



PIComposer Community Edition

Chi W. Ng, Ph.D.

Version v0.1, 1.10.2023

Table of Contents

1. Introduction	1
2. Getting Started	4
2.1. Installation	4
2.2. Confirm email	4
3. Workspace and Projects	6
4. Project, Models and Templates	8
4.1. Spatial Template	8
4.2. Project Pane	11
4.3. Entity Templates	12
4.3.1. Simple Entity Template	13
4.3.2. Parametric Template	13
4.3.3. Template Transformation and Procedural Entity	14
4.4. Working with Entity Template	14
4.4.1. Type Filter	15
4.4.2. Tag Filter	16
4.4.3. Viewing and Editing Entity Template	17
5. PComposer BIM Model and Instance	19
5.1. Working with the Spatial Tree	20
5.1.1. Add Child	21
5.1.2. Add Template As Child	23
5.1.3. Other Spatial Tree Node Commands	23
5.2. Working with the Placement Tree	24
5.3. Working with Filters	25
5.3.1. Instance Type Filter	25
5.3.2. Instance Id Filter	26
5.4. Working with Instance Detail Tree	26
5.4.1. Instance Root Node	28
5.4.2. Component and Composite	29
5.4.3. Working with Attributes	29
5.4.4. Simple Attributes	30
5.4.5. Complex Attributes	31
5.4.6. Instance Aggregation and Composition	31
5.4.7. Working With Select	33
5.4.8. Working with Collection	34
5.5. Working with Propertyset, QuantitySet and Their Templates	35
5.5.1. Propertyset Template	36
5.5.2. Quantityset Template	38
5.6. Working with layers	39

5.6.1. Layer Filter	39
5.7. Ifc Exporting and 3d Viewer	40
6. Preferences	41
6.1. Status Displays.....	42
6.2. Model Page Options	42
6.3. Instance Detail options	43
6.4. Layer Filter options	44
6.5. Spatial Tree Node Command Options	44
6.6. Create Command options	45
6.7. Private Template Root Node Command options.....	45
6.8. login options:.....	45
7. User Account	46

Chapter 1. Introduction

PIComposer is a STEP ISO-10303 Standard document editor powered by a high performant EXPRESS document store. To satisfy its aim of support real time applications the document store has its own lossless STEP compliant proprietary encoding of EXPRESS data.

Moreover, to deal with models that will not fit into device memory, PIComposer lazy loads all data on demand. EXPRESS model and its entity instances (or instances) are faithfully presented in its user interface as trees and directed acyclic graphs.



PIComposer community edition: although the document store is capable of handling any STEP AP, it is restricted to the latest version of IFC: ifc4x3. To keep our discussion concrete, for the rest of this document we will concentrate on IFC.



A user of PIComposer community edition is assumed to have some knowledge of, or at least a desire, to learn [IFC](#). Otherwise, what follows in this document and the application PIComposer itself will make little sense, as PIComposer manipulates IFC models by providing user interfaces that directly interact with IFC model content: entity, select, enum, etc.

PIComposer has an architecture similar to that of an Entity Component System ([ECS](#)).

In PIComposer, entities and components are stored as documents, and systems are internal functions or external scripts written in dart or C++ that operate on entities and components.

A PIComposer entity is an instance of a non-abstract IfcProduct subtype. Within a model, each instance has an immutable 64-bit integer identifier.

A PIComposer component is a container of a set of IFC instances; it composes and aggregates a group of logically related IFC instances into a single document. For example, an IfcLocalPlacement component comprises of axis entity instances, position entity instance, etc. All the data that would completely define IfcLocalPlacement are placed in a single store document.



Only non-abstract entity types could be instantiated as instance.

Currently, component must be of the following type:

- IfcShapeRepresentation
- IfcLocalPlacement
- IfcRelDefinedByType
- IfcRelDefinedByProperties
- IfcPresentationLayerAssignment
- Spatial relationship entities:
 - IfcRelAggregates
 - IfcRelContainedInSpatialStructure

- Other subtype of IfcRelationship

In addition to entities and components, other concrete IFC types could be instantiated into the model as free form instance and will be stored as a store document.

Most BIM modeling work flow in PComposer such as—creating an model, modeling an real world object say a table with an instance within a model—can be assisted by utilizing templates. Templates are also the primary mechanism for a user to configure, enhance, or customize PComposer. A template, at its most basic, is a set of pre-programmed instructions to be executed by PComposer where the language is either dart, C++ or json.

PComposer supports multiple types of templates. They include:

- spatial template
- instance template
 - (simple) template
 - parametric template
 - transform
 - procedural entity
- propertyset template
- quantityset template
- enum template

A spatial template allows a user to scaffold an IFC model during initial creation. A spatial template specifies the initial model spatial structure such as the number of IfcBuildings, IfcBuildinStoreys, each spatial element's relative placement, etc.

A spatial template is authored using json.

When a spatial template is executed, a model with a root IfcProject and related context information such as model unit system is also created.

In PComposer, geometry uses millimeter for distance measure and radian for angle measure. Property and quantity conforms to MKS unit measure system. This convention defines an implicit context for templates.



PComposer does not support imperial unit system or other non SI system.

Instance templates are instance factories, they are used to simplified instance creation in a model. There are 4 different types of instance templates and they must all be of IfcShapeRepresentation type or of a IfcProduct subtype.

A simple template is an extract of a component or instance in the scope of a project that could be reinstantiated in a model. It is the most basic form of reuse in PComposer—a form of copy and paste across space and time. A template could be private to a user's project or be shared in the cloud and be available for all PComposer users.

A parametric template is a template that exposes its instances and their attribute to direct user manipulation. It is configured via json.

A transform is a parametric template with an associated procedure. The procedure may take a simple template and output something far more complex. An example transformation being the input a brep box and the output a frustum.

Procedural entity, as its name implies, is a creational procedure that instantiates an entity's instance. A few examples of procedural entity are provided with PIComposer.

Since not all propertyset and quantityset are published within the ifc schema EXPRESS file, these missing propertyset, quantityset, and user defined propertyset must be configured using templates, so that PIComposer could instantiate them. These templates are json files. Many examples are provided with the PIComposer release.

The ifc4x3 schema has 130 defined types, more than 240 enum types, almost 100 select types, and nearly 900 entity types. In a typical office tower IFC model, it is common to have tens of million of entity instances. To analyze and dissect this massive volume and variety of BIM data set, PIComposer provides a multitude of filters.

In a model, instances could be filtered by:

- instance type
- instance id and range
- tag (instance could be tagged and searched)
- layer

Templates could be filtered by: type and tag.

IFC models are 3d datasets. For the community edition, 3d view is provided via integration with web-ifc-viewer from the Ifc.js project.

Finally, in closing:

PIComposer community edition, release free of charge, is a tool that the author wished was available when he first started learning and working with BIM while working at Gehry Technologies. With its versatile and powerful template systems, simple data presentation, it is a great tool for learning, exploring and creating BIM data.

Chapter 2. Getting Started

PIComposer is a connected desktop system, a user must have access to PIComposer cloud service to use PIComposer's essential features. Therefore, a user must create an account with PIComposer cloud service. First, a user must install PIComposer.

2.1. Installation

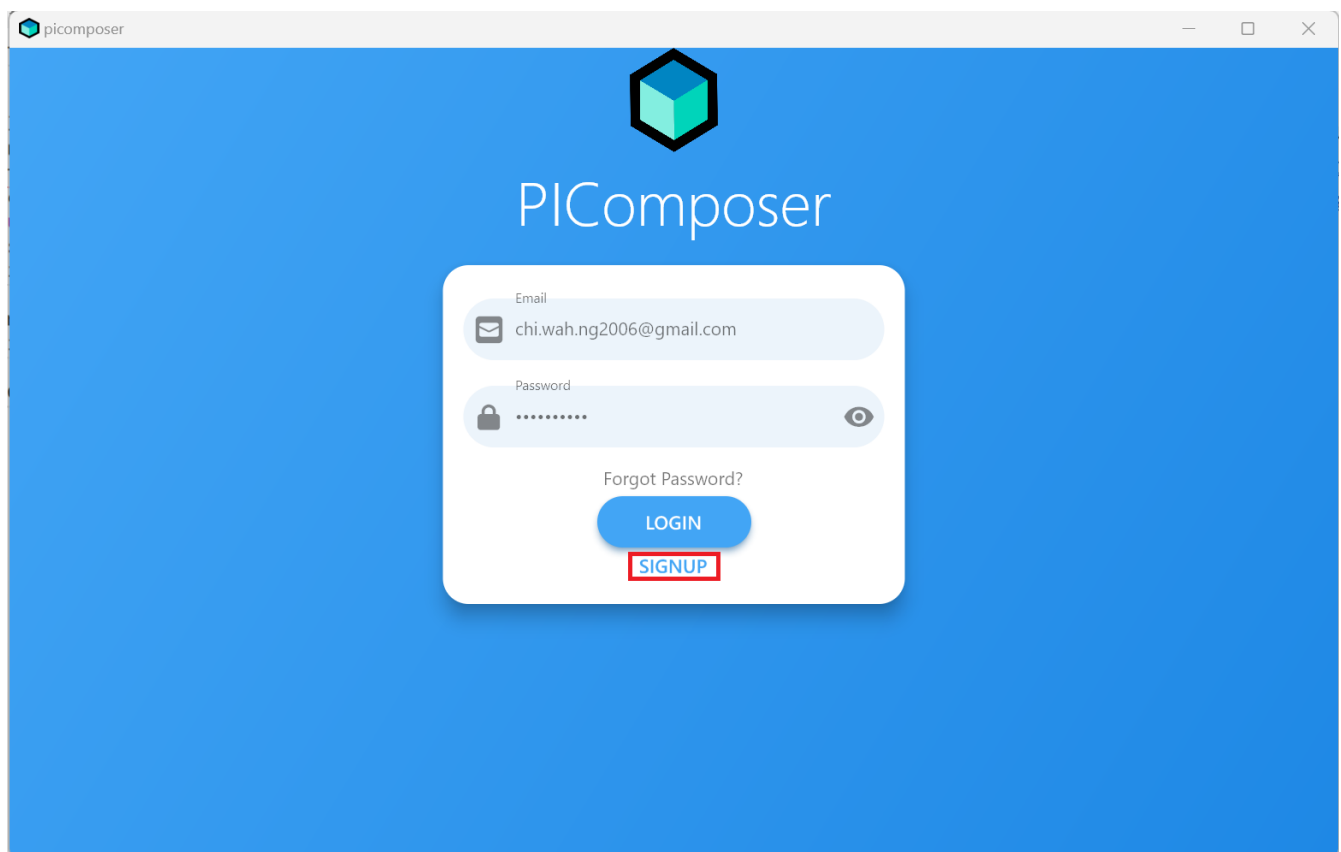
To install PIComposer, first download the installer from the [PIComposer Community Edition](#) github repository.

The installer is a self signed msix package.

To enable the installer, install the self signed certificate by following [these steps](#). Once the certificate is installed, install PIComposer.

Next, launch PIComposer and follow the signup workflow by clicking the signup button.

Read and accept licensing terms.



signup screen

2.2. Confirm email

Confirm your email to complete your signup process and PIComposer is ready for use.

Confirm Your Signup

Inbox x



noreply@mail.app.supabase.io
to me ▼

Confirm your signup

Follow this link to confirm your user:

[Confirm your mail](#)

email confirmation



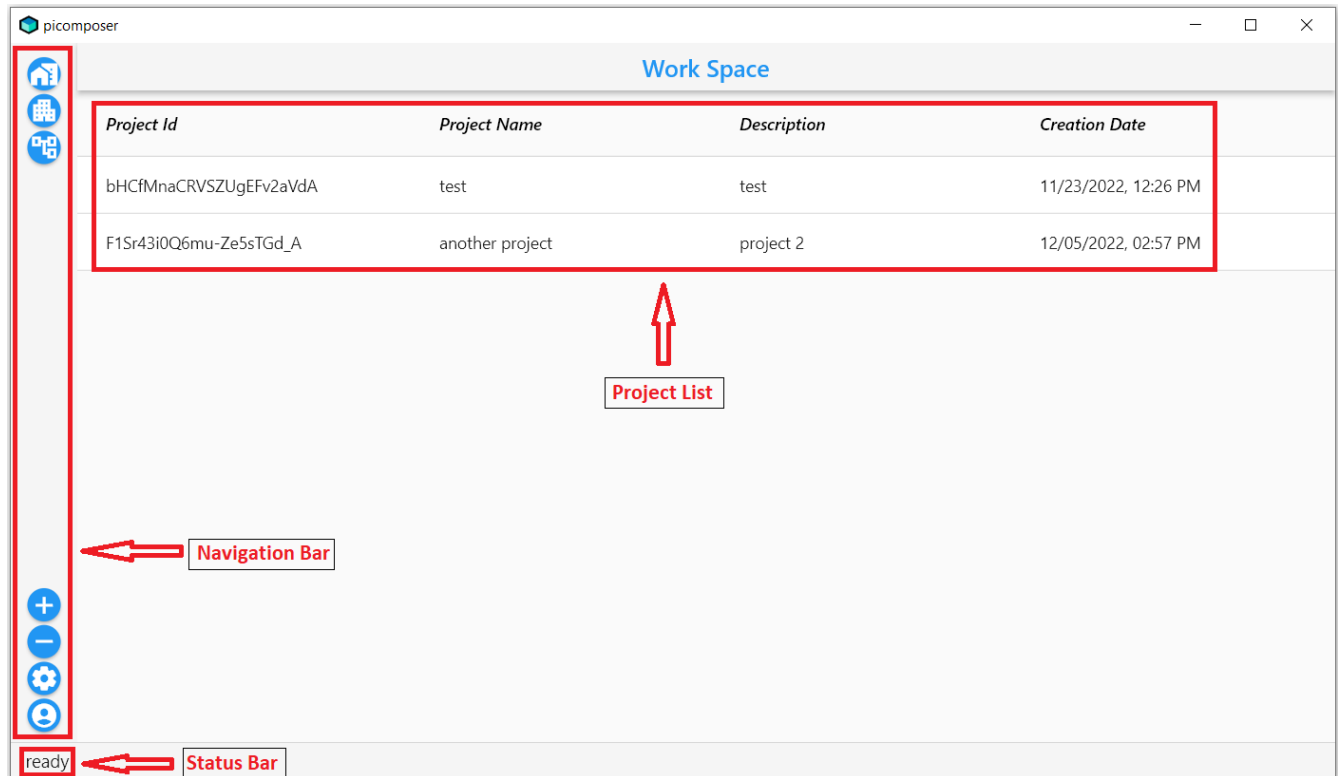
It might takes a few minutes for the confirmation email to be sent. Some email client might place the confirmation email in the spam folder.

Chapter 3. Workspace and Projects

PIComposer is a multi-user system. Each login user on a device has his or her own distinct workspace.

In a PIComposer workspace, users organize their projects. A project is collection of models and documents.

After successful authentication, a user is first routed to the workspace pane.



The workspace pane

There are three main user interface components:

- The project list
- The navigation bar
- The status bar

Projects are listed in the project list pane.

To add a project, click the add button .

To remove a project, select it on the project list, click the minus button .



Most PIComposer commands are context depend. The add and minus button, for example, would add or remove a models in the project pane.



The workspace button is the home button. It routes to the workspace pane from anywhere in PComposer.



The project button activates the project pane and displays the selected project.



Double clicking on any project on the project list also activates the project pane and displays the project.



The model button activates the model pane and displays the last active model. See the [\[PComposer BIM models\]](#) section for more detail about how to work with BIM models.



The preference button activates the preference page. See the [\[Preferences\]](#) section for more detail.



The account button activates the account page. See the [User Account](#) section for more detail.

PComposer provides user feedback using the status bar. Depending on the nature of the command, the status shows:

- A progress bar for long running command
- Notification of successful completion of a command
- Error message for failed command

Each type of status can be turned on and off using preferences.

Chapter 4. Project, Models and Templates

PIComposer manages models and templates within a project. Model metadata and private templates are persisted in a project document store; templates within the project are available to all models. Users enter the project pane from the workspace.

4.1. Spatial Template

The EXPRESS models are extremely powerful and versatile. Unfortunately, to achieve such versatility, it remains extremely verbose. To name just a few instance types that must be included in a bare minimum BIM model:

- Header objects
- IfcProject
- IfcGeometricRepresentationContext
- IfcUnitAssignment
- multiple instances of IfcUnit
- IfcApplication
- IfcPerson
- IfcOrganization
- IfcPersonAndOrganization
- ...

To cut down on boilerplates, PIComposer provides a multitude of templates.

PIComposer uses spatial templates to scaffold a model. A static spatial template is a user provided json file specifying the initial model spatial hierarchy. A example spatial template is listed below:

```
{
  "schema": "ifc4x3",
  "element": {
    "type": "ifcproject",
    "name": "from_template",
    "children": [
      {
        "type": "ifcsite",
        "name": "bluff",
        "location": [
          0.0,
          0.0,
          0.0
        ],
        "children": [
          {
            "type": "ifcbuilding",
```

```
"name": "East Wing",
"location": [
    0.0,
    0.0,
    0.0
],
"children": [
    {
        "type": "ifcbuildingstorey",
        "name": "first floor",
        "tag": "floor.1",
        "location": [
            0.0,
            0.0,
            0.0
        ]
    },
    {
        "type": "ifcbuildingstorey",
        "name": "second floor",
        "tag": "floor.2",
        "location": [
            0.0,
            0.0,
            2300.0
        ]
    }
]
}
}
```

A spatial template is a json object and it defines a recursive structure. To understand the json structure, it might be easier to look at the corresponding dart object that the json object is deserialized to.

```
@JsonSerializable(implicitToJson: true)
class SpatialElement {
    String? icon;
    String? schema;
    Element element;
    SpatialElement({
        this.icon,
        this.schema = 'ifc4x3',
        required this.element,
    });
}
```

```

factory SpatialElement.fromJson(Map<String, dynamic> json) =>
    _$SpatialElementFromJson(json);

Map<String, dynamic> toJson() => _$SpatialElementToJson(this);
}

@JsonSerializable(explicitToJson: true)
class Element {
    String type;
    String? name;
    String? tag;
    String? description;
    String? relType;
    int? count;
    List<double>? location;
    List<double>? axis;
    List<double>? refDirection;
    List<Element>? children;
    Element({
        required this.type,
        this.name,
        this.tag,
        this.description,
        this.relType,
        this.count,
        this.location,
        this.axis,
        this.refDirection,
        this.children,
    });

    factory Element.fromJson(Map<String, dynamic> json) =>
        _$ElementFromJson(json);

    Map<String, dynamic> toJson() => _$ElementToJson(this);
}

```

A spatial template is first deserialized into a SpatialElement object. For PCompose to create a model from this template, the SpatialElement object must have:

- Schema—the schema name of the model, must be ifc4x3
- Element—the root instance in the spatial hierarchy, must be IfcProject type

Starting from the root IfcProject element, the rest of the spatial structure is defined recursively; IfcProject element have "children", which in turn might have its own children, etc.

For each object of type Element we have:

- Type — concrete subtype of IfcSpatialStructureElement, for example, IfcBuilding
- Name — name of the instance, optional

- Tag — enable search by tag, optional
- Description — description, optional
- Reltype — relationship type to parent, default to IfcRelAggregates, optional
- Count—number of instance to create, default to 1, optional
- Location—object placement location, optional
- Axis—object placement axis, optional
- RefDirection—object placement refDirection, optional
- Children—decendants, optional



use the "location" coordinates to set story elevation, like it is done on second floor of the example

When executing a spatial template, PIComposer creates the IfcProject, the necessary headers, contextual objects, and the model's unit system. PIComposer creates two parallel hierarchy or trees:

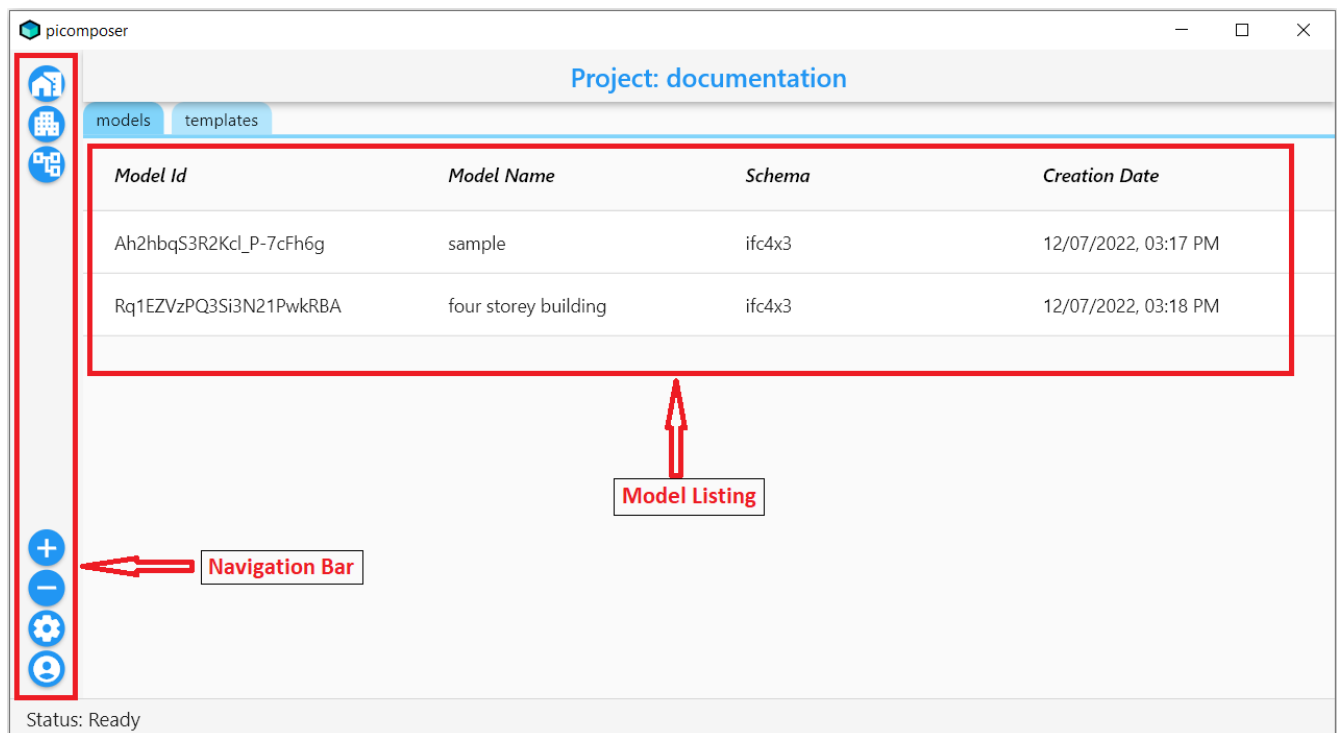
- The spatial tree that defines the logical containment/parent and children relationship
- placement tree. Child placement is always relative to its containing parent.



Spatial template are placed in the installation subfolder: `picomposer_data/spatial_template/`

4.2. Project Pane

User works with models and templates in the project pane.



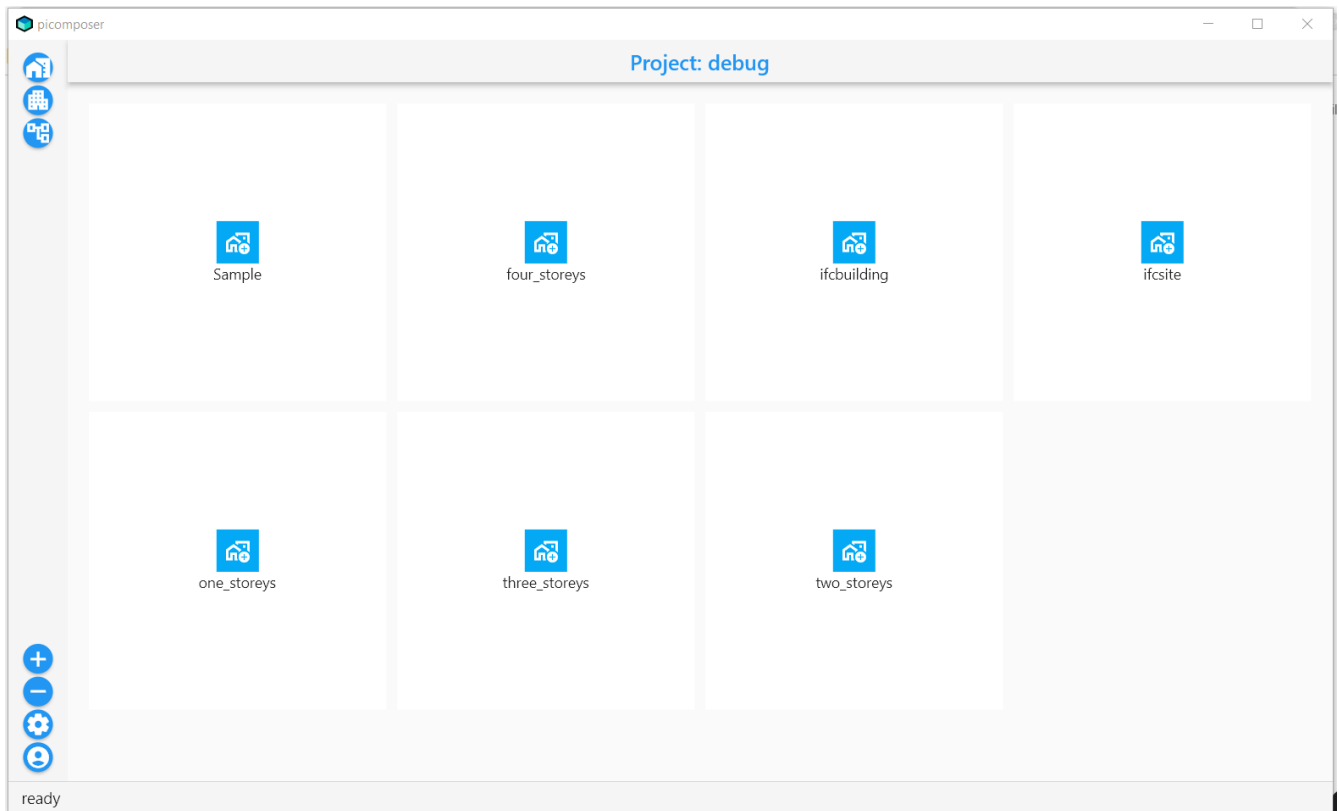
Project pane, model tab

To add a model, while the active tab is model, click the



button. This routes the user to the

template list.



Spatial template tile list

Click on a desired spatial template on the template list and complete the model creation process by entering the model name on the dialog.



All spatial template in the `spatial_template` folder appears in the template list, including user created ones.

4.3. Entity Templates

Entity templates are the core unit of use in PICOcomposer models. They are the equivalence to CATIA's Power Copy. Only they are simpler, much easier to create and in many aspects, more flexible and powerful. Moreover, they could be shared and available to all users.

They are easy to create—they are derived from the basic building blocks of a model a user already created.

They are customizable—they add dynamic behavior to a static element and model, allowing users to manipulate only the essential property of an component.

They are extensible—they could be associated with a script written in dart or C++ and transform simple component into highly complex ones.

They could be a pure component creational procedure in dart or C++.

4.3.1. Simple Entity Template

Any component of type IfcShapeRepresentation and IfcProduct subtype in PIComposer could be extracted as a simple template. A template groups a set of entity instances into package and reuse in a compatible context. Some components may contain complex spatial hierarchy. For example, an IfcWall template may contain openings and doors/windows. A private template is stored in the project store and is available to all models in a project.

A shared template is a template uploaded to the cloud and available to all users. A user uses the template by downloading one into a project's private template collection. A template has a unique identifier for life time management and tracking. Only the shared template author is allowed to remove or update a shared template. A shared template update is made by re-uploading the shared private copy.



PIComposer releases many shared templates.

4.3.2. Parametric Template

A parametric template is a simple template with internal attributes exposed to user manipulation allowing for simple transform of the simple template. Input parameters are specified by a json file. Multiple json files are possible for a given simple template. Each parametric template has a unique identifier. These identifiers could be copy/pasted from PIComposer. See the [Viewing and Editing Entity Template](#) section for details.

We explain the parametric template's structure by examining the example below:

```
{
  "is_procedural": false,
  "interactive": true,
  "template_id": "QK4heQ6jS-ikgAnbBYxEyg",
  "transform_id": "YTtBFTBCSViKoVBsqEhs1w",
  "description": "cylindrical extrude transform",
  "attributes": [
    {
      "value": {
        "name": "depth",
        "fun_type": "REAL",
        "value": null,
        "type": "RealValue"
      },
      "path": {"type": 1, "nodes": [[6, 0, 0, [8, 2495730769]], [1, 0, 2], [2, 0, [7, 1494207214]]], "index": [3, null, null, null]},
    },
    {
      "value": {
        "name": "radius",
        "fun_type": "REAL",
        "value": null,
        "type": "RealValue"
      }
    }
  ]
}
```



```

    },
    "path": { "type": 1, "nodes": [[6, 0, 0, [8, 2495730769]], [1, 0, 2], [2, 0, [7, 1494207214]], [2, 0, [5, 1936262883]]], "index": [3, null, null, null]}
  },
  ],
  "defaults": []
}

```

- `is_procedural`—always false for a parametric template, no associated procedure.
- `interactive`—should mostly be true, unless the template expect no user input.
- `template_id`—the unique identifier of the template we wish to parametrize.
- `transform_id`—the parametric template identifier.
- `description`—the description of the parametric template. This appears on the template listing.
- `attributes`—the set of parameters for the template.
 - `value`:
 - `name`—the name of the parameter, this will appear in the interactive dialog.
 - `fun_type`—the possible types are: REAL, INT64, BOOLEAN
 - `value`—the exchanged value of this attribute
 - `type`—the type could be: RealValue, IntValue, BoolValue
 - `path`—the path location of the attribute. See the [Viewing and Editing Entity Template](#) section on how to obtain this value.
- `defaults`—default values for the attribute, used to populate the interactive dialog.



A parametric template is a simple transformation template. Their json files are placed in the installation folder `/picomposer_data/template_transform/template_id/`

4.3.3. Template Transformation and Procedural Entity

A template transformation is a parametric template associated with a procedure written either in dart or C++.

Procedural Entity is a dart or C++ script that interacts with and updates the model directly.

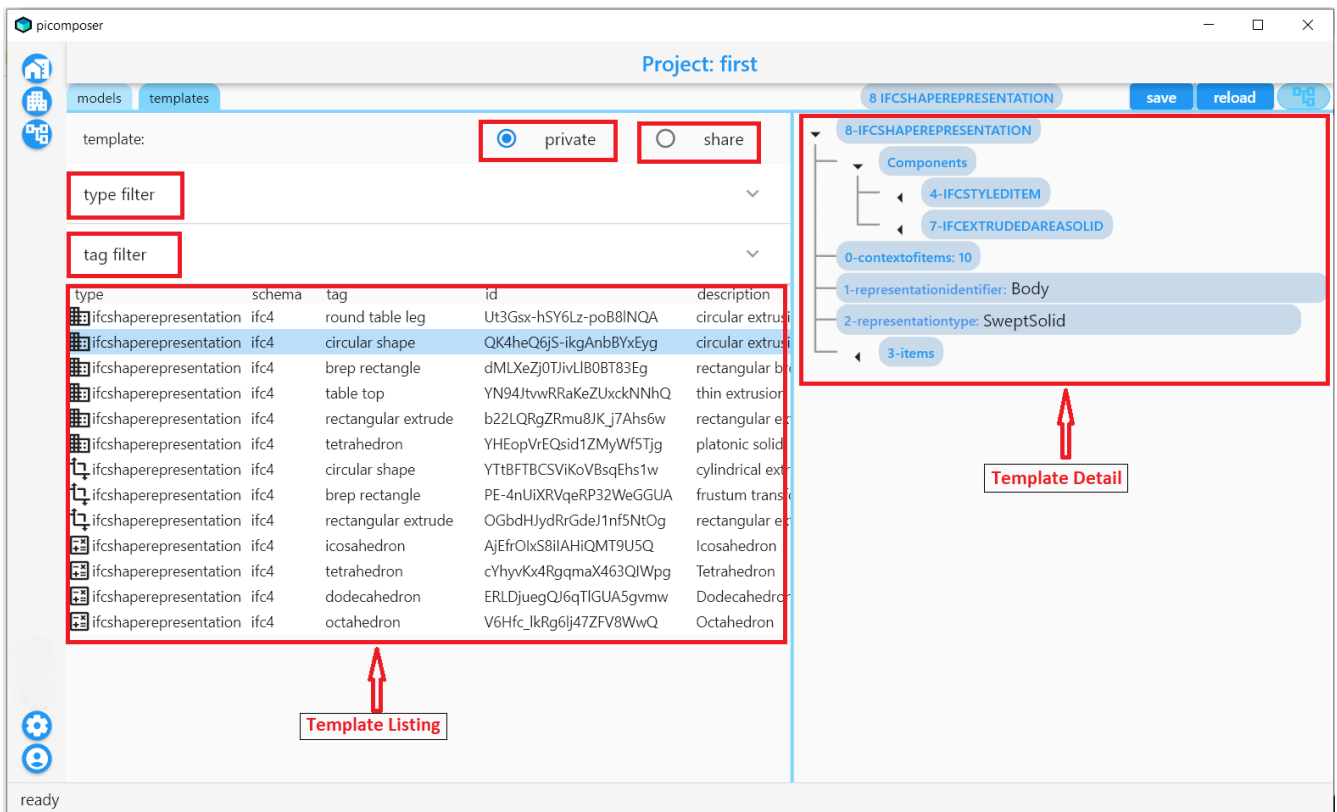
PIComposer releases many examples of template transformations and procedure entities.



Tooling for creating template transform and procedure entity is not available in PIComposer Community edition.

4.4. Working with Entity Template

Entity templates are created in the model page and are managed in the project pane under the template tab.

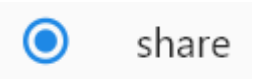


Project pane, template tab

Shared templates are templates published to the cloud and is accessible to all PICOMPOSER users.

- Shared templates are immutable
- Updated by the republication a private copy
- Update privileges are limited to the owner of the shared template

Since templates are mostly stored in databases, in the cloud or locally on a device, they must be retrieved using database query. PICOMPOSER distills these query into filters.

To browse the shared library, click the  **share** shared radio button and use the type or tag filter.



When the template tab is activated, PICOMPOSER loads a list of template types, setting it up for the type filter.



When a user clicks the shared button, the list of types is retrieved from the cloud, users will experience some latency.

4.4.1. Type Filter

Filters plays an essential role in accessing and using PICOMPOSER's data content. There are type filters for both entity instances in a model and templates in a project.

template: ☒ private ☐ share

type filter type ^

type: ifcfurniture ▼

Type filter

All type filters function the same way.

To use the type filter, first select the type:

1. Select a type from the dropdown list
2. Select **custom type** text field, and manually key in the type.
3. To list all, type the word **all** in the custom text field

Then click on the type type filter button.



Type filter on the shared template library is executed on the cloud.

4.4.2. Tag Filter

For search and retrieval purpose, both entity instance and template can be tagged at creation time. Tags are indexed in document stores. A shared template has the same tag as the private copy.

template: ☒ private ☐ share

type filter ▼

tag filter tag.. ^

tag tetrahedron

Tag filter

To apply the tag filter, enter the tag in the textfield and click the tag.. tag filter button.

4.4.3. Viewing and Editing Entity Template

PIComposer's main tool in viewing and editing a BIM model element is through the instance detail tree (technically, it is acyclic graph).

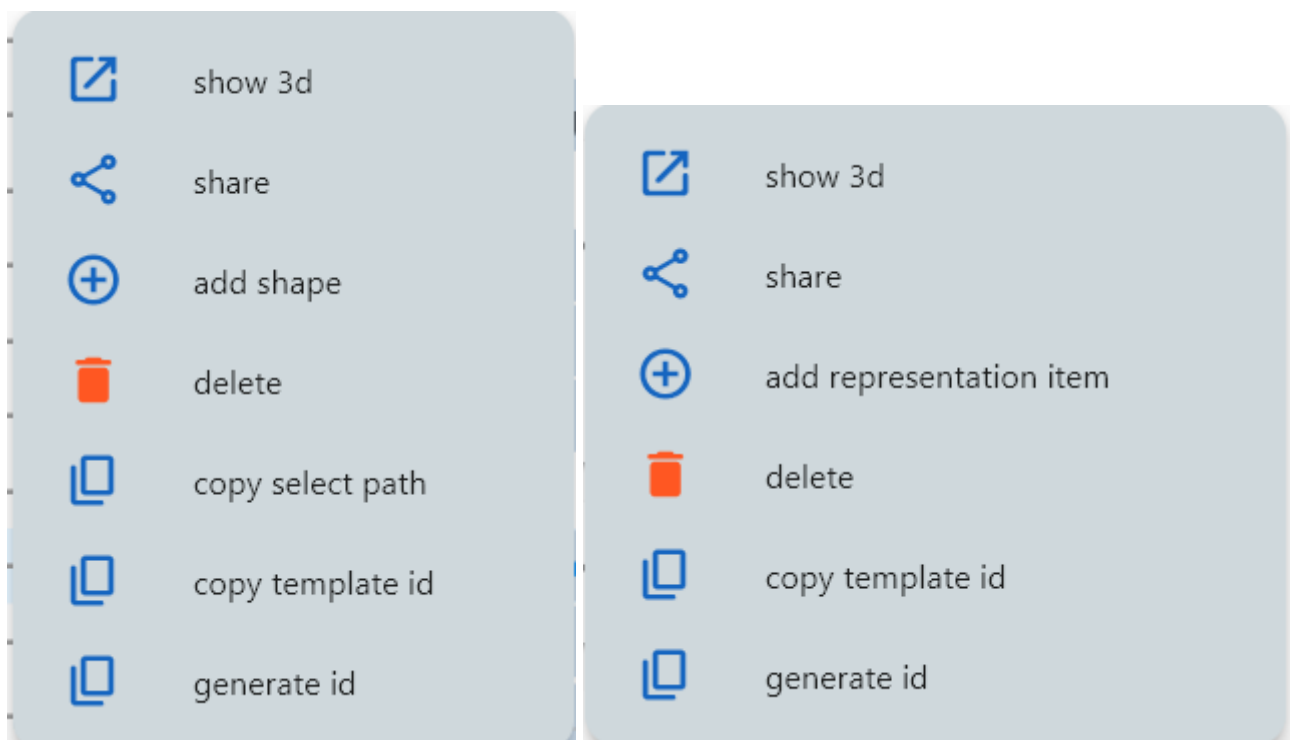


Since a template is a special instance of an entity instance, details regarding template editing, see the instance detail graph in the model session.

Commands on a node of instance detail graph are context sensitive, that is, PIComposer only allow commands relevant node type.

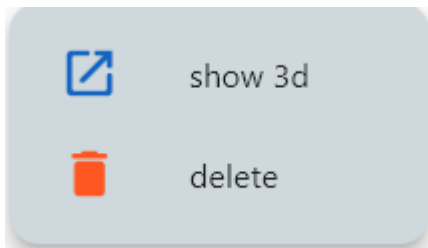
To interact with a template, select it on the template list. All commands related to the loaded template is available on the root node of the tree.

Private template command menu



- show 3d—launch browser and show template in 3d viewer
- share—publish private template to PIComposer communal cloud
- add shape—add a IfcRepresentationShape to the IfcProduct
- add representation item—add a representation item to a IfcRepresentationShape
- delete—remove the template from project
- copy select path—copy the path of the selected node to clipboard, helpful for creating parametric template
- copy template id—copy the current template id to clipboard
- generate id—generate a UUID and copy it to clipboard

Shared templates, private parametric templates, transformation templates and procedure entities are immutable.



commands for readonly template



Each attribute of a private template are mutable like those of an entity instance in a model.

Chapter 5. PComposer BIM Model and Instance

A PComposer model is a collection of instances which could be grouped into logical components. Moreover, each PComposer model embodies two essential concepts:

- spatial structure
- object placement.

The spatial structure enriches a BIM model with a logical organizational structure, while object placement defines the physical location of a instance within the model. Each of these concepts are presented as trees in PComposer.

These trees are managed in parallel; a node created in the spatial tree will trigger a corresponding creation of a node in the placement tree.

The placement tree structure is immutable.



Placement tree could be turned of by preference

Each instance is also presented as a tree (an acyclic graph actually)—the instance detail tree—where a tree node is either a logical grouping node or an attriubted node of the instance. All instances attribute addition, modification and deletion are managed using the instance detail tree.

Many instance's component could be displayed on the instance detail tree, including:

1. propertyset/quantityset
2. object placement
3. placement absolute coordinate
4. typeproduct
5. shapes

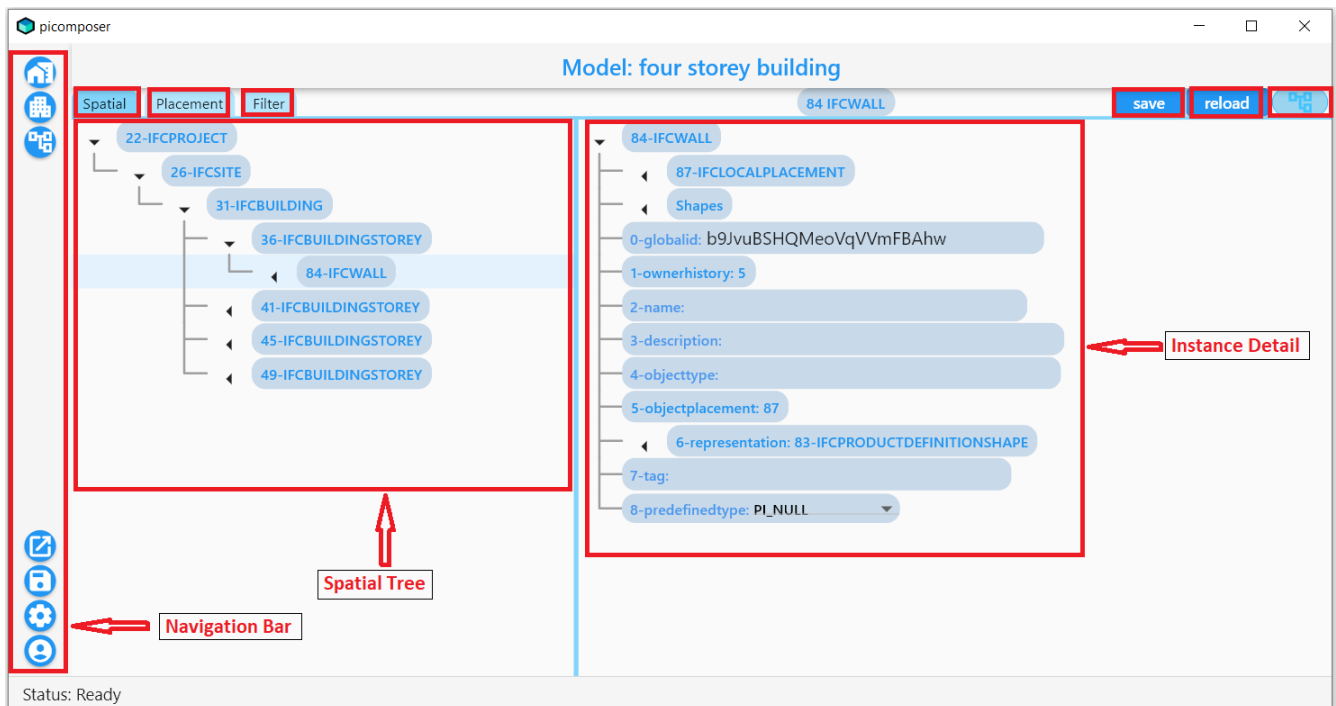
This information could be turned on/off using user preference.

PComposer Community Edition is a single document application. To load a model, select a model in

the Project pane model listing and click the



model button or simply double click on the item on the model list.



Model pane, spatial tree



User could return to the last active model by clicking the



model button

PICOMPOSER splits model presentation into two areas:

- the model structure
- model instance detail

The model structure area comprises of three tabs

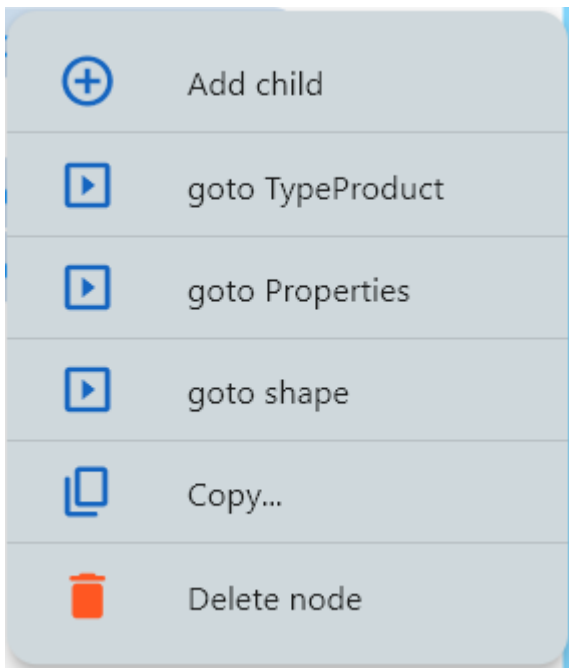
- spatial
- placement
- filters



Selection set is an essential component of the PICOMPOSER user interface. Each of these tabs above have a selection set. Many PICOMPOSER commands make users of a selection set. The selection set allows a user to create inter-node links, insert selected nodes into other nodes, and enable many other actions.

5.1. Working with the Spatial Tree

The spatial tree is where the spatial structure is managed. The commands to manipulate the tree are directly available on the tree node as popup menu items.



typical spatial command



All tree commands are context sensitive, i.e. Their availability and action depends on the node type and the content of the node.

The commands available to the spatial tree nodes are:

- add child—add a IfcSpatialStructural element or IfcProduct to the spatial structure.
- add template child—instantiate the selected IfcProduct template in the filter tab to the spatial structure.
- delete child—delete the select node
- go to propertyset/quantityset—show the current node's propertyset/quantityset in the instance detail tab.
- go to shape—show the IfcShapeRepresentation of the node
- go to typed product—show the IfcTypedProduct of the node in the detail tab.
- copy—copy the current to clipboard
- paste—paste the clipboard content as child.



Only leaf node can be deleted, except nodes with child opening or fill.

5.1.1. Add Child

To add a child to the spatial tree, select the parent node, right click on it to activate the node menu,

and select the



Add child

add child menu item. This command allow the user to create three sets of information:

- the ifcproduct entity

- the shape component and the material for the shape
- local object placement

The image shows a software dialog box with three tabs: "Product", "Placement", and "ShapeRepresentation". The "ShapeRepresentation" tab is currently selected. Inside this tab, there are three input fields. The first is a dropdown menu with the text "custom IfcProduct" and a downward arrow. The second is a dropdown menu labeled "IfcRelationship:" with the text "auto" and a downward arrow. The third is a text input field containing the text "tag". At the bottom right of the dialog, there are two buttons: "Ok" and "Cancel".

product creation dialog.

Fill in the necessary information on each tab and click OK when done.



To cancel adding IfcShapeRepresentation, leave the type dropdown to 'null'.



Adding children using the spatial tree node menu always adds a corresponding placement node to the placement tree. The child is placed relative the parent.



Adding a child also creates multiple inverse links in the database to manage the various relationships between the node in different trees. To have the correct model content appear on the trees, it is best to use the available commands.

5.1.2. Add Template As Child

This command uses filter selection set. To instantiate an IfcProduct template as a child of a spatial node, follow these steps:

1. activate the template tab
2. check the template radio button
3. use the type or tag filter to find the desired template, see the section [Working with Entity Template](#).
4. select the template

To add a template as child to the spatial structure, select the parent node, right click on it to activate



add child template

the node menu, and select the add child template menu item. Enter the necessary information in the template creation dialog if the template is interactive.

5.1.3. Other Spatial Tree Node Commands



Delete node

The delete child command deletes the selected node and the corresponding placement node in the placement. If the inverse link count to the shared IfcShapeRepresentation is 0, the IfcShapeRepresentation will be deleted from the model.



Copy...

The copy command copies the node to clipboard. Only leaf node can be copy.



Copying exception: nodes that could be templates are copyable. For exmple an IfcWall with openings and doors/windows are copyable although it has openings as children



Paste

The paste command pastes the content of the clip to the tree.



Unlike templates, which do not package IfcTypedProduct, IfcPropertyset components, the copy/paste dual do.



goto TypeProduct

The go-to typeproduct command

shows the IfcTypedProduct component of an entity in the detail tree.



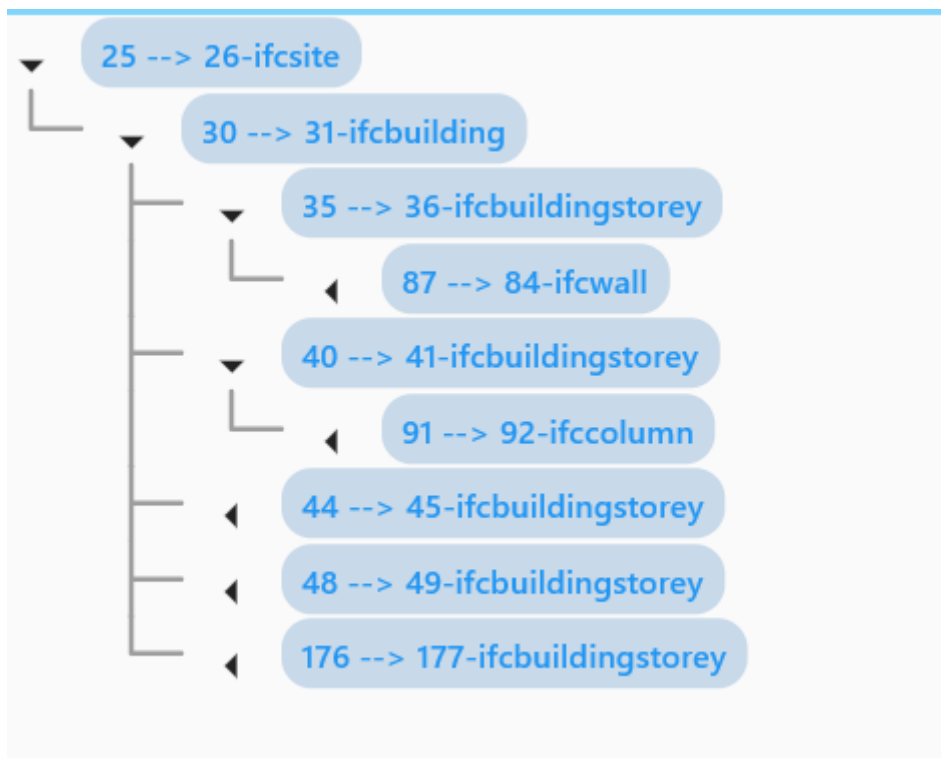
The go-to xxx commands apply the instance id filter to the targeted component to show the component in the detail tree.

5.2. Working with the Placement Tree

To underline its importance in the BIM model, the user interface includes a placement tree.

The placement tree is readonly, its structure is created during the creation of the spatial tree. Its node are updated using the instance detail tree.

To access the placement tree, activate the placement tab.



The placement tree



goto placed instance

The only command available to the placement tree is which navigates to the IfcLocalPacement's placed instance.



The go-to placed instance command uses an inverse link to find the placed instance.



The placement tree can be hidden/shown by setting the hide flag in user preference.

5.3. Working with Filters

Filters allows a user to search and retrieve an item in the model or project quickly. The filter tab also provides the workbench for dealing with free form entity instance and a place to work with entities outside of the spatial structure framework. There are two types of filters in the model pane: instance and template.

For details about template filters see the section [Working with Entity Template](#). We will concentrate on instance filter.

The screenshot shows the 'Filter' tab in a software interface. At the top, there are three tabs: 'Spatial', 'Placement', and 'Filter'. Below the tabs, there is a 'mode:' section with two radio buttons: 'instance' (selected) and 'template'. The main area is divided into four sections: 'type filter', 'tag filter', 'id filter', and 'layer filter'. Each section has a text input field and a button. The 'type filter' section has a 'type' button, a 'subtype...' button, and a 'create' button. The 'tag filter' section has a 'tag..' button. The 'id filter' section has an 'instance id..' button. The 'layer filter' section has a 'filter' button. The 'type' input field is set to 'custom type'. The 'tag' input field is empty. The 'instance id' input field is set to 'instance ids'. The 'layer' input field is set to '372 -- blue'.

Instance Filter

5.3.1. Instance Type Filter

The basic functionality of the instance type filter is similar to template [Type Filter](#). Instance filter has two addition features:


- filter by subtypes

- create entity instance of the filter type

The subtype filter functions same way the type filter does except that it consider all subtypes of the given type in the type textfield.



The type could be abstract for the subtype filter.

To create instance of the desire type, enter the type in the type text box and click  the create button.



To create an instance of type not on the dropdown list, enter the type using the 'custom type' text box. The type user wants to create must not be abstract.



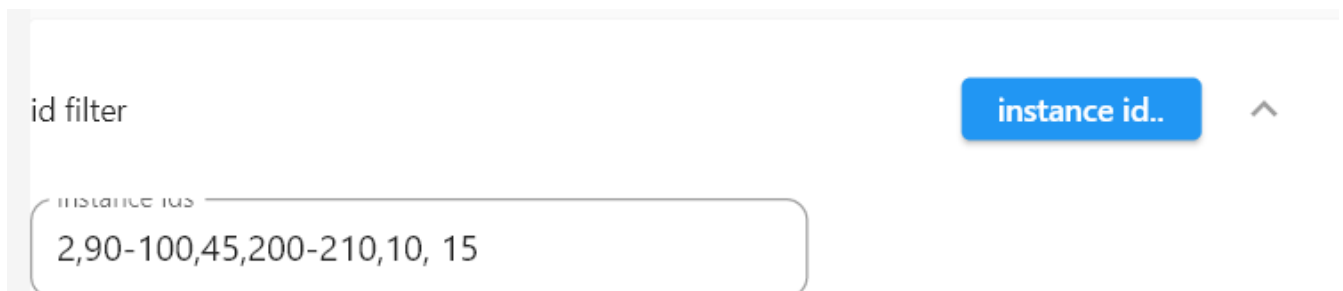
Instance type filter is limited to return 1000 result instances.



The instance tag filter functions exactly like the tempate tag filter, see [Tag Filter](#).


5.3.2. Instance Id Filter

Instance type filter is based on a list of specific instance ids and a list of ranges. If users who had a user print will be familiar with the paradigm used.



Instance id Filter

To use the id filter, first enter the search string in the instance id text box. The syntax is a common separated list of positive integers and a list of integer range specified using a pair of positive integer a dash. In the example above, we are looking in the ranges 90 to 100, 200 to 210 and the instance ids 2, 45, 10 and 15.

Next click  the instance id filter button.



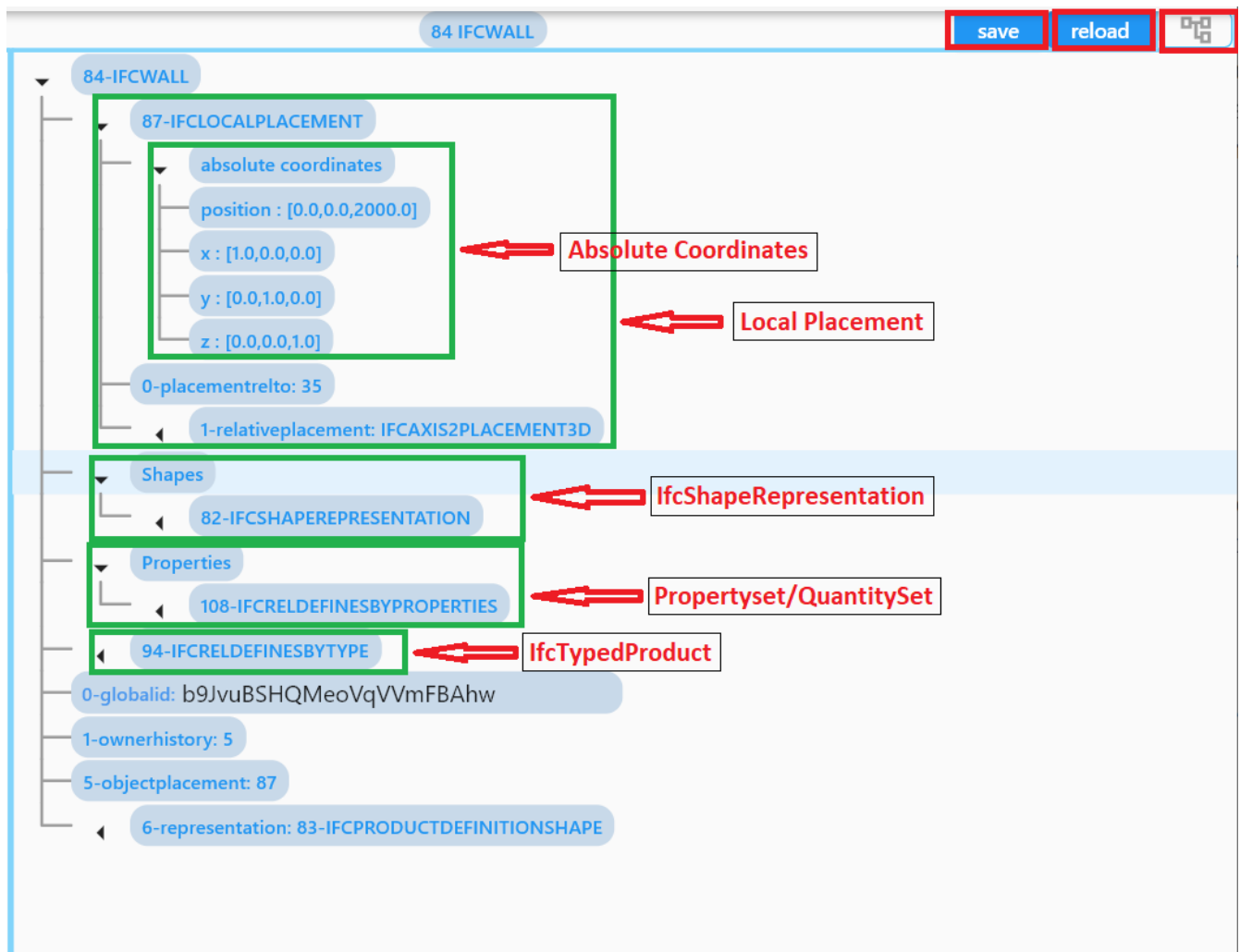
Each range is limited to return 1000.

5.4. Working with Instance Detail Tree

The STEP ISO-10303 standard set includes an object model specification using the EXPRESS language. According to ISO-10303-11: "EXPRESS data types are classified according to their nature as: simple data types, aggregation data types, constructed data types, named data types, and

generalized data types."


The instance detail tree is where PComposer users interact with "entity data type" instance.



Instance detail tree

The instance detail is a federated view when the detail is an entity instance. It could display the entity instance plus most of its components. The user has many levelers to control what gets exposed on the detail tree. The optional nodes include:

- placement
 - absolute coordinates
- IfcTypedProduct
- propertyset/quantityset
- IfcShapeRepresentation

Moreover, the  show detail toggle, allows users to turn on/off the optional attribute of an instance.



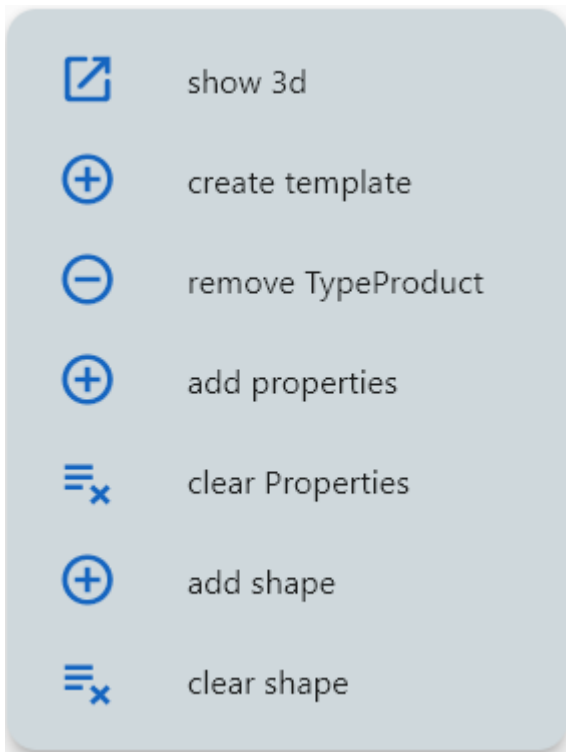
The  show detail toggle, is also part of user preference

List the spatial tree, user modify tree content by operating directly on the tree nodes.

5.4.1. Instance Root Node

Operations that directly affect the basic structure of the node or entity's component are anchored at the root node. The operations on the root node include:

- show 3d—show the instance in 3d. Available if instance has IfcShapeRepresentation
- add to layer—add instance to a layer, available if there is layer in the model and the instance is of type IfcShapeRepresentation
- clear layer—remove instance from all layer it is assigned to
- create template—create a private template from the instance.
- add shape—add a IfcShapeRepresentation component to instance
- clear shape—remove all shape. Also, if the reference count of any shape reach zero, it will be removed from model
- Add RepItem—add a IfcRepresentationItem to a IfcShapeRepresentation componenet.
- add shape to selected—add selected IfcShapeRepresentation instance to the spatial tree selected node. Current instance is the selected filtered instance.
- add type to selected—add IfcTypedProduct to the selected spatial tree node. The current instance is the selected IfcRelDefinedByType component in the instance filter.
- add type product—add IfcTyedProduct to instance
- remove type product—remove instance from IfcRelDefinedByType component
- add pset—add IfcRelDefinedByProperties to instance
- clear pset—remove all IfcRelDefinedByProperties from instance
- add pset to selected—add filter selected IfcRelDefinedByProperties component to the spatial selected node.
- add template shape to selected—add template shape to the spatial selected node.
- clone—clone the instance, available in instance filter
- delete—remove the instance, available in instance filter



Typical IfcProduct root node menu items

5.4.2. Component and Composite

A PCompose component is a self-contained entity instance package.

5.4.3. Working with Attributes

An Entity instance is a collection of attributes. Each attribute has a data type. The basic attribute data types are:

- number
- real
- integer
- string
- boolean
- logical
- binary
- aggregation—collection
- defined type
- entity
- enum
- select



Each attribute node tooltip shows its respective data type.



Defined type is an alias for some already existed underline type. For example IfcLogical is a LOGICAL type.

PIComposer divides attributes into two groups:

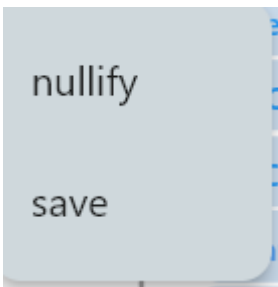
- simple—single value, requires only single node for user inter-action
- complex—multiple value, need a subtree to store its values

5.4.4. Simple Attributes

The simple type include:

- number
- real
- integer
- string
- boolean
- logical
- binary
- defined type, with simple underlining type
- enum

Numeric types, string and binary are stored in a text box. Boolean, logical, and enums are presented in a dropdown list. In all cases, user interaction is straight forward.



typcal attribute node command

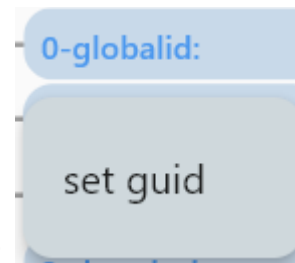


When an attribute has been modified, there are two ways to commit updates to data store. Right click on the label of the attribute to access the tree node menu and select the save menu item.



The save button on the right upper corner is a batch save button. It commits all changes to the data store all at once.

The defined type IfcGloballyUniqueId has string as its underlining type. Since the user might not have access to a UUID generator, PIComposer provides the set guid command. To set a guid



attribute, right click on the label of the attribute and select the command.

set guid

5.4.5. Complex Attributes

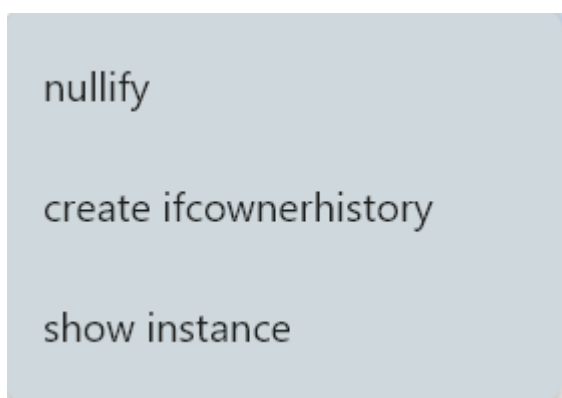
The complex attribute types are:

- entity
- select
- collection

A complex attribute usually requires multiple nodes to present its value. In particular an entity instance, when composed in situ, will occupy a full subtree.

Attribute node commands include:

- create—create an entity instance in place
- set reference—set attribute value as entity instance reference
- set reference selected—set spatial tree entity instance attribute value to reference the selected instance in instance filter.
- remove—remove instance or instance reference from a instance collection attribute
- show instance—show attribute instance in detail using instance filter.
- nullify—set optional attribute to null



typical entity attribute command

Just as there are two ways—[aggregation and composition](#)--to associate one object to another, there are two way to associate an attribute to a entity instance.

5.4.6. Instance Aggregation and Composition

Aggregation is the default instance to instance association in STEP standard, in fact, it is the only

option in ISO 10303 part21 standard.

To use the reference selected instance command, follow the steps below:

1. select the desired reference target in instance filter
2. right click on the entity attribute node label to activate popup menu
3. choose **reference selected instance** set reference selected menu item



Reference selected command will only be available if the selected instance is compatible to the Entity attribute. If attribute is optional, it must be null. To reset to a different reference, nullify first

To set reference without using instance filter, select the **set reference** set reference menu item and enter the target instance id on the dialog.

set reference

For a component, as a rule for non-shared attribute instance, the create command is the preferred method when populating entity attribute values. In the case of shared instance in a component, use the add composite command to create the shared instance, see [Component and Composite](#).

To create attribute instance in place, right click on the attribute label, then select **create ifcobject** command. If necessary fill in the detail in object creation dialog.



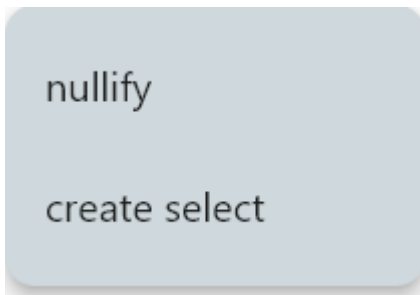
If the type to create has no subtype, attribute instance is directly created; no input from user is solicited.

5.4.7. Working With Select

A select entity attribute is a single value polymorphic container; it could hold value of any defined type within its specification.

There are three commands for select attributes:

- nullify—set optional select attribute to null value
- create—create the select object in place.
- remove—remove select from a collection of select

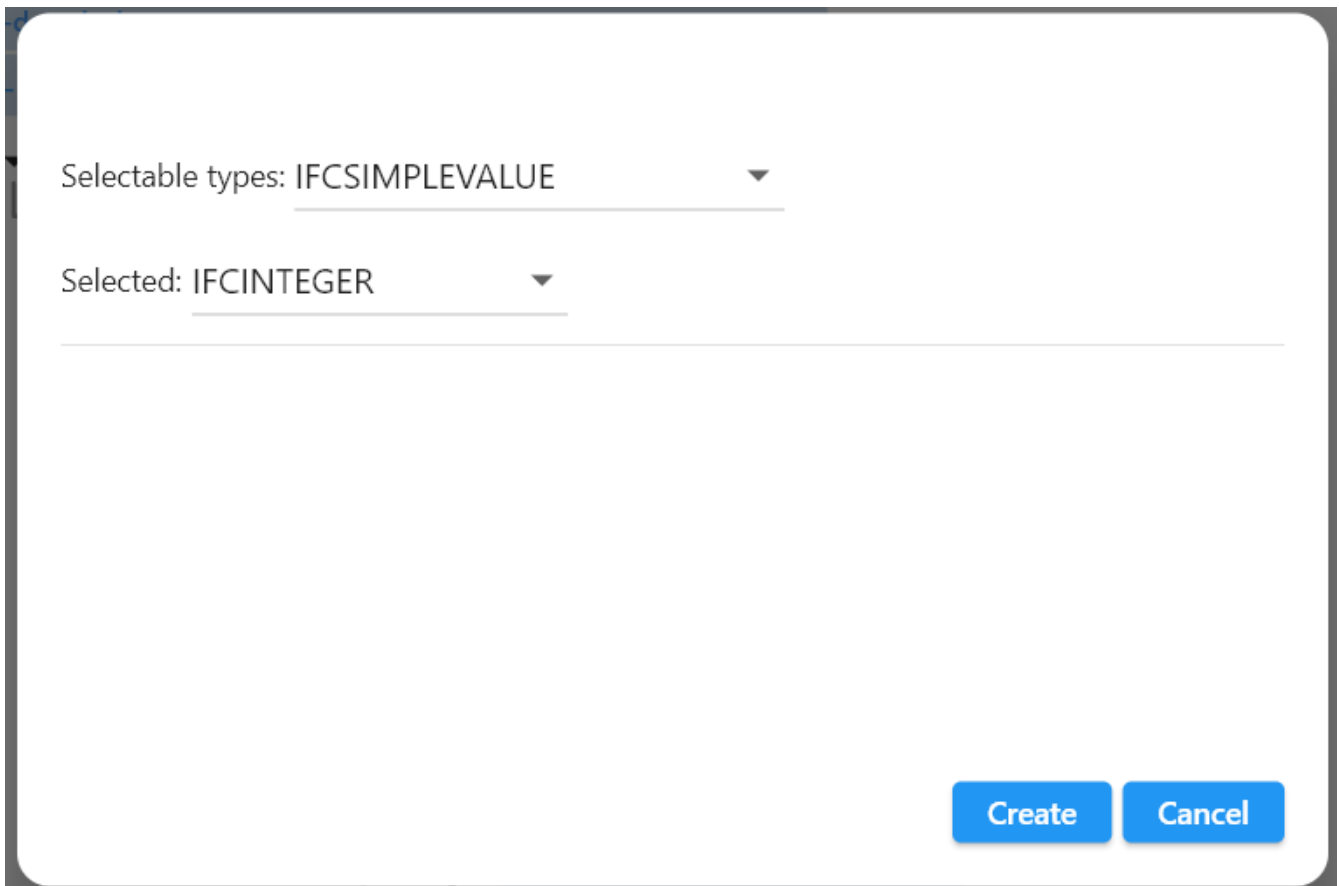


select attribute menu

The PCompose create select user interface is based on two key concepts:

- the selected type
- the value of the selected type

To create a select attribute value, select the  create select menu item. Fill in the detail in the create select dialog by first specifying the selected type, then value type.



Selectable types: IFCSIMPLEVALUE ▼

Selected: IFCINTEGER ▼

Create Cancel

example of select of a select attribute

5.4.8. Working with Collection

STEP aggregation data types are collections of defined types. Aggregations could contain order or unordered, unique or none-unique values. Aggregation containers include:

- array—indexed ordered collection
- list—ordered collection
- bag—unordered collection
- set—unordered and unique valued collection



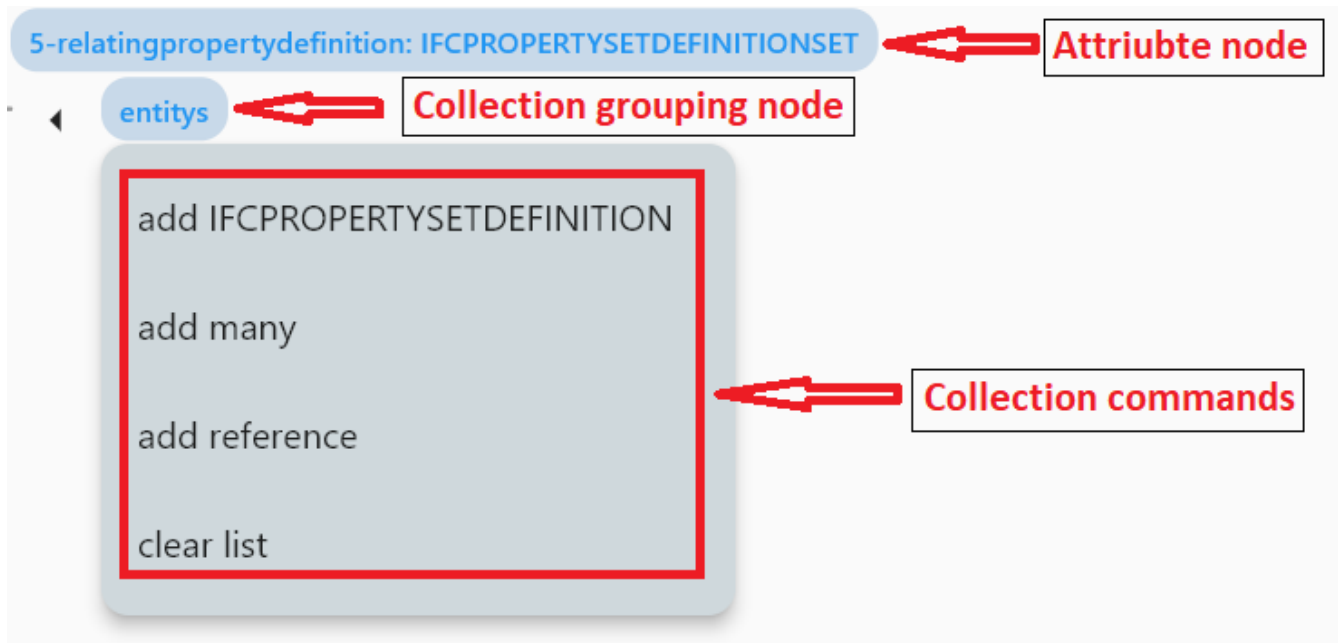
A collection such as a list could be restricted to contain unique values by using the key word UNIQUE. For example, the EXPRESS expression LIST [3:?] OF UNIQUE IfcCartesianPoint; means list of unique cartesian points.

PIComposer has a simplified collection data model; PIComposer models aggregation data types as vector of defined types. PIComposer supports multi-dimension aggregation.

PIComposer users use the following commands to work with collections:

- add—add an instance
- add many—add multiple instances
- add reference—available for entity container only
- clear list—empty the collection

In PComposer, a collect attribute has its own grouping node where collection commands are anchored.






Entity collect commands


Each add command activates a dialog box. To add multiple instances into a collection, right click on the collection, and select the **add many** add many command. .Entity collection, add many image::model-collection-attribute-add-many.png[]

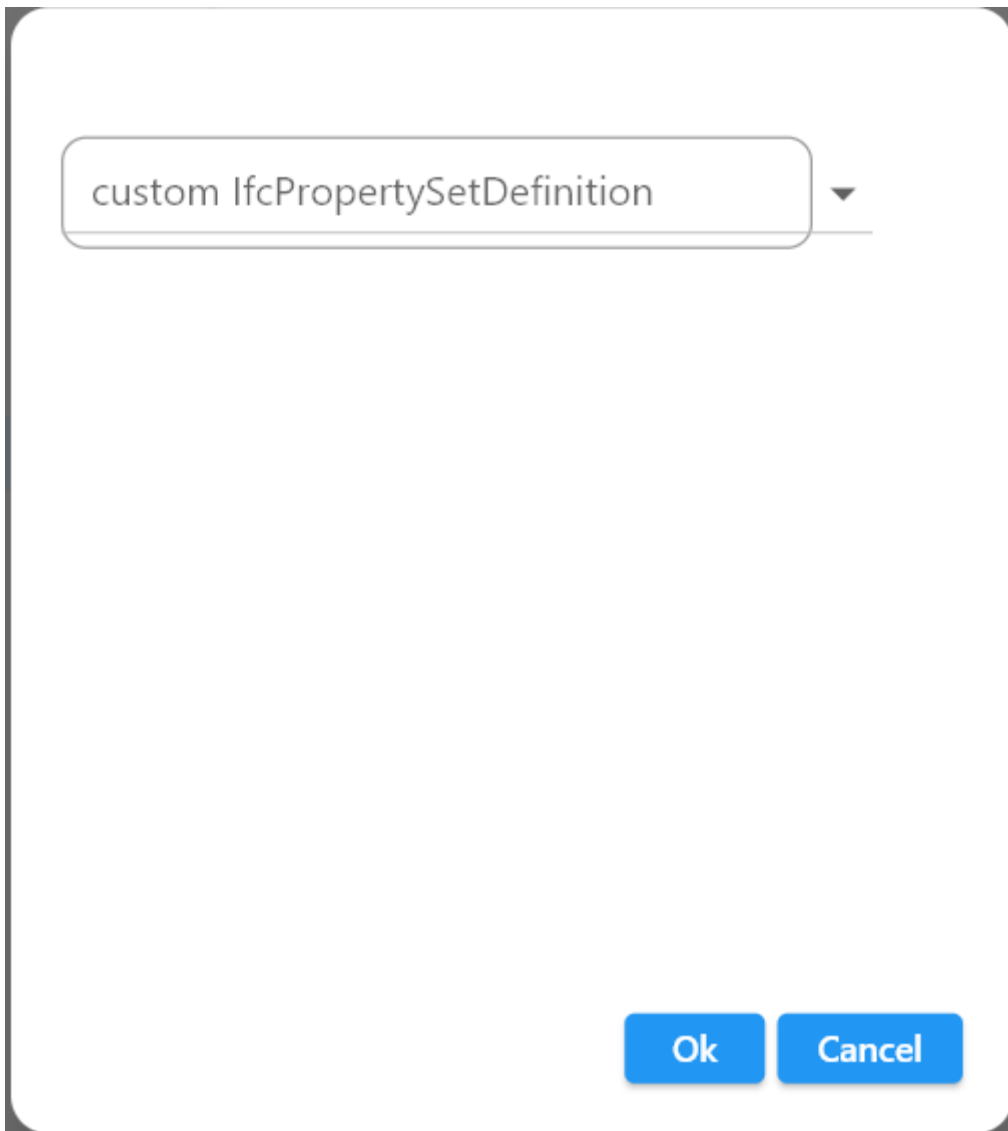
Complete the add many command by choosing the desired and entering instance count.

5.5. Working with Propertyset, QuantitySet and Their Templates

Propertyset and quantityset are packaged in a IfcRelDefinesProperties component. The preferred way to work with IfcRelDefinesProperties is using the following command on the root detail node:

- add propertyset—  **add propertyset** add propertyset command
- add selected propertyset—  **add selected propertyset** add selected propertyset.
- add propertyset to selected—  **add propertyset to selected** add propertyset to selected instance (from instance filter).

Executing the  **add propertyset** add propertyset command activates the create propertyset dialog. Choose the desired propertyset or quantityset and click OK to complete propertyset creation.



Create Property dialog



The dropdown in the above dialog lists all propertyset and quantityset known to PComposer which including those defined in the standard and those by templates.

Ifc4x3 documentation defines a large number (more than 600) of propertyset and quantityset outside of the schema. PComposer must know their content in order to instantiate them. To configure PComposer to create these and other user defined propertyset, PComposer uses template.

A template is defined using json file.

5.5.1. Propertyset Template

A propertyset is a collection of properties. Each property is an entity type. A property might have multiple attributes. We dive into propertyset template by looking at an abridged example:

```
{
  "__schema": "ifc4x3",
  "__type": "Pset_SlabCommon",
  "Reference" : {
```

```

        "property_type" : "P_SINGLEVALUE",
        "value_type" : "IfcIdentifier"
    },
    "Status" : {
        "property_type" : "P_ENUMERATEDVALUE",
        "value_type": "PEnum_ElementStatus"
    },
    "AcousticRating" : {
        "property_type" : "P_SINGLEVALUE",
        "value_type": "IfcLabel"
    },
    "PitchAngle" : {
        "property_type" : "P_SINGLEVALUE",
        "value_type": "IfcPlaneAngleMeasure"
    },
}

```

The first template two json properties are the headers:

- `__schema`—it must be ifc4x3
- `__type`—the propertyset type, it must be prefix with `Pset_XXX`

The remaining properties are property definition for the propertyset.

Each property has a name, a `property_type`, and `value_type`. In the example above for the first property we have:

- `name`—Reference
- `property_type`—`P_SINGLEVALUE`
- `value_type`—`IfcIdentifier`

`name` is the name of the property.

The value of `property_type` comes for the standard enum `IfcSimplePropertyTemplateTypeEnum`. The possible `property_type` values and their corresponding `IfcProperty` are:

- `P_SINGLEVALUE`—`IfcPropertySingleValue`
- `P_ENUMERATEDVALUE`—`IfcPropertyEnumeratedValue`
- `P_BOUNDEDVALUE`—`IfcPropertyBoundedValue`
- `P_LISTVALUE`—`IfcPropertyListValue`
- `P_TABLEVALUE`—`IfcPropertyListValue`
- `P_REFERENCEVALUE`—`IfcPropertyListValue`

Every `value_type` is a ifc defined type.



propertyset templates are stored in the folder
`□/picomposer_data/propertyset_template`



To create user defined propertyset at runtime without using templates, use the add propertysset command to create IfcPropertyset then add any IfcProperty.

5.5.2. Quantityset Template

A quantityset is a collection of quantity. A quantity is a subtype of IfcPhysicalSimpleQuantity that has a numeric attribute. Like other templates, quantityset template specification are json files. Below is a full example:

```
{
  "__schema": "ifc4x3",
  "__type": "Qto_BeamBaseQuantities",
  "Length": "Q_LENGTH",
  "CrossSectionArea": "Q_AREA",
  "OuterSurfaceArea": "Q_AREA",
  "GrossSurfaceArea": "Q_AREA",
  "NetSurfaceArea": "Q_AREA",
  "GrossVolume" : "Q_VOLUME",
  "NetVolume" : "Q_VOLUME",
  "GrossWeight" : "Q_WEIGHT",
  "NetWeight" : "Q_WEIGHT"
}
```

In the json file, first comes the header, which include the properties:

- `_schema`—must be ifc4x3
- `_type`—the quantityset name, must be prefixed with *Qto*

The remaining json properties are a list of quantities which has name and type. The possible types and their corresponding entity type are:

- `Q_LENGTH`—IfcQuantityLength
- `Q_AREA`—IfcQuantityArea
- `Q_VOLUME`—IfcQuantityVolume
- `Q_COUNT`—IfcQuantityCount
- `Q_WEIGHT`—IfcQuantityWeight
- `Q_TIME`—IfcQuantityTime



Quantityset templates are stored in the folder
📁/picomposer_data/quantityset_template



To create user defined quantitysets at runtime without using templates, use the add propertysset command to create IfcElementQuantity and manually add any IfcPhysicalSimpleQuantity subtype.

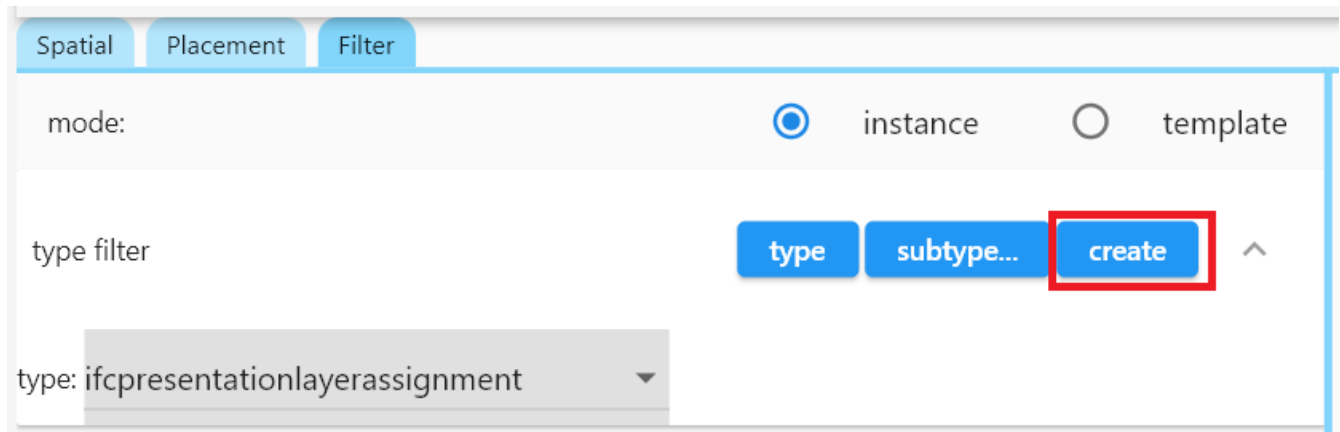
5.6. Working with layers

Layers are commonly used in CAD system for instance grouping and visibility control. The corresponding concept in ifc are embodied in the `IfcPresentationLayerAssignment` entity.



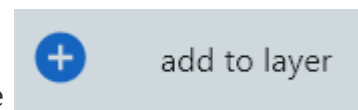
Only component of type `IfcShapeRepresentation` can be assigned to a layer in `PIComposer`. However, instances could be filtered by layer indirectly by its association to a shape, see [Layer Filter](#)

To setup a layer, first create a `IfcPresentationLayerAssignment` instance using the instance filter create command.



create command

To add layers to an `IfcShapeRepresentation` component, use the



add to

add layer dialog

image::model-add-to-layer-dialog.png


In the add layer dialog, select the desired layer from the dropdown list.

5.6.1. Layer Filter

A layer providing grouping of `IfcShapeRepresentation` component. A layer filter provides quick access to this grouping quickly. Optionally, instead of shape components, users may list the instances that are associated to the grouped shapes.



layer filter

To use the layer filter, select the layer from the dropdown list, and press the  filter by layer command.



The filtered result type is controlled by preference.



PIComposer does not show the layer filter if the model does not has any IfcPresentationLayerAssignment instance.

5.7. Ifc Exporting and 3d Viewer


To export models to ifc, click the  export model ifc model on the navigation bar.

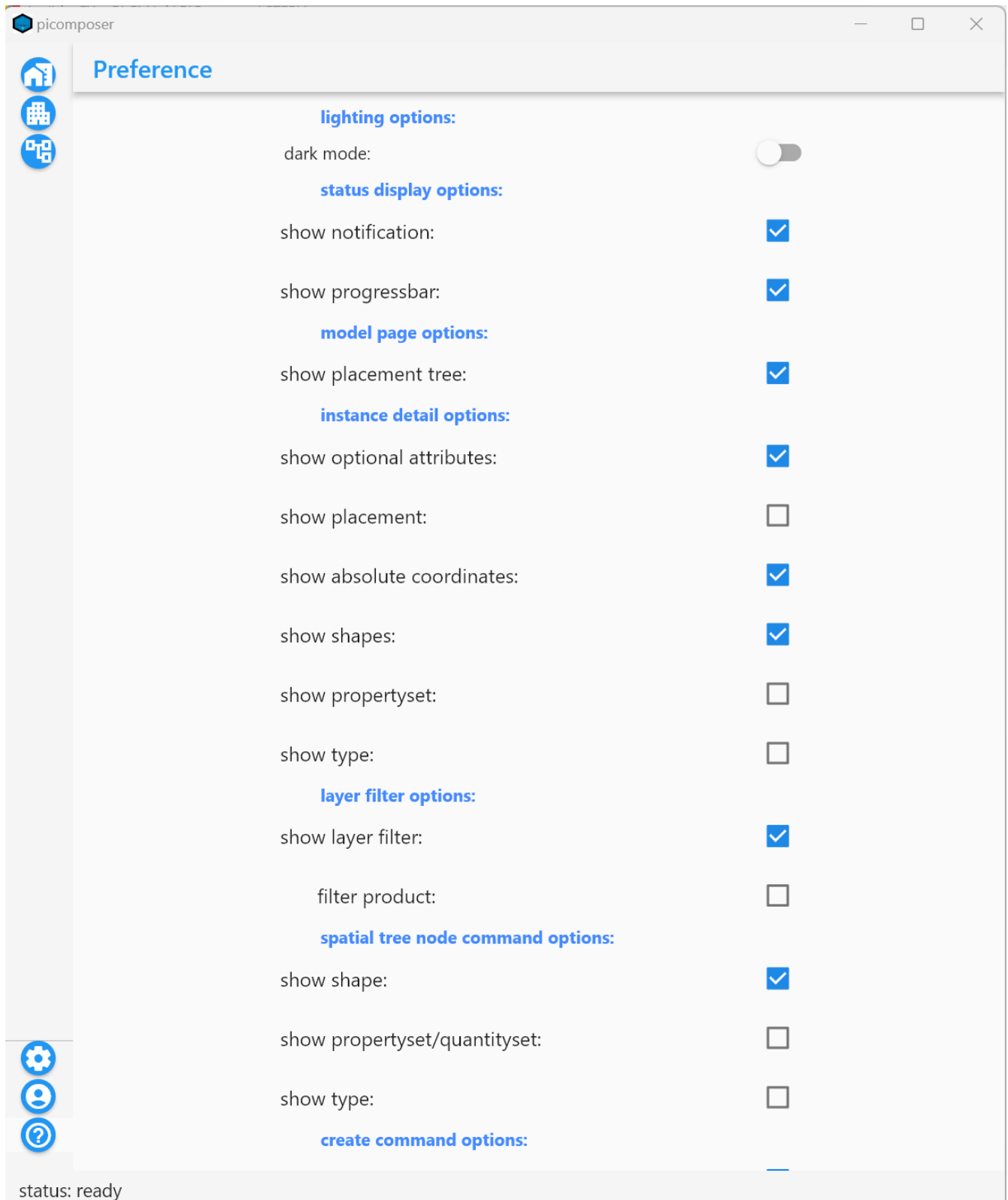
To view a model in 3d view, click show the  3d button on the navigation bar. This will kick start the export of the model to part21. When the export is completed, the web 3d viewer will be launched.



Both commands are background tasks. User may continue working in parallel while the command is running.

Chapter 6. Preferences

User interface customization is done using the preference pane. The preference setting options are grouped into major sections. Activate the preference pane by clicking the  button.

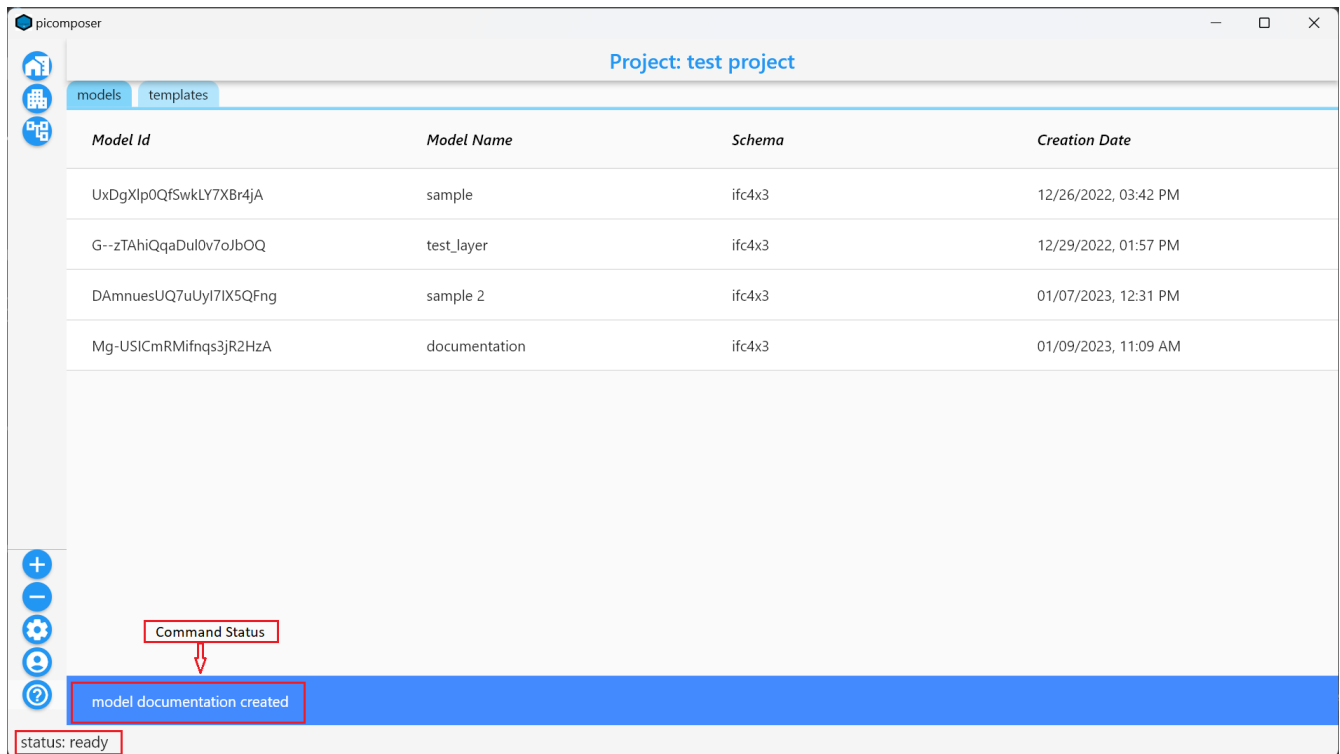


Preference Pane

6.1. Status Displays

Status display gives feedback of success or failure to user issued commands, and for long running commands, a progress bar. The options are:

- show notification—show notification when user command is completed (error message if completed in failure).
- show progress bar—show progress bar for long running command.

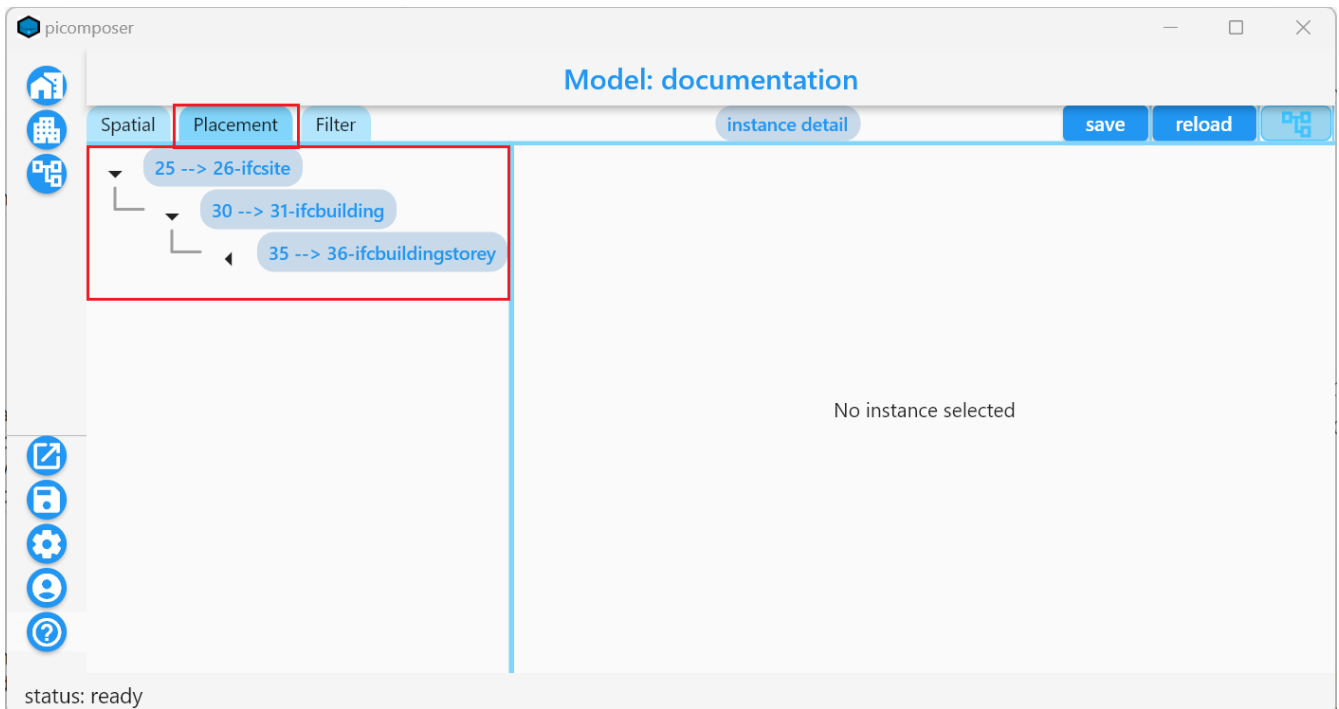


Command status

6.2. Model Page Options

The model page's main function is to display model structural information. PICOMPOSER displays two structural elements of a model: the logical object spatial hierarchy and the physical object placement hierarchy. To underscore placement's importance, it has its own dedicated optionally display as a tree.

- show placement tree—hide show placement tree in the model page.

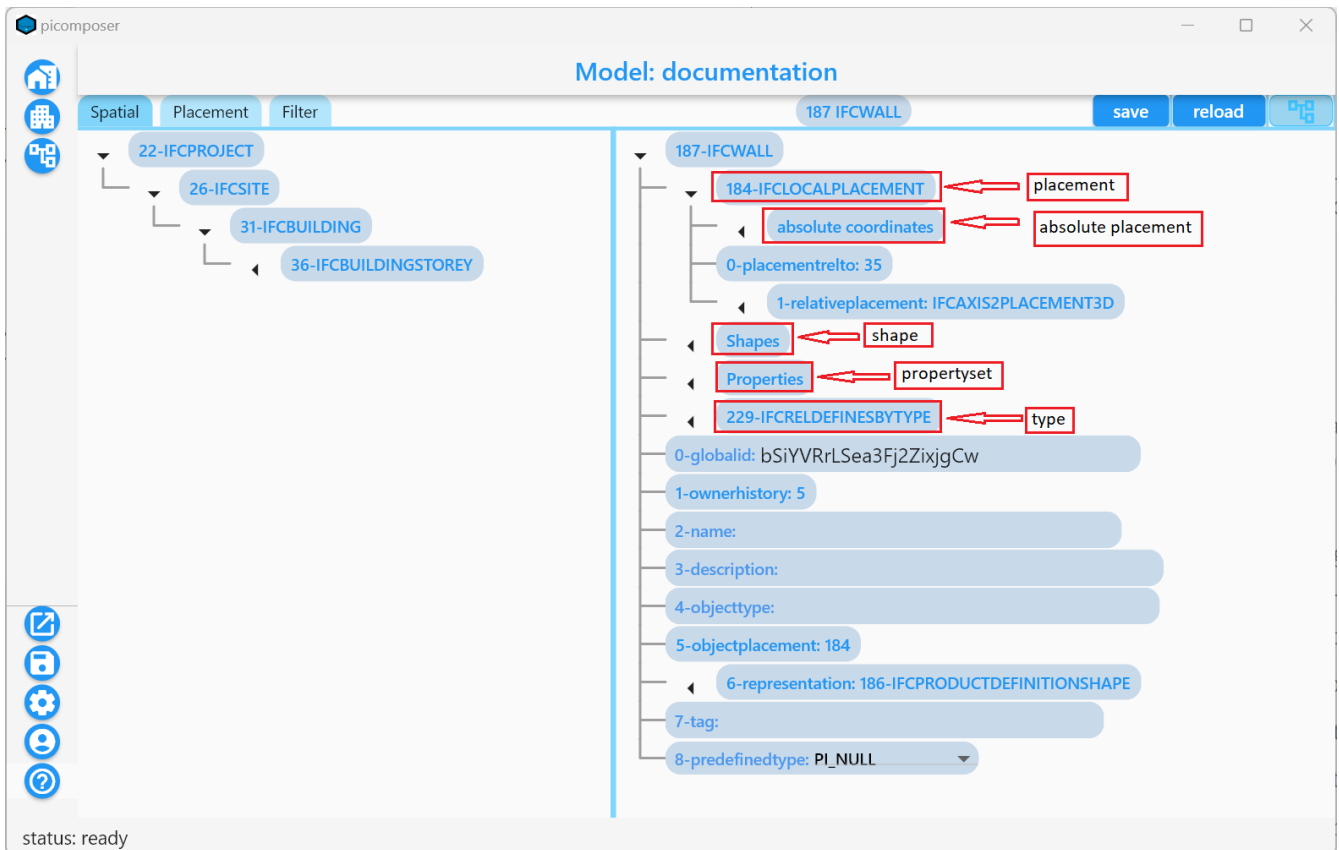


placement tree option on

6.3. Instance Detail options

Users most likely spend more time on the instance tree than any other part of PicoComposer's user interface. In addition to the instance attributes, the instance tree may optionally show its related components—shapes, propersets etc—allowing a user to manipulate an instance and its related component centrally. The options are:

- show optional attributes—show all attributes (including optional ones) in instance tree. Turn this off when it is desirable to create a valid instance quickly and make sure only required attributes are valid.
- show placement—option to show local placement component in instance tree.
- show absolute coordinate—show computed absolute coordinate.
- show shape—show IfcShapeRepresentation components in instance tree.
- show propertyset—show instance's propertyset/quantityset as IfcRelDefinesByProperties components.
- show type—show IfcRelDefinesByType component in instance tree.



Instance Detail

6.4. Layer Filter options

Layer filter is a object grouping mechanism useful in managing large CAD data sets and is a feature found on all main stream CAD system. PICOcomposer's layer filter system is configured by the layer filter options, and these options are:

- show layer filter—show layer filter when model contains at least one layer (IfcPresentationLayerAssignment).
 - show product associated to the layer assigned IfcShapeRepresentation.

6.5. Spatial Tree Node Command Options

PICOcomposer provides model navigation through the use the spatial tree. Moreover, the spatial tree provides context switching between an instance and its components through a series of command taking advantage of the instance filter. By isolating a component in the instance filter, a component could easily be shared among instances.

- show Shape detail—enable show shape command in the spatial tree. The command show the entity's IfcShapeRepresentation component in the detail tree.
- show propertyset detail—enable show propertyset command in the spatial tree. The command show the entity's IfcReldefinesproperties in the instance filter.
- show shape detail—enable show shape detail command on the spatial tree when entity has shape. This provides a short cut to share shape as mapped items among instances.
- show propertyset detail—enable show propertyset detail command on the spatial tree when

entity has propertyset.

- show type detail—enable show type detail command on the spatial tree when entity has IfcReldefinesByType

6.6. Create Command options

One way to create an instance in the model is using the create command in the filter tab in the model page. Some entity types correspond to component entity types. Thus it is convenient to create such entity using the component creation work flow. The following options activate just such a work flow:

- show shape create dialog—show the shape creation dialog when creating a IfcShapeRepresentation in the instance filter.
- show propertyset create dialog—show propertyset create dialog when create IfcRelDefinesProperties in the instance filter
- show type create dialog—show type create dialog when creating IfcRelDefinesByTypes in the instance filter
- show create style dialog—show type create dialog when creating IfcStyledItem

6.7. Private Template Root Node Command options

Options in this section enable commands for authoring instance templates. These are:

- copy attribute path—enable copy attribute path command in the template filter where the template type is private
- copy template id—enable copy template id command in the template filter where the template type is private
- guid generator—enable clipboard guid generation command




6.8. login options:

Login options toggles the persistents of user login information.

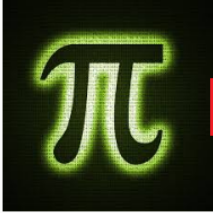
- remember email—save user email and use it for the next login attempt
- remember password—save user password and use it for the next login attempt

Chapter 7. User Account

The User Account page summarized a user's account information as it is known to PComposer. Users sign out of current PComposer sessions and update user account information in the user account pane.



Account



Upload

Email

chi.wah.ng2006@gmail.com

Phone Number

424-703-8115

User Name



Chi W. Ng

Website

www.pisys.io

Update

Sign Out



Status: Ready

User Account

There are three commands:

- update—update user information
- upload—upload user avatar
- sign out—sign out of current user session