatically from start to end without any user interaction (e.g., when they are copied and pasted or when they are created by adapting an existing joint to a new location in the model. Therefore it is important to not introduce user interaction during the element creation part of the joint or introduce custom or unusual user interaction anywhere (Steel Connections for Revit might be blocked and unable to perform automatic workflows in such cases).
* Joints can be exploded. If you explode a joint, it will lose its smart quality, and it will not adapt to any input element changes. Also, if among the created objects there were other joints, they will become top-level joints after exploding the joint that contained them.

**Note**: Make sure you also read the Key Concepts chapter from the Steel Connections for Revit COM API Reference Guide SDK document. It applies to .NET joints as well.

# Chapter 2 Creating .NET Joints

## Creating a .NET API joint

Conceptually, the process of developing a joint using the .NET API is not different from the one using the COM API. However, there are a few differences. These are the steps involved in creating a new .NET joint:

1. Build a new .NET dll.
2. Reference the following Steel Connections for Revit SDK libraries (these are the most important libraries, but you may not need all of them or you may need others as well):
   1. ***ASObjectsMgd*** - to get access to joint specific classes and interfaces and most Steel Connections objects.
   2. ***ASGeometryMgd*** - to get access to 2D and 3D geometry libraries, points, vectors, matrices, curves, etc.
   3. ***ASCadLinkMgd*** ­­- to get access to ObjectId and a few other CAD related classes and utilities.
   4. ***DotNetRoots*** - to get access to units, databases, etc.
   5. ***ASProfilesMgd*** - Steel profiles database access
   6. ***ASModelerMgd*** - Steel modeler access
3. Create a new class that will represent your new joint.
   1. Make it implement the Autodesk.AdvanceSteel.ConstructionTypes.IRule interface (found in the *ASObjectsMgd.dll* library).
4. Make sure you implement all the IRule methods (they are all called during the lifecycle of a joint in a model).
   1. **JointId** is just something you may need later – you can provide a simple auto-implementation like:



* 1. **Query** 
     1. this is where your joint can select / define its input elements (which at this time can be any Autodesk.AdvanceSteel.CADAccess.FilerObject, usually beams, plates, “UserDefinedPoint”.
     2. after selecting the input elements, the query method should end with something like this:



The joint needs to be told which are the input objects that it will later depend on.

* 1. **GetTableName** is an optional method. It is useful if a table is defined inside the AstorRules database that will store predefined values for your joint. At this time, we are still looking into improving the behavior related to this method / table.
  2. **GetRulePages**
     1. is the place where the .NET joints can describe the parameters that the user can modify to control the joint behavior.
     2. This is an important difference from the COM API based joints. The .NET API joints are no longer free to define any UI – the UI will be built in a descriptive manner which has 3 important effects:
        + To simplify the process for the joint developer (no need to worry about UI, controls interaction, page size, updating joint when user changes something, etc.).
        + To ensure UI consistency – all joints will have a similar look and feel.
        + To somewhat restrict the possibilities related to the UI (we plan to enhance this in the future according to user feedback, but for the first version it will provide a quite simple but functional UI).
  3. **Save / Load**
     1. designed to define the parameters that the joints need to consider as input for its calculations.
     2. the best practice is that the parameters saved here are reflected in the UI. The user can directly change the parameters in the joint UI (the descriptions of the UI used in the “GetRulePages” method will refer to the parameters saved / loaded on those methods).

**CreateObjects** is the method where the joint is supposed to do all the calculations, object creation, modification, etc. having access to the input elements defined in the Query method and the user controllable parameters that are serialized on the Save / Load methods.

## Making the joint available in Revit

To make the user defined connection available in Revit:

* Set the project references to the required Revit installed version (e.g.: %Program Files%\Autodesk\Revit 2025\AddIns\SteelConnections – if the connection should load into Revit 2025).
  + Tip: A “.props” file similar with the StructuralConnectionsSDKSamples.Common.props provided in this documentation can be used for this.
* Compile the project
* Create and add the required configuration file about the new joint.
  + This is a .xml file that should be created under a folder named “ThirdPartySettings”.
  + The “ThirdPartySettingsFolder” can be placed in one of the following locations:
    - in the Steel Connections data path plugin installation, next to the SteelConnectionsData.xml (e.g. %ProgramData%\Autodesk\Revit Steel ConnectionsVersion\lang\ThirdPartySettings\CompanyName\UserCreatedCustomConnection.xml).
    - in the AppData folder for the Steel Connections Add-in (%Appdata%\Autodesk\Revit\Autodesk Revit Version\SteelConnections\ThirdPartySettings\Company\CustomConnection.xml).
  + Multiple .xml files with different names could be added in “ThirdPartySettings” folder
  + Subfolders can be created under “ThirdPartySettings” directory with .xml files in them.
  + If the folder doesn’t exist, it should be created.

**.xml file syntax:**

**Tips:**

* The content structure should be like the SteelConnectionsData.xml from the %ProgramData%\Autodesk\Revit Steel Connections 2025\lang\ path
* There is a sample SteelConnectionsSampleJoints.xml file provided with this documentation

<SteelConnectionsPath> the path to the binary and resource files for the user defined connection

- Note: if this field is missing, Revit will search for the binaries only in the “SteelConnections” folder from the installation location.

<ResourceDll> - the name of the resource dll file that contains the images (Both the image and preview image should be contained in resource .dll files). The resources .dll filenames should be separated with a comma (“,”).

Example:



<TypeId> - contains the guid from the “classId” column from the **AstorRules**.*HRLDefinition* table.

<PreviewText> - contains the tooltip for the connection.

<Images> - the name of the image file to be displayed under various dialogs in Revit, found in the .dll file containing the resource image

<PreviewImages> – the name of the preview image for the connection that should be found in the .dll file containing the resource image

Example:



* Prepare the **AstorRules** database with joint information for the new custom joint.

**Note**: The Steel Connections databases are delivered archived upon installation of Revit. Please make sure to start Revit once and press the Connection Settings button to make sure the databases are unarchived in the installation folder. After this step all MDF databases should be in that folder. There should also be some LDF files created automatically.

1. This process is almost identical to the one from the COM API joints.
2. Open the **AstorRules** database (typically:*C:\ProgramData\Autodesk\Revit Steel Connections 2025\en-US* for English or the currently installed country folder).
3. Open **RulesDllSigned** table and add a new record:
   1. Key – something well above the standard Steel Connections range,  
      e.g. 150000.
   2. FileName – the name of the dll (library) that implements your connection.
      1. Tech – One of the following options:
      2. “0” for connections based on COM API, implemented in C++.
      3. “1” for connections based on COM API, implemented in .NET.
      4. “2” for .NET API connections. (If you follow this tutorial, you should use “2”).
   3. Signature – currently unused
4. Open the **HLRDefinition** table and add a new record here:
   1. Key – something well above the standard Steel Connections range,  
      e.g. 150000.
   2. Rule run name – this is how users will see your connection labeled.
   3. Internal name – a unique, simple name with no spaces that is used to invoke your connection.
   4. Category – Used to group connection families.
   5. Dll – The key used when adding the new record to the **RulesDllSigned** table above.
   6. SubNameInDll – The complete name of the class that implements the IRule interface, including the namespace, i.e. your connection (see step 3).
   7. ClassId – Generate a new, unique, GUID here for your connection – this will be used also later, in a xml configuration file mentioned in the next step. Useful also if you create a connection design module for your connection as well.
5. Update table **AutoFilteringConfig** with information about the new joint.
   1. Key – Use a value well above the standard Steel Connections range.
   2. RunName – User defined name for the connection
   3. Category – Category of the connection, defined in the **AutoConnectionCategories** table.
   4. InputSet – Acceptable input profile type combinations, defined in the **AutoConnectionInputSet** table.
   5. InputSetConds – Conditions for input element set, see **AutoConnectionInputObjectCondSet** table.
   6. RuleInternalName – The InternalName from the **HRLDefinition** table (see step 3).
   7. ObjectsOrderForJoints – Value indicating the order of the input elements for the connection, from the **AutoConnectionObjectsOrderForJoints** table.
   8. OwnerText - User defined value that can be left blank.

Example entry in the **AutoFilteringConfig** table:

|  |  |
| --- | --- |
| *Key* | *999999* |
| *Category* | *SampleJoint* |
| *RunName* | *ColOrRaf Any to ColOrRaf Any* |
| *InputSet* | *Any+Any* |
| *InputSetConds* | *No Condition* |
| *RuleInternalName* | *SampleJoint* |
| *ObjectsOrderForJoints* | *2 Beams inversed* |
| *OwnerText* |  |

**Note**: For more information about configuring the AstorRules.AutoFilteringConfig please check the chapter [Configure AstorRules.mdf for Filtering Steel Connections in Revit](#_Chapter_3_Configure)

* Add info about the new joint in SteelConnectionsData.xml:
* Make sure the TypeId field from the xml has the same GUID as the ClassId field from the **AstorRules.*HRLDefinition*** table for the record pointing the joint.
* Fill the Images and PreviewText with the names of your corresponding resources.

Example:



* Add the resource dll name (the one that contains the image and preview image for the joint) to the *ResourceDll* and *PreviewResourceDll* fields. The resources dll names should be separated with a comma (“,”) from the existing dll names.  
  Example:



* Installation (connection .dll, .NET resources .dll and all the databases and xml files updated for all the languages). Currently the Steel Connections addin is installed with many languages. As such, the databases and SteelConnectionsData.xml for each country must be updated accordingly.

The path to the SteelConnections binaries should be composed from the Revit install location and the relative path to the Revit addins.

The Revit install location can be read from the Registry Keys, e.g.

*HKEY\_LOCAL\_MACHINE\SOFTWARE\Autodesk\Revit\2025\REVIT-05:0409\InstallationLocation*

The result should be something like this:

*“C:\Program Files\Autodesk\Revit 2025\”* + *“AddIns\SteelConnections”*.

The SteelConnections addin stores its data in a dedicated folder specified by the “DataPath” key from the *ASSettings\_Advance.xml* configuration file.

This file is usually found in *C:\ProgramData\Autodesk\Revit Steel Connections 2025*.

The path does not include the language folder that is dynamically appended on startup. Steel Connections data (including databases) is archived in a .zip file and it is automatically unzipped on the first usage of the SteelConnections functionality in Revit.

## Joint input objects

Steel Connections for Revit joints currently work on the Revit structural families (framings, columns).

The structural steel columns and framing elements need to meet a series of requirements to be used in the steel fabrication workflow. See [Supported Structural Steel Shapes and Families for Steel Fabrication](http://help.autodesk.com/view/RVT/2023/ENU/?guid=GUID-6244E741-5D14-4DAD-AE25-5069F71B69F3).

**Note**: There are no special requirements for adding a Steel Connections for Revit joint design module to Revit. The .dll file has to be installed in the Steel Connections folder and the databases for Steel Connections for Revit have to be updated accordingly.

## Default attribute state for objects controlled by connections

Starting with Revit 2021, all attributes of steel objects controlled by connections are accessible by default.

Using the .NET API, setting the Attributes property is performed simply by enumerating them in the Attributes member of the corresponding CreatedObjectInformation structure.

These attributes will become inaccessible from the object GUIs and will not be transferred on connection updates.

Connection developers have the possibility to switch back to the old behaviour where the attributes of controlled objects are not accessible by default. This can be accomplished by setting the IJoint.DefaultAttributeStateAccessible property to the false value. If this is done, developers must then specify the accessible attributes from within the connections, for example, the attributes that the connection does not control, as before.

To preserve the old behaviour (Revit 2020 and earlier) of attributes for existing connections, the *Version* field was introduced in the ***HRLDefinition*** table. Setting this to 24 (corresponding to Revit 2020) or lower for existing connections, will enforce the old behaviour - attributes are not accessible by default, only the specified ones are accessible.

New connections should have the Version field set as the current internal version. This ensures that new attributes, not known prior to a certain version, are correctly excepted from the above rules. The version can be found by looking at the major file version (or product version) from the “Details” tab in the dll properties window of any dll from the currently installed Steel Connections folder(e.g. 29 for Revit 2025).

## Steel Connections with Ranges of Applicability

It is possible to associate steel connections with profile sizes by using ranges of applicability attached to the connection types. This allows for automated creation of structural steel connections relying on rule-based solutions.

An example implementation can be found as part of the Revit SDK sample *SampleCommandsSteelElements*. See the *AddRangesToConnectionType.cs* source code file for details.

For additional information on ranges of applicability in Revit, see [Library-Based Connection Design Automation](https://help.autodesk.com/view/RVT/2023/ENU/?guid=RevitDynamoSteel_Library_Based_Connection_Design_Automation_html).

# Chapter 3 Configure AstorRules.mdf for Filtering Steel Connections in Revit

## Database tables

To add a connection in the Type list, the connection needs to be configured in the AstorRules.mdf database.

Once the input elements are selected, the Type list is filtered based on several criteria:

* Connection zone.
* Element type.
* Section type.
* Geometrical conditions.

The order of applying the criteria is based on the relations between the input elements.

Related Tables:

**AutoFilteringConfig**

1.[***AutoConnectionProfiles***](#_AutoConnectionProfiles)

*2.*[***AutoConnectionCategories***](#_AutoConnectionCategories)

*2.1.*[*AutoConnectionCategoriesItems*](#_AutoConnectionCategoriesItems)

*2.2.*[*AutoConnectionCategoriesItemsDesc*](#_AutoConnectionCategoriesItemsDesc)

*2.2.1.*[*AutoConnectionConnectionZones*](#_AutoConnectionConnectionZones)

*2.2.2.*[*AutoConnectionModelRoles*](#_AutoConnectionModelRoles_1)

*2.2.3.*[*AutoConnectionModelRolesItems*](#_AutoConnectionModelRolesItems_1)

*3.*[***AutoConnectionInputSet***](#_AutoConnectionInputSet)

*3.1.*[*AutoConnectionInputObjects*](#_AutoConnectionInputObjects)

*4.*[***AutoConnectionInputObjectCondSet***](#_AutoConnectionInputObjectCondSet)

*4.1.*[*AutoConnectionInputObjectCondItems*](#_AutoConnectionInputObjectCondItems)

*4.1.1.*[*AutoConnectionGeomConditions*](#_AutoConnectionGeomConditions)

*4.1.1.1.*[*AutoConnectionGeomConditionsBase*](#_AutoConnectionGeomConditionsBase)

*4.1.1.2.*[*AutoConnectionGeomConditionsBaseRelPos*](#_AutoConnectionGeomConditionsBaseRel)

*4.1.1.3.*[*AutoConnectionGeomConditionsBaseMainMemberConn*](#_AutoConnectionGeomConditionsBaseMai)

*5.*[***AutoConnectionObjectsOrderForJoints***](#_AutoConnectionObjectsOrderForJoints)

**Note**: The Steel Connections databases are delivered archived upon installation of Revit. Please make sure to start Revit once and press the Connection Settings button to make sure the databases are unarchived in the installation folder. After this step all MDF databases should be in that folder. There should also be some LDF files created automatically.

## Quick Configuration

The simplest configuration looks like this:

Graphical user interface, application

Description automatically generated

|  |  |
| --- | --- |
| Key | User defined |
| RunName | User defined |
| Category | Defined as described in section [3.3.2.***AutoConnectionCategories***](#_AutoConnectionCategories) |
| Based on input element number, the model role and connection zone of the elements. |

*E.g., The case where you want to define a Category for one element, with model role set for rafter and column and for a connection located at the end of the input element:*

*Graphical user interface, application, Word

Description automatically generated*

*Steps:*

1. *In* ***AutoConnectionModelRolesItems*** *create two entries, one for Column, one for Rafter, same key, different Subkey.*
2. *In* ***AutoConnectionModelRoles*** *create a new entry and for the Items value add the key previously defined in* ***AutoConnectionModelRolesItems****.*
3. *In* ***AutoConnectionCategoriesItemsDesc*** *create a new entry with:*

* *set ConnectionZone to End and Middle Zone.*
* *set ModelRoleSet to the previously defined entry from AutoConnectionModelRoles.*

1. *In* ***AutoConnectionCategoriesItems*** *create a new entry with the description set to the previously defined entry key from* ***AutoConnectionCategoriesItemsDesc****.*
2. *In* ***AutoConnectionProfiles*** *set a new profile and set the value for UsedForFiltering to 1.*
3. *In* ***AutoConnectionCategories*** *add a new entry with:*

* *Items value set to the previously defined entry key from* ***AutoConnectionCategoriesItems****.*
* *Profile set to the previously defined entry from* ***AutoConnectionProfiles****.*

1. *Added in* ***AutoFilteringConfig*** *in Category field for the defined case.*

* InputSet:
  + defined as described in section [3.3.3 ***AutoConnectionInputSet***](#_AutoConnectionInputSet)***.***
  + associates criteria based on the section type of the input elements.

*E.g., The simplest case, where you want to select all the metal sections, for the case with only one element, is to define an InputSet with one element and with an InputObjects which will contain all steel sections.*

*Graphical user interface, application

Description automatically generated*

*Steps:*

*1. In the* ***AutoConnectionInputObjects*** *table define an entry with:*

* *ObjectType - 203 – hard-coded value corresponding to steel elements.*
* *ObjectSpecific - I,U,T,W,O,H,F,D,Q .*

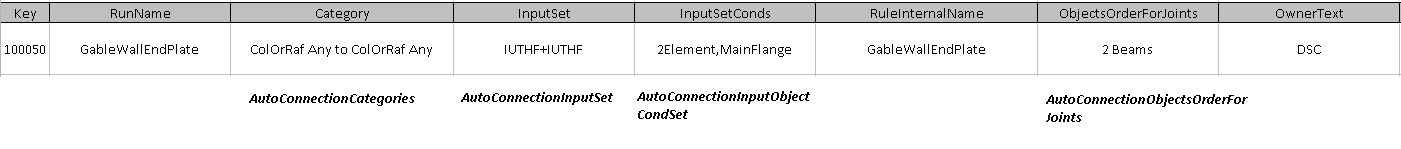
*2. In* ***AutoConnectionInputSet*** *table define an entry with:*

* *Items – 1 – one element.*
* *Items1 – previously defined ObjectSpecific from AutoConnectionInputObjects.*

*3. Added in* ***AutoFilteringConfig*** *in InputSet field for the defined case.*

* InputSetConds:
  + defined as described in section [3.3.4 ***AutoConnectionInputObjectCondSet***](#_AutoConnectionInputObjectCondSet)
  + define a geometrical relationship between input elements.
  + can be set to the predefined type “No Condition” so it does not take into account any geometric limitation.
* RuleInternalName – defined in HRLDefinition table, InternalName.
* ObjectsOrderForJoints – as described in [3.3.5. ***AutoConnectionObjectsOrderForJoints***](#_AutoConnectionObjectsOrderForJoints)*e.g., For one element you can choose “1 Beam”, for two “2 Beams” and so on.*

## Detailed Configuration

**AutoFilteringConfig**

|  |  |
| --- | --- |
| **Key** | User defined |
| RunName | User defined |
| Category | defined by ***AutoConnectionCategories*** |
| InputSet | defined by ***AutoConnectionInputSet*** |
| InputSetConds | defined by ***AutoConnectionInputObjectCondSet*** |
| RunInternalName | defined in HRLDefinition table, InternalName |
| ObjectsOrderForJoints | defined by ***AutoConnectionObjectsOrderForJoints*** |
| Translation |  |

### AutoConnectionProfiles

Used to define a group of rules/connections. You can use it to group all your connections.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Key | RunName | UsedForFiltering | Owner | Deutsch | English |
| 99000 | Connections | 1 | DSC |  |  |

Key - user defined

RunName - user defined

UsedForFiltering = 0 – if the category will not be used in Revit Type list;

= 1 – if the category will not be in Revit Type list.

Owner

Translation

### AutoConnectionCategories

Make the connection between rules and the corresponding profile.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | RunName | Items | Profile | Translation |
| 99011 | ColOrRaf Any to ColOrRaf Any | 99011 | Connections |  |

Key - user defined

RunName - user defined

Items - key used in ***AutoConnectionCategoriesItems*** table

Profile - RunName from ***AutoConnectionProfiles*** table

Translation - Translation language (e.g.: Deutsch, English)

#### AutoConnectionCategoriesItems

Used to group the rules.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | SubKey | RunName | Description | Translation |
| 99011 | 1 | ColOrRaf Any to ColOrRaf Any | 99111 |  |

Key - user defined

Subkey - user defined

RunName - user defined

Description - key used in ***AutoConnectionCategoriesItemsDesc*** table

Translation - Translation language (e.g.: Deutsch, English)

#### AutoConnectionCategoriesItemsDesc

Used to define the rules for each input element (e.g., for 2 input elements will be needed 2 rules).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | SubKey | RunName | ConnectionZone | ModelRolesSet |
| 99111 | 1 | Column or Rafter Any | End and Middle Zone | Column or Rafter |
| 99111 | 2 | Column or Rafter Any | End and Middle Zone | Column or Rafter |

Key - user defined

SubKey - user defined

RunName - user defined

Connection Zone - RunName from ***AutoConnectionConnectionZones***

ModelRoleSet - RunName from ***AutoConnectionModelRoles***

Translation - Translation language (e.g.: Deutsch, English)

##### AutoConnectionConnectionZones

Define the three possible connection zones on the element:

|  |  |  |  |
| --- | --- | --- | --- |
| Key | RunName | OwnerText | Translation |
| 1 | End Zone | DSC |  |
| 2 | Middle Zone | DSC |  |
| 3 | End and Middle Zone | DSC |  |

##### AutoConnectionModelRoles

Group the elements according to their type:

|  |  |  |  |
| --- | --- | --- | --- |
| Key | RunName | Items | Translation |
| 99959 | Column or Rafter | 99959 |  |

Key - user defined

RunName - user defined

Description - key used in AutoConnectionModelRolesItems table

Translation - Translation language (e.g.: Deutsch, English)

##### AutoConnectionModelRolesItems

Used to define the type of each input element.

|  |  |  |
| --- | --- | --- |
| Key | SubKey | ModelRole |
| 99959 | 1 | Column |
| 99959 | 2 | Rafter |

Key - user defined

Subkey - user defined

ModelRole - just Column, Rafter or Foundation model roles can be

used for SteelConnection project.

### AutoConnectionInputSet

Used to define the section type criteria.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Key | RunName | OwnerText | Items | Item1 | Item2 | Item3 | … | Item7 | Translations |
| 89506 | IUTHF+IUTHF | DSC | 2 | IUTWF | IUTWF |  |  |  |  |

Key - user defined

RunName - user defined

OwnerText - user defined

Items - number of the input elements

Items”i” (i=1…7) - RunName from ***AutoConnectionInputObjects***

Translation - Translation language (e.g.: Deutsch, English)

#### AutoConnectionInputObjects

Used to define group of elements based on section type.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | RunName | ObjectType | ObjectSpecific | Translations |
| 99905 | IUTWF | 203 | I,U,T,W,F |  |

Key - user defined

RunName - user defined

ObjectType

|  |  |
| --- | --- |
| Element | Code |
| STEEL BEAM | 203 |
| CONCRETEBEAM | 220 |
| FOOTINGCONTINUOUS | 222 |
| CONCRETECOLUMN | 224 |
| SLAB | 336 |
| FOOTINGISOLATED | 337 |
| WALL | 335 |

ObjectSpecific - defined by the string of the elements, separated

by commas.

|  |  |
| --- | --- |
| Element | Section Type |
| STEEL BEAM | I |
| STEEL BEAM | U |
| STEEL BEAM | T |
| STEEL BEAM | W |
| STEEL BEAM | O |
| STEEL BEAM | H |
| STEEL BEAM | F |
| STEEL BEAM | D |
| STEEL BEAM | Q |
| STEEL BEAM | C |
| STEEL BEAM | T |
| STEEL BEAM | Z |
| STEEL OR CONCRETE ELEMENT | Any |

Translation - Translation language (e.g.: Deutsch, English)

### AutoConnectionInputObjectCondSet

Used to define geometrical relationship between input elements.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | RunName | OwnerText | Items | Deutsch |
| 95007 | 2Element,MainFlange | DSC | 95007 |  |

Key - user defined

RunName - user defined

OwnerText - user defined

Items - key from ***AutoConnectionInputObjectCondItems*** table

Translation - Translation language (e.g.: Deutsch, English)

#### AutoConnectionInputObjectCondItems

Defines geometrical condition sets.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Key | SubKey | ObjectSubKey1 | ObjectSubKey2 | ObjectSubKey3 | Condition |
| 95007 | 1 | 2 | 1 | 0 | MainFlange |

Key - user defined

SubKey - user defined

*Case a. There are only two input elements*

ObjectSubKey1 - the element which relates

ObjectSubKey2 - the element to which ObjectSubKey1 relates

*Case b. There are three input elements*

ObjectSubKey1 - the first element which relates with ObjectSubKey3

ObjectSubKey2 - the second element which relates with ObjectSubKey3

ObjectSubKey3 - the element to which ObjectSubKey1 and ObjectSubKey1

Relates

Condition - RunName from ***AutoConnectionGeomConditions*** table

##### AutoConnectionGeomConditions

Used to define a condition.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RunName | CondType | RelativePosition | MainMemberConnection | StartAngle | EndAngle | OwnerText | Translation |
| MainFlange | MainMemberConnection | - | Flange | 0 | 0 | DSC |  |

Key - user defined

RunName - user defined

CondType - RunName from ***AutoConnectionGeomConditionsBase*** table

RelativePosition - RunName from ***AutoConnectionGeomConditionsBaseRelPos*** table

MainMemberConnection - RunName from ***AutoConnectionGeomConditionsBaseMainMemberConn***

table

###### AutoConnectionGeomConditionsBase

Geometrical condition – hard-coded, possible values are:

|  |  |  |  |
| --- | --- | --- | --- |
| Key | RunName | OwnerText | Translation |
| 0 | No Condition | DSC |  |
| 1 | Perpendicular | DSC |  |
| 2 | Liniar | DSC |  |
| 3 | Angle in plane | DSC |  |
| 4 | RelativePosition | DSC |  |
| 5 | MainMemberConnection | DSC |  |
| 6 | RotationAngle | DSC |  |
| 7 | X Perpendicular on Y | DSC |  |
| 8 | X Perpendicular on Z | DSC |  |
| 9 | X Perpendicular on xOz | DSC |  |
| 10 | X Perpendicular on xOy | DSC |  |
| 11 | VerticalLowestEnd | DSC |  |
| 12 | VerticalHighestEnd | DSC |  |
| 13 | Not Parallel | DSC |  |
| 20 | Same side of main | DSC |  |
| 21 | Opposite side of main | DSC |  |

Key - hard-coded

RunName - hard-coded, possible values are:

No Condition - *used for relation between 2 elements - no conditions*

Perpendicular - *[X1 Ⱶ X2 and Y2 Ⱶ Z1 and Z2 Ⱶ Y1 ] or [X1 Ⱶ X2 and Y2 Ⱶ Y1 and Z2 Z1]*

Liniar - *used for relation between 2 elements - X1 II X2*

Angle in plane - *used for relation between 2 elements - the angle between X1 and X2*

RelativePosition - *used for relation between 2 elements - require a secondary rule from AutoConnectionGeomConditionsBaseRelPos* *table*

MainMemberConnection - *used for relation between 2 elements - require a secondary rule from AutoConnectionGeomConditionsBaseMainMemberConn table*

RotationAngle - *used for relation between 2 elements - the angle of the first beam's web to second's beam direction, projected on the first beam yOz plane*

X Perpendicular on Y - *used for relation between 2 elements - X1 Ⱶ Y1*

X Perpendicular on Z - *used for relation between 2 elements - X1 Ⱶ Z2*

X Perpendicular on xOz - *used for relation between 2 elements - X1 Ⱶ (X2oZ2)*

X Perpendicular on xOy - *used for relation between 2 elements - X1 Ⱶ (X2oY2)*

VerticalLowestEnd - *used for 1 element – Column – rules - bottom*

VerticalHighestEnd - *used for 1 element – Column – rules - top*

Not Parallel - *used for relation between 2 elements - X1 ~~II~~ X2*

Same side of main - *used for relation between 3 elements - Element 2and 3 are on the same side, in relation element 1*

Opposite side of main - *used for relation between 3 elements - Element 2and 3 are on the opposite side, in relation to element 1*

OwnerText - user defined

Translation - Translation language (e.g.: Deutsch, English)

###### AutoConnectionGeomConditionsBaseRelPos

Rules defined based on the relative position of the input elements.

|  |  |  |  |
| --- | --- | --- | --- |
| Key | RunName | OwnerText | Translation |
| 0 | - | DSC |  |
| 1 | Any | DSC |  |
| 2 | Aligned | DSC |  |
| 3 | Inversed | DSC |  |

Key - hard-coded

RunName - hard-coded, possible values are:

Any

- - *none*

Aligned - *Y1 II Y2 or Z1 II Z2*

Inversed - *Y1 II Z2 or Y2 II Z1*

OwnerText - user defined

Translation - Translation language (e.g.: Deutsch, English)

###### AutoConnectionGeomConditionsBaseMainMemberConn

|  |  |  |  |
| --- | --- | --- | --- |
| Key | RunName | OwnerText | Translation (e.g.: Deutsch, English) |
| 0 | - | DSC |  |
| 1 | Any | DSC |  |
| 2 | Web | DSC |  |
| 3 | Flange | DSC |  |

Key - hard-coded

RunName - hard-coded

OwnerText - user defined

Translation

### AutoConnectionObjectsOrderForJoints

Set the order of the input element.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | RunName | ObjNr | Obj1 | Obj2 | Obj3 | Obj4 |
| 0 | 2 Beams | 2 | 0 | 1 | 0 | 0 |
| 1 | 2 Beams inversed | 2 | 1 | 0 | 0 | 0 |
| 2 | 3 Beams [ 0, 1, 2] | 3 | 0 | 1 | 2 | 0 |
| 3 | 3 Beams [ 2, 1, 0] | 3 | 2 | 1 | 0 | 0 |
| 4 | 3 Beams [ 2, 0, 1] | 3 | 2 | 0 | 1 | 0 |
| 5 | 1 Beam | 1 | 0 | 0 | 0 | 0 |
| 6 | 4 Beams [ 0, 1, 2,3] | 4 | 0 | 1 | 2 | 3 |

Key - user defined

RunName - user defined

ObjNr. - number of input elements

Obj”i” (i=1…4) - set the order of the elements.

**Note**: In the above description, the entries in the tables correspond to a real case. Copyig them to AstorRules database will obtain the case when GableWallEndPlate apply for the intersection of two elements, no matter the zone of relative intersection (End and Middle Zone – 2.2), no matter if they are beams or columns (Column or Rafter Any – 2.2), input elements must be sections in Class I, U, T, H or F (IUTHF+IUTHF -3) and the second element being on the main element flange (MainFlange – 4.1).

