Cospec Innolab Co., Ltd.

Ruleset Authoring Guide

A Python-Based Ruleset Authoring Guide for BIM Quality Review Automation

Table of Contents

1. Overview

The success of a BIM (Building Information Modeling) project depends on the accuracy and consistency of the model data. The review process required to ensure the quality of this data is essential, but performing it manually consumes significant time and effort and carries the risk of human error. This document is a technical guide that walks through the development of the Ruleset, which plays a central role in automating the quality review of BIM models.

A ruleset is a collection of rules containing specific verification criteria and logic, through which repetitive, standardized review tasks can be automated. This guide explains, step by step, how a developer can efficiently develop and apply a powerful, flexible custom ruleset using Python scripts.

A ruleset file is fundamentally structured as XML (eXtensible Markup Language) with a .ccl extension. This document comprehensively covers the XML file structure that forms the foundation of the ruleset, how to write the Python scripts that implement the actual verification logic, how to use the core API functions for accessing BIM data, and finally, authoring examples that demonstrate real-world application.

By following this guide, developers will be able to programmatically access BIM data, implement complex verification logic directly, and build a custom quality review system tailored to a project's requirements.

2. Ruleset XML File Structure Analysis

A ruleset file is built around its XML structure. XML provides the framework that defines the hierarchy, settings, and execution logic of each review item. In other words, it serves as the container for all verification rules, providing the skeleton that determines how the ruleset behaves.

The top-level root element of a ruleset is <CcCodeDataSet>, under which sit <CcSetting>, which handles the ruleset's overall settings, and <CcCode>, which defines individual review items.

```
<CcCodeDataSet>
├── <CcSetting>
│     ├── <Editable>
│     ├── <Id>
│     └── ...
└── <CcCode>  (Occurs multiple times)
      ├── <Id>
      ├── <ParentId>
      ├── <IsGroup>
      └── ...
```

2.1. <CcSetting>: Overall Ruleset Settings

The <CcSetting> element defines how the ruleset behaves overall and its metadata. Through this setting, you configure whether the ruleset can be edited, UI display options, report cover information, and more.

Element	Data Type	Key Function	Usage Strategy
Editable	xs:boolean	Controls modification permission: if true, the user can modify the ruleset's logic	When distributing a ruleset that must not be modified — such as a corporate or regulatory standard — set this to false to ensure compliance
ShowTypology	xs:boolean	Displays the building-type selection menu: if true, a UI for selecting the building type is shown	Useful when different review criteria must be applied depending on building use (e.g. healthcare facilities, multi-family housing)
InfoUrl	xs:anyType	Links a help URL: specifies the URL of an external help page related to the ruleset (e.g. the original regulatory text)	Provides a link to the relevant regulation or in-house guideline so users can easily verify the basis for the review criteria
Id	xs:string	Ruleset unique identifier: defines a unique ID that distinguishes the ruleset	For systematic ruleset management, it is recommended to use a naming convention that includes a project code or version information
Title	xs:string	Name shown in the ruleset list: defines the name displayed in the quality-review program's ruleset list	Use a clear, descriptive name so users can intuitively understand the ruleset's purpose
RulesetName	xs:string	Report cover name: defines the cover name of the report generated after the quality review is complete	Used to specify the official title of the report (e.g. '2024 First-Half Architectural Design Quality Review Report')
RulesetUpdateDate	xs:anyType	Ruleset last-modified date: records the last modification date for version management	Helps track the ruleset's change history and confirm whether the user is on the latest version
ShowSearch	xs:boolean	Whether to show a search box: if true, a search box is displayed above the list of review items	Enable for complex rulesets with many review items, so users can quickly find a specific item

Below is a concrete example of the <CcSetting> element.

```
<CcSetting>
  <Editable>true</Editable>
  <ShowTypology>false</ShowTypology>
  <InfoUrl>http://www.inno-lab.co.kr/KBimAssess/RegulationCode.htm</InfoUrl>
  <Id>TEST</Id>
  <Title>Quality Review List</Title>
  <RulesetName>Quality Review Report</RulesetName>
  <RulesetUpdateDate>2022-11-23</RulesetUpdateDate>
  <ShowSearch>false</ShowSearch>
</CcSetting>
```

2.2. <CcCode>: Definition of Individual Review Items

The <CcCode> element defines the individual items where the actual quality review takes place. Each element has a unique ID and forms a hierarchical structure via ParentId.

Element	Data Type	Key Function	Usage Strategy
Id	xs:unsignedShort	Review item unique ID: a unique numeric ID identifying each review item	Assign IDs systematically so they do not overlap anywhere within the ruleset
ParentId	xs:byte	Parent item ID: specifies the Id of the parent item to form the hierarchy. The top-level item is set to -1	Groups items into logical structures — such as chapter/section/clause of a regulation — to improve readability and usability for the user
Number	xs:string	Review item title: the name displayed in the review list	Write it specifically, e.g. '1.1. Wall Thickness Check', so the content being inspected is clear
IsGroup	xs:boolean	Whether the item is a group: if true, it acts as a group (table-of-contents) that can hold child items; no script is included.	Used to create a logical container that holds child verification items. The item itself performs no verification
Desc	xs:string	Item description: a detailed description of the review item's purpose, criteria, and method	Specify the regulatory clause or design standard underlying the review to increase the credibility of the results
Script	xs:string	Verification script: the Python script code containing the actual verification logic	Included only in execution items where <IsGroup> is false; the actual BIM data verification logic resides here

Below is a comparison of a group item and an actual verification item, both expressed as <CcCode>.

Example Group Item (acts as a table of contents)

IsGroup is set to true, and the element acts as a container that holds child items. It does not include a Desc or Script element.

```
<CcCode>
  <Id>1</Id>
  <ParentId>-1</ParentId>
  <Number>Pre-check on BIM Model</Number>
  <IsGroup>true</IsGroup>
</CcCode>
```

Example Validation Item (Contains Validation Logic)

This item is designated as a child of the group item above via ParentId, and includes the actual verification logic through Desc and Script. IsGroup defaults to false, so it can be omitted.

```
<CcCode>
  <Id>1003</Id>
  <ParentId>1</ParentId>
  <Number>Presence of Space Objects</Number>
  <Desc>Every area must have a space object modeled.</Desc>
  <Script>
    <![CDATA[def Check():
    # ... verification logic ...
```

```
    ]]>
  </Script>
</CcCode>
```

In this way, the XML structure defines the 'what' (Number, Desc) and the 'where' (hierarchical structure) of the verification rule. Next, let's look at how to write the Python script that determines 'how' this rule is executed.

3. Writing Python Scripts

The heart of a ruleset is the Python script written inside the <Script> tag. This script interacts directly with the BIM model data and performs the actual task of evaluating the model against the defined verification criteria and returning the results.

The basic requirements that must be followed when writing a script are as follows.

- Use Python 3.4 syntax.
- All logic must be written inside the def Check(): function.
- Access to BIM data must use the provided API — for example, the SELECT() function.
- CDATA must be used to safely embed the code inside the XML.

The Python script must be embedded safely within the XML document. To do this, wrap the Python code in a <![CDATA[...]]> section inside the <Script> tag. A CDATA (Character Data) section instructs the XML parser to treat the enclosed text as raw, literal data rather than parsing it. This prevents parsing errors even when the code contains characters that carry special meaning in XML, such as <, >, and &.

For example, writing code such as the following without CDATA,

```
<Script>
def Check():
  # ...
  if door.SELECT('width').NUMBER() < 0.9:  # XML parsing error occurs!
    door.ERROR('The door's clear width is less than 0.9 m.')
</Script>
```

causes the XML parser to mistake the < character for the start of a new tag, resulting in a structural error. Using CDATA resolves this problem completely.

Below is a correct example script that checks for the 'presence of space objects'.

```
<Script>
  <![CDATA[def Check():
  # Use the SELECT('storey') API to iterate over every storey in the model
  for storey in SELECT('storey'):
    # Find areas on the current storey where no space has been defined ('undefined
space')
    undef_spaces = storey.SELECT('undefined space')

    # If the count (COUNT) of undefined spaces is 0,
    if undef_spaces.COUNT() == 0:
      # mark this storey as 'appropriate' (SUCCESS) and output a message
      storey.SUCCESS('Space objects are modeled for every area.')
    else:
      # otherwise, iterate over each undefined space
      for undefined_space in undef_spaces:
        # convert the area value to a number (NUMBER) in 'm2' units
        area = undefined_space.SELECT('area').UNIT('m2').NUMBER()
```

```
        # mark the space as 'inappropriate' (ERROR) and include the area in the
message
        undefined_space.ERROR('No space object (area: ' + str(area) + 'm2)')
  ]]>
</Script>
```

Now that the basic structure of a script is understood, let's take a closer look at how to use the core API functions that serve as the script's practical tools.

4. Core API Function Usage Guide

The API (Application Programming Interface) is the primary mechanism for querying a BIM model's semantic and geometric data. Mastering these functions is essential for writing complex and meaningful verification rules. This section provides a practical guide to the most important functions in a developer's toolkit.

4.1. Retrieving Data: SELECT(query, arg)

SELECT() is the basic function used to retrieve BIM model objects. It retrieves data by passing the type of information desired, as a string, to the first argument (query).

Below is a list of frequently used query parameters.

Parameter	Information Retrieved	Description
storey	Storey information	Returns a list of all Storey objects contained in the model
space	Space information	Returns all Space objects contained in a given object (e.g. a storey)
undefined space	Undefined-space information	Returns areas in which no space object has been created
element	Object information	Returns all building Element objects
door	Door object information	Returns all Door objects contained in a given object
window	Window object information	Returns all Window objects contained in a given object
name	Object name	Returns the object's Name property value as a string
width	Object width	Returns the object's Width value as a number
area	Object area	Returns the object's Area value as a number
element id	Object ID	Returns the object's unique ID as a string
prop	Specific property information	Pass the property name as the second argument (arg) to retrieve that property's value

Using the prop parameter

To query a specific property value, use the second argument of SELECT(). For example, to retrieve a beam object's 'fire-resistance status' property and convert it to a boolean, write the following.

```
# Retrieve the 'fire-resistance status' property value from the 'beam' object and
convert it with BOOL()
is_fire_resistant = beam.SELECT('prop', 'fire-resistance status').BOOL()

if is_fire_resistant:
  beam.SUCCESS('This beam has fire-resistant construction.')
```

4.2. Processing Results: SUCCESS(msg), FAIL(msg), WARNING(msg), ERROR(msg)

After the verification logic completes, message functions are used to report the result to the system. Each function represents a verification status, and the msg parameter passes, as a string, the message to be shown in the results window.

- SUCCESS(msg): used when the review result is appropriate.
- FAIL(msg) / ERROR(msg): used when the review result is inappropriate. Both functions perform the same function of reporting an inappropriate result and can be used interchangeably.
- WARNING(msg): used when the review result is not immediately inappropriate but requires user confirmation.

4.3. Data Conversion

Most data retrieved via SELECT() is an object of a specific type. To compare it in conditional statements or use it in calculations, it must first be converted to an appropriate data type.

- COUNT(): returns the number of retrieved items (an object list) as an integer. Useful for checking whether a list is empty.
- UNIT(unit): converts the unit of the retrieved value. The BIM model's default unit is meters (m), but for area or volume calculations it must be converted to units such as 'm2' or 'm3' to obtain the correct value.
- NUMBER(), STRING(), BOOL(): explicitly convert the retrieved value to a specific data type.
- NUMBER(): converts the value to a number (double) type. Essential for numeric comparisons.
- STRING(): converts the value to a string type.
- BOOL(): converts the value to a boolean (true/false) type. Used when working with true/false property values.

Combining these core functions makes it possible to implement everything from simple property checks to complex geometric and logical relationship verifications. Next, let's walk through a hands-on example that builds a ruleset from scratch to see how these functions are used in practice.

5. Practical Example: Checking the Space Classification Code Property

By turning theory into actual code, we can get a feel for the full flow of ruleset development. The goal of this example is to build a ruleset that "checks whether every Space object in the model has a correctly entered 'class code' (space classification code) property."

Step 1: Create the file and write the basic XML structure

First, create a new file named testRuleset.ccl and write the XML declaration and the top-level root element, <CcCodeDataSet>.

```
<?xml version="1.0" standalone="yes"?>
<CcCodeDataSet xmlns="http://tempuri.org/CcCodeDataSet.xsd">
  <!-- Ruleset content is added here -->
</CcCodeDataSet>
```

Step 2: Configure <CcSetting>

Add the <CcSetting> element to define the ruleset's overall settings. Set the ruleset's Id to 'TestRuleSet' and the name shown in the list to 'Quality Review Test Ruleset List'.

```
<?xml version="1.0" standalone="yes"?>
<CcCodeDataSet xmlns="http://tempuri.org/CcCodeDataSet.xsd">
  <!-- Step 2: Add the CcSetting element -->
  <CcSetting>
    <Editable>true</Editable>
    <ShowTypology>false</ShowTypology>
    <InfoUrl></InfoUrl>
    <Id>TestRuleSet</Id>
    <Title>Quality Review Test Ruleset List</Title>
    <RulesetName></RulesetName>
    <RulesetUpdateDate></RulesetUpdateDate>
    <ShowSearch>false</ShowSearch>
  </CcSetting>
  <!-- <CcCode> is added here -->
</CcCodeDataSet>
```

Step 3: Define the <CcCode> item

Now add a <CcCode> element to define the actual review item. Set the Id to '1001' and ParentId to '-1', because it is a top-level item. Use Number and Desc to clearly state what this rule does.

```
<?xml version="1.0" standalone="yes"?>
<CcCodeDataSet xmlns="http://tempuri.org/CcCodeDataSet.xsd">
  <CcSetting>
    <Editable>true</Editable>
    <ShowTypology>false</ShowTypology>
    <InfoUrl></InfoUrl>
    <Id>TestRuleSet</Id>
    <Title>Quality Review Test Ruleset List</Title>
    <RulesetName></RulesetName>
    <RulesetUpdateDate></RulesetUpdateDate>
    <ShowSearch>false</ShowSearch>
  </CcSetting>
  <!-- Step 3: Add the CcCode element -->
  <CcCode>
    <Id>1001</Id>
    <ParentId>-1</ParentId>
    <Number>Space Object Property Entry</Number>
    <Desc>Space objects must have a space classification code property.</Desc>
```

```
    <Script>
      <!-- The script is added here -->
    </Script>
  </CcCode>
</CcCodeDataSet>
```

Step 4: Implement the Python script logic

Inside the <Script> tag, define the def Check(): function, and write logic that uses the SELECT API to iterate over every storey and space and retrieve the 'class code' property.

Step 5: Conditional branching and result processing

Use an if statement to check whether the retrieved classCode value is empty. Complete the code so that, if it is empty, an ERROR message is issued, and if a value is present, a SUCCESS message is issued for the corresponding space object.

Final complete code

Combining all of the steps above, the complete code for the final testRuleset.ccl file is as follows.

```
<?xml version="1.0" standalone="yes"?>
<CcCodeDataSet xmlns="http://tempuri.org/CcCodeDataSet.xsd">
  <CcSetting>
    <Editable>true</Editable>
    <ShowTypology>false</ShowTypology>
    <InfoUrl></InfoUrl>
    <Id>TestRuleSet</Id>
    <Title>Quality Review Test Ruleset List</Title>
    <RulesetName></RulesetName>
    <RulesetUpdateDate></RulesetUpdateDate>
    <ShowSearch>false</ShowSearch>
  </CcSetting>
  <CcCode>
    <Id>1001</Id>
    <ParentId>-1</ParentId>
    <Number>Space Object Property Entry</Number>
    <Desc>Space objects must have a space classification code property.</Desc>
    <Script>
      <![CDATA[def Check():
      # Iterate through every Storey object in the model
      for storey in SELECT('storey'):
        # Query every Space object belonging to the current storey
        spaces = storey.SELECT('space')
        for space in spaces:
          # Retrieve the 'class code' (space classification code) property from each
space, as a string
          classCode = space.SELECT('class code').STRING()

          # Handle the result depending on whether the class code value is present
          if classCode == '':
            # If the property value is empty, mark it as 'inappropriate'
            space.ERROR('Space classification code property is missing.')
          else:
            # If the property is defined, mark it as 'appropriate' and include the
code value in the message
            space.SUCCESS('Space classification code property is defined. (' +
classCode + ')')
      ]]>
    </Script>
```

```
  </CcCode>
</CcCodeDataSet>
```

A ruleset file written in this way can be loaded and run in a quality-review program such as 'KBim Assess-Lite'. The program interprets the ruleset, inspects each space object, and displays the result on screen as 'SUCCESS' or 'ERROR'.

This example has walked through the overall flow of ruleset development. To write more diverse and complex rules, the next section provides a comprehensive reference covering every available API function.

6. Full API Function Reference

This section is a comprehensive reference that a developer can consult whenever specific BIM object information is needed. Every function (parameter) listed here is used as the first argument (query) of the SELECT() function.

6.1. Common

Property-query functions that can generally be applied to any object type.

Parameter	Return Type	Function / Purpose
name	string	Object name
width	distance	Width
height	distance	Height
length	distance	Length
area	area	Projected area (along the z-axis)
area-side	area	Side projected area
area-y	area	Width * height
area-x	area	Length * height
excluded area	area	Area excluding the projected areas of the specified objects (arg2: target object array)
perimeter	length	Perimeter
volume	volume	Volume
boundary line count	number	Number of edges of the projected polygon
elevation	length	Elevation
parent	object	Parent object
distance2	distance	2D distance to the target object (arg2: target object)
is name in	bool	Whether the name contains a specified string (arg2: string array)
element id	string	Object ID
element guid	string	Object GUID
prop	property value	A specific property value (arg2: property name)
center	vector3	Center point of the object
class code	string	Space classification code
footprint	polygon	Projected polygon
is contained in	bool	Whether the object is contained in the target object (arg2: target object)

Parameter	Return Type	Function / Purpose
level difference	distance	Height difference from the parent object
element type	string	Object type (e.g. 'IfcWall')
related objects	object array	Ifc related objects
adjacent element	object array	Adjacent objects
near element	object array	Objects located within a given distance (arg2: distance)
my storey	object	Storey the object belongs to
use code	string array	Use code
is target use	bool	Whether the building/storey use is included in the target use codes (arg2: target code array)
zone area	area	Total area of the building/storey corresponding to the target use codes (arg2: target code array)
max diagonal	distance	Maximum 2D diagonal distance
obstacle in circle	bool	Whether an obstacle exists within a given radius of the object's center (arg2: radius, height)
material	string	IfcMaterial information
is connected to upper storey	bool	Whether the stair or ramp is connected to the upper storey
get lower edge height	distance array	Lower-edge height of the object

6.2. Building & Storey

Functions for querying objects and information related to the entire building or a specific storey.

Parameter	Return Type	Function / Purpose
storey	object array	All storeys of the building
building type	string	Main use of the building
slab	object array	Slab objects
roof	object array	Roof objects
column	object array	Column objects
curtain wall	object array	Curtain-wall objects
element	object array	IfcBuildingElement-related objects
distribution element	object array	IfcDistributionElement-related objects
transport element	object array	IfcTransportElement-related objects
geographic element	object array	IfcGeographicElement-related objects
furnishing element	object array	IfcFurnishingElement-related objects
civil element	object array	IfcCivilElement-related objects
proxy	object array	IfcBuildingElementProxy objects
undefined space	object array	Areas in which no Space has been modeled
lift car	object array	Elevator
window	object array	Window objects
beam	object array	Beam objects
covering	object array	Covering objects
is above ground	bool	Whether the storey is above ground
floor	number	Number of floors
lower storey	object	Storey below the current storey

Parameter	Return Type	Function / Purpose
upper storey	object	Storey above the current storey
elevation above	distance	Elevation above ground

6.3. Space

Functions that query information or child objects relative to a Space object.

Parameter	Return Type	Function / Purpose
min headroom	distance	Effective ceiling height (minimum height from the floor to the ceiling)
ceiling height	distance	Height to the ceiling object
window	object array	Windows contained in the space
door	object array	Doors contained in the space
wall	object array	Walls forming the space
curtainwall	object array	Curtain walls contained in the space
stair	object array	Stairs contained in the space
edge	object array	Boundary edges of the space
escape route	route	Escape route (arg2: target object array — door or stair)
connected space	object array	Connected spaces that allow ventilation
containing space	object array	Spaces contained within the space
opening	object array	Openings in the space
clear width	distance array	Effective width of the space
side space	object array	Spaces beside the space
front back space	object array	Spaces in front of and behind the space
lower space	object array	Spaces on the storey below
railing	object array	Railings
floor	object array	Floors
ramp	object array	Ramps
covering	object array	Coverings
beam	object array	Beams
column	object array	Columns
containing element	object array	All objects contained in the space
floor level difference	distance	Level difference between floors within the space
is my door	bool	Whether the door belongs to this space (arg2: door object)
passable space	object array	Passable spaces connected via doors or openings

6.4. Door

Functions that check a door object's properties and its relationship with surrounding spaces.

Parameter	Return Type	Function / Purpose
is inward	bool	Whether the door opens inward (arg2: space object)
clear opening	distance	Clear width of the door
has clear manoueuvring space	polygon	Whether sufficient maneuvering space is secured in front of the door (arg2: 'width x length' array)
has clear leading edge space	polygon	Whether there is clearance beside the leading edge of

Parameter	Return Type	Function / Purpose
		the door (arg2: 'width x length')
has side space	polygon	Whether there is clearance beside the door (arg2: 'width x length')
is external entrance	bool	Whether the door is an external entrance
corridor width	distance	Width of the corridor connected to the door
outward space	object array	Space located outside the door
outward obstacle	object array	Obstacle located outside the door (arg2: 'distance, height')
operation type	string	The door's opening/closing method (e.g. hinged, sliding)
outward ramp	object array	Ramp located outward from the door
low level ramp	object array	Ramp installed on the lower-floor side of the door
is on same line	bool	Whether the object is aligned on the same line as another object (arg2: target object)
floor difference	distance	Interior/exterior floor level difference at the door
is double door	bool	Whether the door is a double (paired) door

6.5. Stair & Ramp

Functions that retrieve detailed dimensions and properties of stairs and ramps.

Parameter	Return Type	Function / Purpose
(stair) riser height	distance array	Riser height of the stair
(stair) tread width	distance array	Tread width of the stair
(stair) clear width	distance	Effective width of the stair (minimum of the tread widths)
(stair) rail height	distance array	Handrail height of the stair
(stair) rail / (ramp) railing	object array	All handrails
(stair) riser count	distance array	Number of stair risers
(stair) clear landing width	distance array	Effective width of the stair landing
(stair) is spiral	bool	Whether it is a spiral staircase
(stair/ramp) landing	object array	Landing object of the stair/ramp
(stair/ramp) flight	object array	Flight object (continuous run of steps) of the stair/ramp
(stair) is direct	bool	Whether it is a direct (straight-run) staircase
(stair) is connected to lower storey	bool	Whether the stair is connected to the storey below
(stair) short tread edge	object array	Short edge of the stair tread
(stair) long tread edge	object array	Long edge of the stair tread
(stair) distance	distance	Distance between the entry points of two stairs (arg2: target stair)
(ramp) level change	object array	Slope/level change of the ramp

6.6. Handrail

Functions that check the installation condition and specifications of a handrail.

Parameter	Return Type	Function / Purpose
is left	bool	Whether the handrail is positioned on the left
is continuous	bool	Whether the handrail is continuous
extension	distance array	Horizontal length of the handrail
grip width	distance	Width (thickness) of the handrail grip
is wall-mounted	bool	Whether the handrail is wall-mounted
distance from wall	distance	Gap between the handrail and the wall

6.7. Window & Wall

Functions that retrieve the properties and related information of windows and walls.

Parameter	Return Type	Function / Purpose
(window) opening area	area	Area of the window opening
(window) lower edge height	distance array	Height from the lower edge of the window to the finished floor
(window) dist from ceil	distance	Distance from the lower edge of the window to the ceiling
(window) railing	object array	Railing of the window
(window) wall related	object array	Wall adjacent to the window
(wall) isexterior	bool	Whether the wall is an exterior wall
(wall) opening	object array	Openings in the wall (windows, doors, etc.)