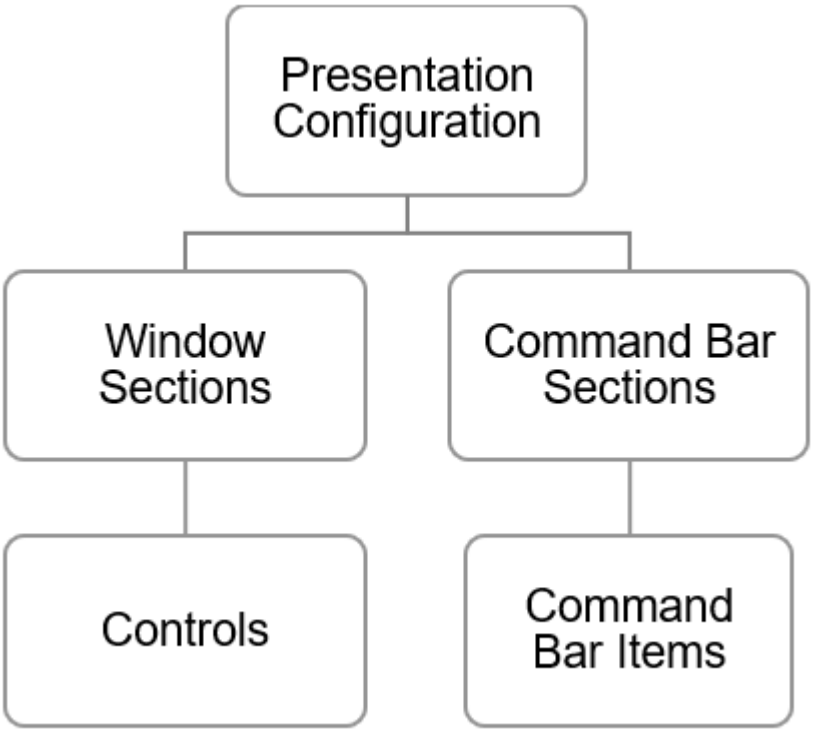
Configurable User Interface

This section describes the underlying technical details of configuring the Aras Innovator web client. For hands-on instructions, see Section 4.

Understanding the CUI Data Model

The Configurable User Interface, or CUI, is a modeling mechanism in Aras Innovator that allows administrators to define the layout and behavior of a client application. Standard features like the table of contents (TOC), sidebars, toolbars, menus, keyboard shortcuts, and accordion sections within item views are modeled with CUI controls.

A simplified diagram of the CUI data model



Presentation Configurations

Presentation Configurations serve as containers for CUI configurations, defining the scope of the related CUI items as either *global* or *ItemType-specific*. A *global* Presentation Configuration is associated with a specific client via the “presentation” item property on a Client Presentation item. In a standard Aras Innovator database, there is a single Client Presentation item to identify the global Presentation Configuration that defines the layout and UI elements that are inherited throughout the Aras Innovator web client.

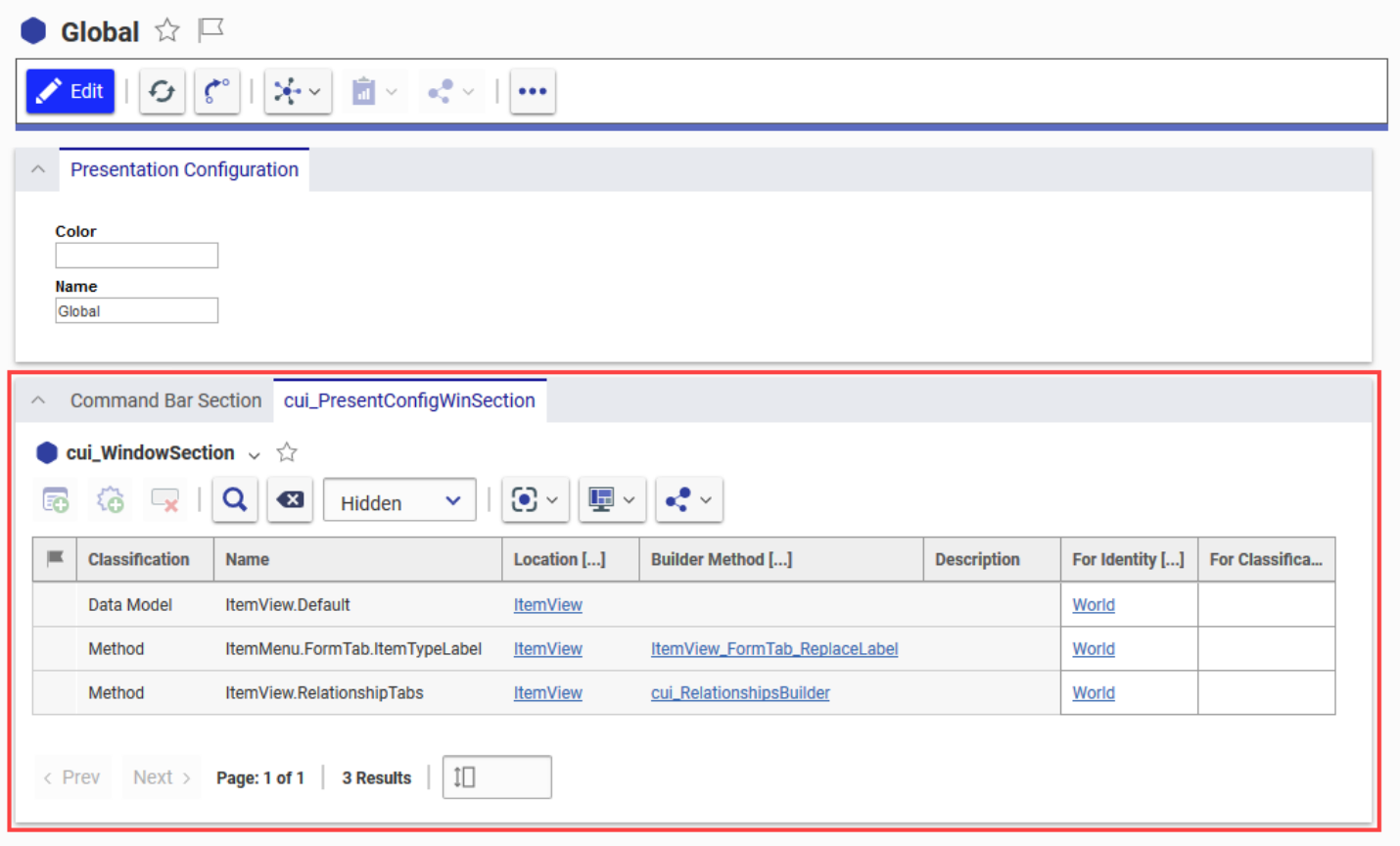
Important

The CUI data model was architected with the intent that the user interface of any client application could be defined in the Aras Innovator database. That’s why there’s only one Client Presentation item in an out of the box Aras Innovator database – only the Aras Innovator web client is defined by default. However, an admin could create another Client Presentation to define the UI of a custom client application or connector.

The scope of an *ItemType-specific* Presentation Configuration is determined by relating it to an ItemType via the Client Style relationship tab. These Presentation Configurations allow administrators to override or augment the global configuration for a specific ItemType without affecting the UI of every ItemType. As of Aras Innovator version 12.0, ItemType-specific Presentation Configurations are also used in conjunction with an ItemType’s TOC Access relationship(s) to determine where – and whether – the ItemType is displayed in the table of contents (TOC). Admins don’t need to manually create the CUI items that display ItemTypes in the TOC; however, they will need to package the CUI configuration for any custom ItemTypes.

Window Sections

Window Sections are used to define the layout of a client application screen. Because they’re related to Presentation Configuration items, Window Sections can be inherited globally or defined for an ItemType-specific scope to create different layouts. If we look at the Window Sections related to the global Presentation Configuration in a standard Aras Innovator database, we can see examples of the two ways to define a Window Section – *declaratively* and *dynamically*.

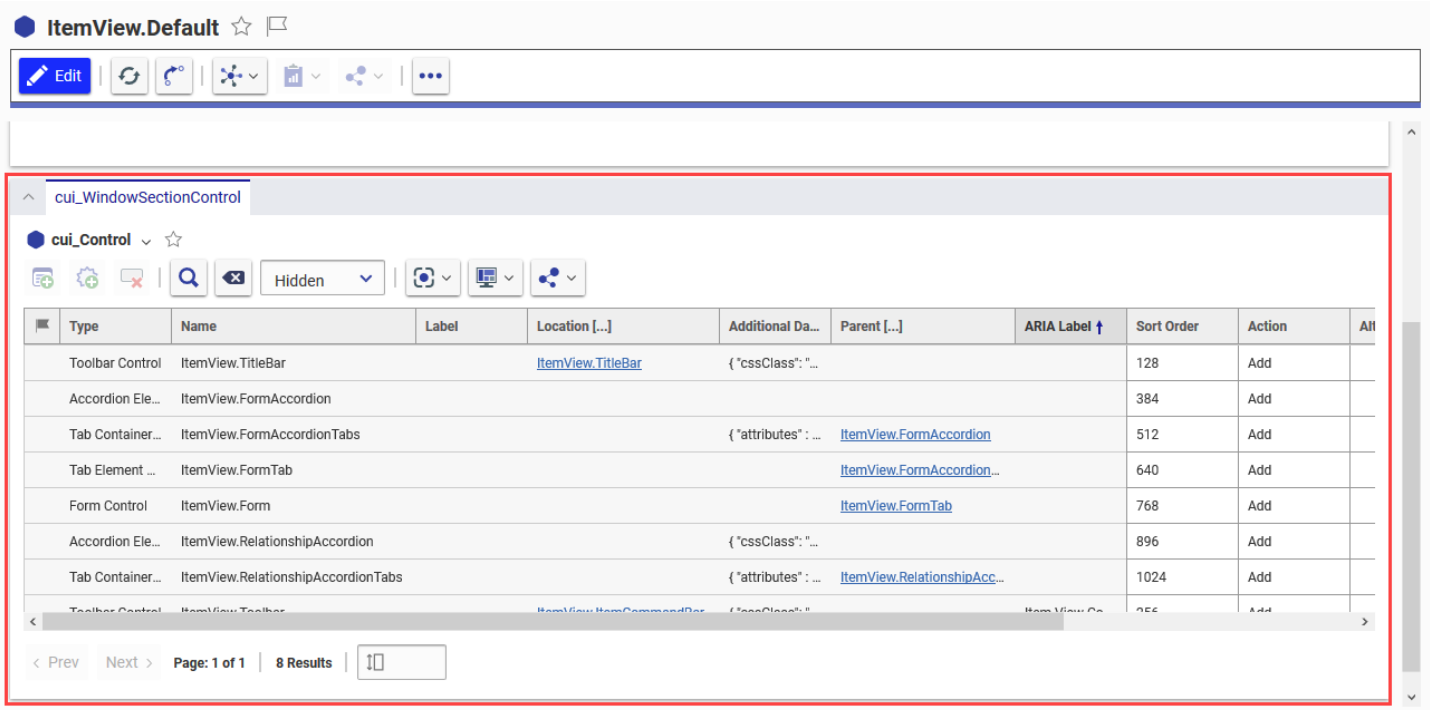


Globally scoped Window Sections

Declaratively defined Window Sections are configured by an administrator at “design time” and are identified with the *Data Model* classification. At runtime, the client application retrieves the Window Section and its child Controls to determine how the UI should be displayed. This approach is useful for defining client screens (or areas of client screens) when the desired layout is known in advance. On the other hand, *dynamically-defined* Window Sections are populated by a Builder Method at runtime – hence the *Method* classification. This approach is often used for making minor adjustments, like updating the label displayed on a declaratively defined element, or completely populating a section based on the context item, as in the case of the default ItemView.RelationshipTabs Window Section.

Controls

Controls define the layout of the client UI within a parent Window Section. Using the Action property on the cui_WindowSectionControl, admins can *add*, *remove*, *replace*, or *clear all* Controls from a Window Section.



Example Window Section and Controls See the following list for more details about the different types of Controls. Different classifications will support different relationship properties and different options in the Additional Data property.

Toolbar Control

The *Toolbar Control* type indicates where a command bar may be rendered in the client UI. Use the Location property on the Control item to identify the Location that the associated CommandBarSection item should also use. The Additional Data property optionally supports *cssClass* and *attributes* properties. See the default *ItemView.TitleBar* and *ItemView.Toolbar* items for examples.

Accordion Element Control

The *Accordion Element Control* type indicates where a collapsible accordion element may be rendered. An Accordion Element Control may be identified as the parent of one or more other

controls.

The Additional Data property optionally supports the *cssClass* and *attributes* properties.
Use *attributes* to set the accordion's default display state when the item view opens:

- *collapsed*: the accordion is collapsed and can be expanded by clicking the arrow at the start of the accordion.
- *maximized*: the accordion expands over the form. The item's title bar and toolbar remain visible; the main item form is hidden. A Restore button returns the accordion to its default state.
- If neither attribute is present, the accordion displays expanded, which is the default behavior.

```
{ "attributes": { "maximized": true } }
```

Replace *maximized* with *collapsed* to apply the other non-default state.
If more than one accordion on the same item view is set to *maximized*, the accordions stack in reverse configuration order.
An accordion-specific *attributes* setting takes precedence over the ItemType's Default Structure View setting when both apply to the same accordion.
See the default *ItemView.RelationshipAccordion* item for *cssClass* examples and Set a Default State for Accordion Sections for *attributes* examples.

Tab Container Control

The *Tab Container Control* type indicates where a group of Tab Element Controls may be rendered. The end user can click through the tabs in a single container, alternating between the content of the child Tab Element Controls. Use the Parent property on the cui_WindowSectionControl relationship to identify the Accordion Element Control the Tab Container Control should appear in. The Additional Data property optionally supports the *cssClass* property. See the default *ItemView.RelationshipAccordionTabs* item for examples.

Tab Element Control

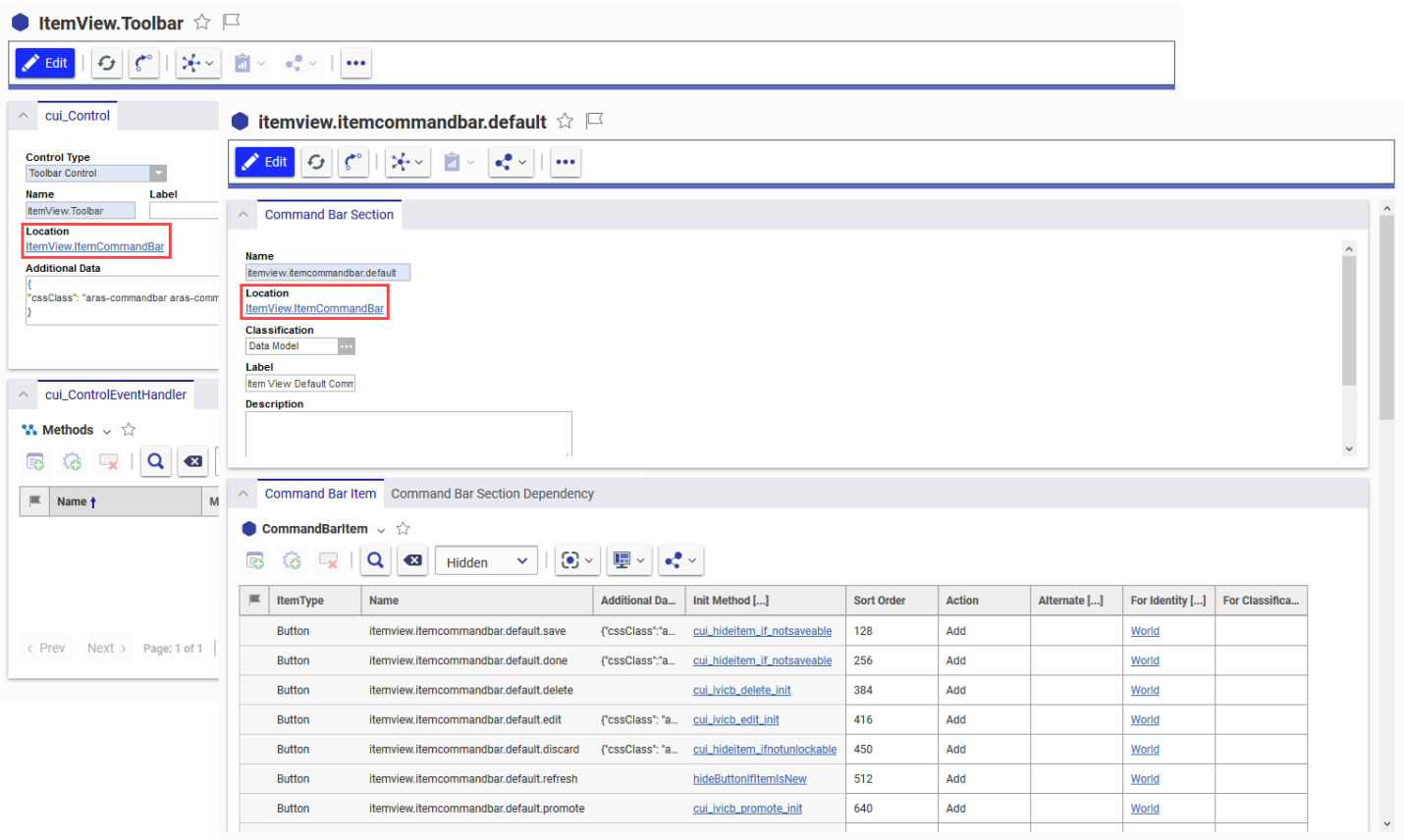
The *Tab Element Control* type indicates where a single tab may be rendered. A Tab Element Control may be identified as the parent of one or more other controls. Use the Parent property on the cui_WindowSectionControl relationship to identify the Tab Container Control the Tab Element Control should appear in.

Form Control

The *Form Control* type indicates that a Form definition may be rendered in the client UI. Use the Parent property on the cui_WindowSectionControl relationship to identify the Tab Element Control the Form content should appear in.

Command Bar Sections

On the surface, Window Sections and Command Bar Sections appear very similar – they're both related to Presentation Configurations, they can be defined either declaratively or dynamically, and they have many of the same properties. However, the two types serve different functions. While Window Sections and Controls define the *layout* of the client application screens, Command Bar Sections define the *content*.

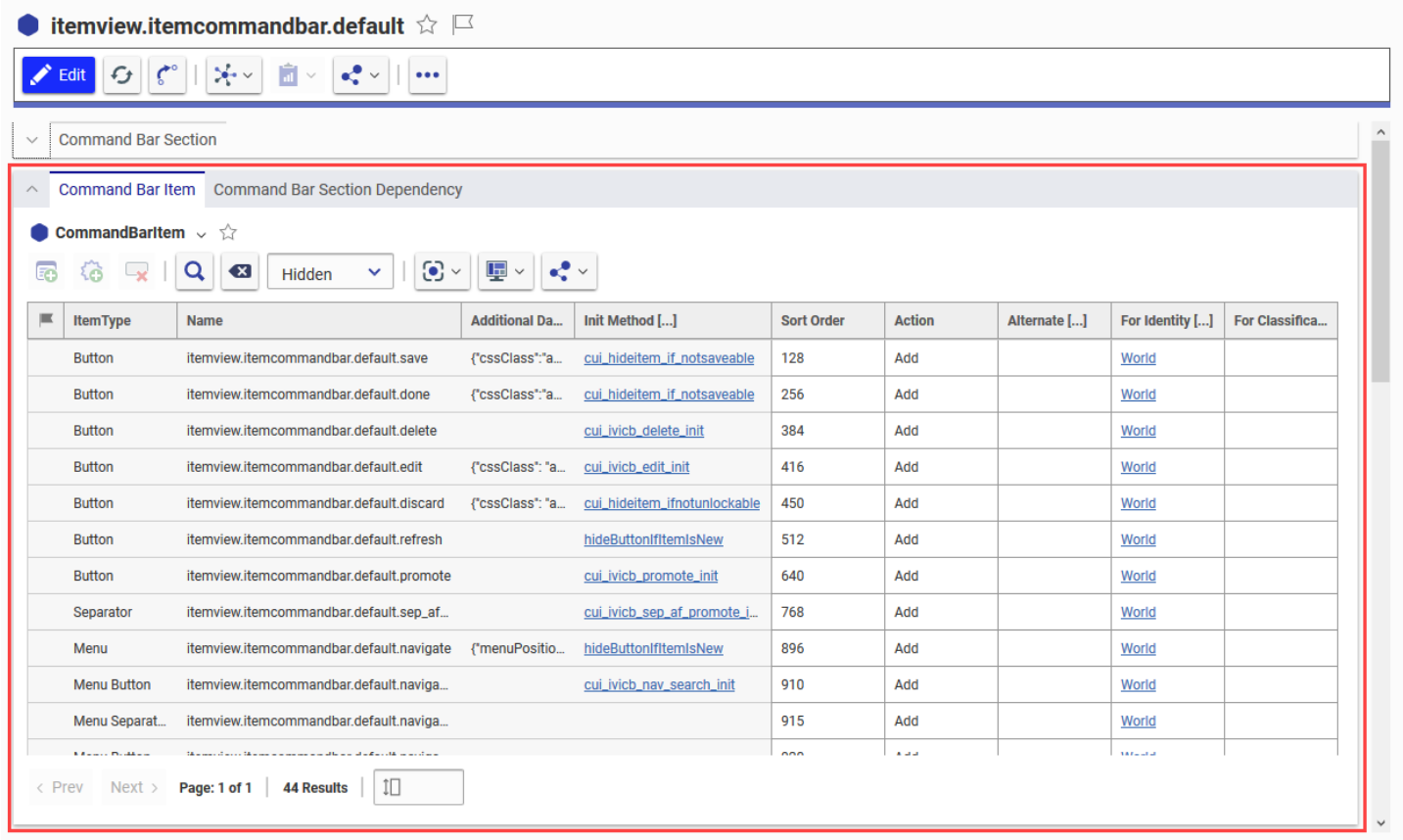


ItemView Toolbar Control and corresponding Command Bar Section Consider this example based on the standard global CUI configuration. The *ItemView.Default* Window Section declares that the Aras

Innovator web client has an “ItemView” screen, and one of the related Controls indicates that the ItemView has a toolbar element with the *ItemView.ItemCommandBar* Location. To determine the content of the layout defined by the Window Section and its Controls, we just need to find the Command Bar Section with the same Location and check out its related Command Bar Items.

Command Bar Items

So far, we’ve defined the *layout* of the client UI (Window Sections and Controls) and we’ve defined the *content* of the UI (Command Bar Sections and their relationships to Command Bar Items), but we haven’t defined the *behavior* of the UI. That’s where Command Bar Items differ from the rest of the data model we’ve reviewed so far – they provide the ability to execute logic based on user interaction, client state, and item context in addition to the add, remove, replace, and clear all actions on the relationship.



Default ItemView Command Bar Items Command Bar Items are also implemented differently. While Window Sections, Controls, and Command Bar Sections all used classifications to differentiate items,

the CommandBarItem ItemType is implemented as a polymorphic ItemType. See the list in the following section for more details on the various ItemTypes that are implemented as sources of the CommandBarItem poly ItemType.

Button

A *CommandBarButton* item represents an element that triggers an action. It can be single state, where a click executes the same action every time, or it can have two states – active and inactive. Buttons support multilingual labels, multilingual tooltips, images, and the following properties:

- *Additional Data:* supports*cssClass*
- *Init Method:* a Method, often used to show, hide, enable, or disable an element when initialized
- *Click Method:* a Method used to execute some logic when the element is clicked by a user
- *Include Events:* state change events that should trigger the item's initialization method

See the default *itemview.itemcommandbar.default.save* item for an example.

Checkbox

A *CommandBarCheckbox* represents a single, atomic item in any non-menu container that can be toggled on/off. Can be styled as a checkbox or radio button. Checkboxes support multilingual labels, multilingual tooltips, and the following properties:

- *Additional Data:* supports custom parameters
- *Init Method:* a Method, often used to show, hide, enable, or disable an element when initialized
- *Click Method:* a Method used to execute some logic when the element is clicked by a user
- *Include Events:* state change events that should trigger the item's initialization method

Dropdown

A *CommandBarDropDown* item represents a combobox element. The element's options are populated from the Additional Data property merged with the result of the Init Method. Dropdowns support multilingual labels, multilingual tooltips, images, and the following properties:

- *Additional Data:* supports*cui_items*, an array of objects each containing an id and name or label
- *Init Method:* a Method, often used to show, hide, enable, or disable an element when initialized
- *Click Method:* a Method used to execute some logic when the element is clicked by a user
- *Include Events:* state change events that should trigger the item's initialization method

Important

If the Init Method returns false or an empty array, the dropdown control will not be created.

Separator

A *CommandBarSeparator* represents a divider to visually separate items in any non-menu container. Separators support the following properties:

- *Additional Data:* supports custom parameters
- *Init Method:* a Method, often used to show, hide, enable, or disable an element when initialized
- *Include Events:* state change events that should trigger the item's initialization method
  See the default *itemview.itemcommandbar.default.sep_af_promote*item for an example.

Menu

A *CommandBarMenu* represents a hierarchical menu that can contain CommandBarMenuButton items or other CommandBarMenu items. Menus support multilingual labels, multilingual tooltips, images, and the following properties:

- *Parent Menu:* the parent CommandBarMenu item if this item is a sub-menu
- *Additional Data:* supports *menuPosition*
- *Init Method:* a Method, often used to show, hide, enable, or disable an element when initialized
- *Click Method:* a Method used to execute some logic when the element is clicked by a user
- *Include Events:* state change events that should trigger the item's initialization method.
  See the default *itemview.itemcommandbar.default.navigate*item for an example.

Menu Button

A *CommandBarMenuButton* represents a single, atomic button in a menu with some associated logic. Menu Buttons support multilingual labels, multilingual tooltips, images, and the following properties:

- *Parent Menu:* the parent CommandBarMenu item
- *Additional Data:* supports custom parameters
- *Init Method:* a Method, often used to show, hide, enable, or disable an element when initialized
- *Click Method:* a Method used to execute some logic when the element is clicked by a user

- *Include Events:* state change events that should trigger the item's initialization method
  See the default *itemview.itemcommandbar.default.navigate.structurebrowser*item for an example.

Menu Separator

A *CommandBarMenuSeparator* represents a divider to visually separate items in a menu. Menu Separators support the following properties:

- *Parent Menu:* the parent CommandBarMenu item
- *Additional Data:* supports custom parameters
- *Init Method:* a Method, often used to show, hide, enable, or disable an element when initialized
- *Include Events:* state change events that should trigger the item's initialization method.
  See the default *itemview.itemcommandbar.default.navigate.aftersearch*item for an example.

Menu Checkbox

A *CommandBarMenuCheckbox* represents a single, atomic item in a menu that can be toggled on/off. Can be styled as a checkbox or radio button. Menu Checkboxes support multilingual labels, multilingual tooltips, and the following properties:

- *Parent Menu:* the parent CommandBarMenu item
- *Additional Data:* supports custom parameters
- *Init Method:* a Method, often used to show, hide, enable, or disable an element when initialized
- *Click Method:* a Method used to execute some logic when the element is clicked by a user
- *Include Events:* state change events that should trigger the item's initialization method

A *CommandBarShortcut* item represents a keyboard shortcut that triggers an action. It can only be used with specific Locations intended for shortcuts. Shortcuts support the following properties:

- *Additional Data:* supports*stopPropagation*, *useCapture*, *preventDefault*, *preventBlur*, and *context*
  - *stopPropagation:* Boolean property, prevents further propagation of the current event
  - *useCapture:* Boolean property, set to true if the "this" pointer must point to the context item
  - *preventDefault:* Boolean property, cancels the default event if it's cancelable, without stopping further propagation

    - *preventBlur:* Boolean property, prevents the blur event if set to true and a domNode has the focus
    - *context:* Object property, the “this” pointer references this object if useCapture is true
- *Click Method:* a Method used to execute some logic when the shortcut is keyed in by a user
- *Shortcut:* the key combination for triggering the shortcut

This table lists valid shortcut combinations:
Valid shortcuts

Ctrl+	Shift+	Ctrl+Shift+	Button on keyboard
Supported	Supported	Supported	a-z
Supported	Supported	Supported	0-9
Supported	Supported	Supported	Num0-Num9
Supported	Supported	Supported	~
Supported	Supported	Supported	tab
Supported	Supported	Supported	enter
Supported	Supported	Supported	insert
Supported	Supported	Supported	delete
Supported	N/A	N/A	pageup, pagedown, home, end
Supported	N/A	N/A	F1-F12
Supported	N/A	N/A	arrows, Num-arrows
Supported	N/A	N/A	<, >
Supported	N/A	N/A	[,]
Supported	N/A	N/A	-, +, Num-, Num+

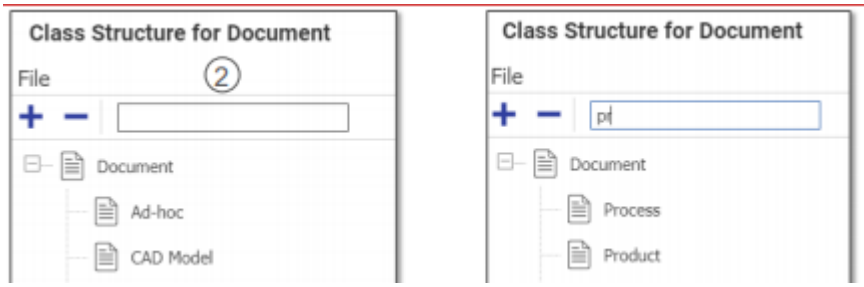
See the default com.aras.innovator.cui_default.mws_delete item for an example.

Edit

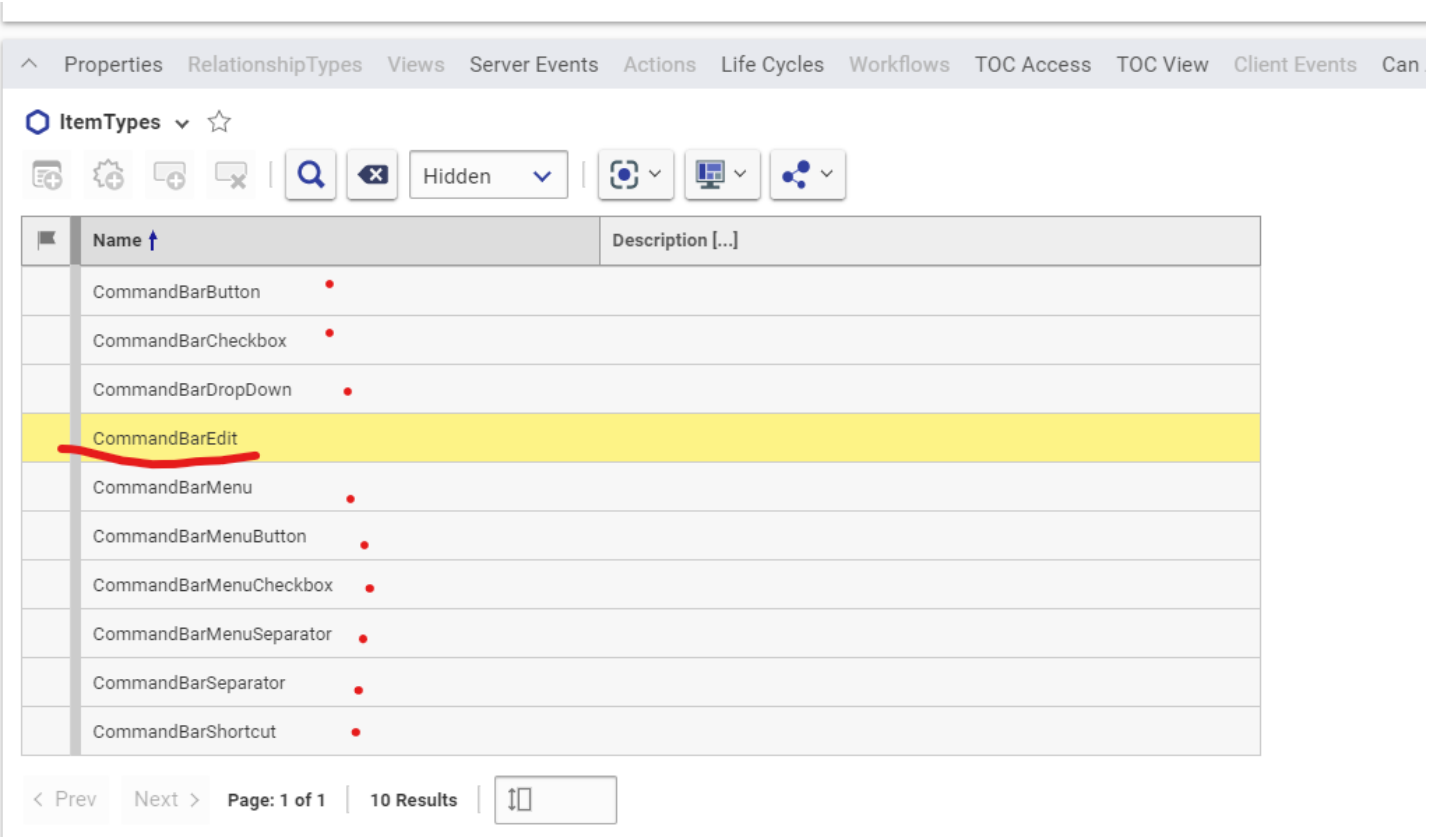
The Edit ItemType enables you to create a text box control and use it in a Command Bar to capture data relevant to a particular command. The Edit option supports multilingual labels, multilingual tooltips, and the following properties:

- *Additional Data:* supports custom parameters
- *On Keydown Method:* a Method used to execute some logic when the shortcut is keyed in by a user

- *Init Handler:* a Method, often used to show, hide, enable, or disable an element when initialized The following screenshot shows an example using the Class Structure dialog box:



In this example, entering “pr” in the search box displays only those documents that start with the letters “pr” enabling you to create a class structure for documents. The Edit ItemType enables you to create a text box control which can be placed in a command bar to capture data relevant to a command. The following figure shows an example using the CommandBarEdit ItemType.



Common CUI Properties

The following properties occur on most or all CUI ItemTypes.

Location

The *Location* property indicates the intended location or purpose of the CUI item. In Aras Innovator 11.0, this property was based on a List of predefined values. As part of the UX/UI enhancements included in Aras Innovator 12.0, the property was changed to an Item property with data source cui_Location. This update improves the ability for admins to define their own custom locations for client applications.

For Identity

The *For Identity* property indicates that the members of the identity will be affected by the CUI configuration item. For example, if a Command Bar Section has a relationship to a Button with the *Add* action and the For Identity property is set to *World*, all users will see the button. If a Command Bar Section has another relationship to the same Button with the *Remove* action and the For Identity property is set to *All Suppliers*, all users **except** members of the All Suppliers identity will see the button.

For Classification

The *For Classification* property indicates that the CUI configuration item will be evaluated when the context item has the specified classification. For example, if a Command Bar Section has a relationship to a Button with the *Add* action and the For Classification property is set to *Component*, all users will see the button only when the context item has the Component classification.