Auxiliary Schema for Preferences Framework

On This Page

An Auxiliary Schema is a JSON document that defines the information needed to build a preferences editor interface, including

- what component(s) to use to render the preference panels,
- where to find HTML templates and string bundles,
- what component(s) to use to act on preference settings.

An auxiliary schema must contain some required properties, described below. In addition to these properties, developers are free to include any other properties their implementation may require.

Properties

Top-level properties

- `namespace` (optional; recommended)
  - the namespace of the component to call to initialize the constructed grades
- `messagePrefix`
  - defines the path to the directory containing the messages
- `message`
  - the path to the message bundle for the prefs editor itself
  - use "%prefix" to use the prefix specified by "messagePrefix" as part of the path
- `templatePrefix`
  - defines the path to the directory containing the templates
- `template`
  - the path to the template for the prefs editor itself
  - use "%prefix" to use the prefix specified by "templatePrefix" as part of the path

Preference block properties

Preference blocks can be given any property name, so long as the name is unique within the schema. Preference blocks will have the following properties:

- `type`
  - used to define the preference setting type
  - must match the string defined by the Primary Schema
- `panel`
  - specifies the configuration for the panel component
  - each preference block can specify only one panel
- `enactor`
  - specifies the configuration for the enactor component
  - each preference block can specify only one enactor

Panel properties

- `type`
  - used to identify the component to use
- `container`
  - the DOM element to use as a container for the component
  - only used in the `"panel"` block
- `template`
  - the path to the template for the panel
  - use `%prefix` to use the prefix specified by `templatePrefix` as part of the path
- `message`
  - the path to the message bundle for the panel
  - use `%prefix` to use the prefix specified by `messagePrefix` as part of the path

Enactor properties

- `type`
    - used to identify the component to use

Composite Panel properties

- `groups`
    - contains named composite panel blocks, similar to panel definitions described above
    - contains special `panels` property listing names of subpanels to include

For detailed information about how to work with composite panels, see Composite Panels.

Example Auxiliary Schema

```
{
    // The author of the auxiliary schema will provide the namespace, which will be used
    //for the component to call to initialize the constructed grades.
    "namespace": "fluid.prefs.constructed",

    // The common path to settings panel templates.
    // The template defined in "panels" element will take precedence over this definition.
    "templatePrefix": "../../../framework/preferences/html/",

    // The path to the preferences editor own template (e.g. the separated panel prefs editor template)
    "template": "%prefix/SeparatedPanelPrefsEditor.html",

    // The common path to settings panel messages.
    // The message defined in "panels" element will take precedence over this definition.
    "messagePrefix": "../../../framework/preferences/messages/",

    "message": "%prefix/prefsEditor.json",
    "textSize": {
        "type": "fluid.prefs.textSize",
        "enactor": {
            "type": "fluid.prefs.enactors.textSize"
        },
        "panel": {
            "type": "fluid.prefs.panels.textSize",
            "container": ".flc-prefsEditor-text-size",
            "template": "%prefix/PrefsEditorTemplate-textSize.html",
            "message": "%prefix/textSize.json"
        }
    },
    "lineSpace": {
        "type": "fluid.prefs.lineSpace",
        "enactor": {
            "type": "fluid.prefs.enactors.lineSpace",
            "fontSizeMap": {
                "xx-small": "9px",
                "x-small": "11px",
                "small": "13px",
                "medium": "15px",
                "large": "18px",
                "x-large": "23px",
                "xx-large": "30px"
            }
        },
        "panel": {
            "type": "fluid.prefs.panels.lineSpace",
            "container": ".flc-prefsEditor-line-space",
            "template": "%prefix/PrefsEditorTemplate-lineSpace.html",
            "message": "%prefix/lineSpace.json"
        }
    },
    "textFont": {
        "type": "fluid.prefs.textFont",
        "classes": {
            "default": "",
            "times": "fl-font-uio-times",
            "comic": "fl-font-uio-comic-sans",
            "arial": "fl-font-uio-arial",
```

```
            "verdana": "fl-font-uio-verdana"
        },
        "enactor": {
            "type": "fluid.prefs.enactors.textFont",
            "classes": "@textFont.classes"
        },
        "panel": {
            "type": "fluid.prefs.panels.textFont",
            "container": ".flc-prefsEditor-text-font",
            "classnameMap": {"textFont": "@textFont.classes"},
            "template": "%prefix/PrefsEditorTemplate-textFont.html",
            "message": "%prefix/textFont.json"
        }
    },
    "contrast": {
        "type": "fluid.prefs.contrast",
        "classes": {
            "default": "fl-theme-default-prefsEditor",
            "bw": "fl-theme-bw-prefsEditor fl-theme-bw",
            "wb": "fl-theme-wb-prefsEditor fl-theme-wb",
            "by": "fl-theme-by-prefsEditor fl-theme-by",
            "yb": "fl-theme-yb-prefsEditor fl-theme-yb",
            "lgdg": "fl-theme-lgdg-prefsEditor fl-theme-lgdg"
        },
        "enactor": {
            "type": "fluid.prefs.enactors.contrast",
            "classes": "@contrast.classes"
        },
        "panel": {
            "type": "fluid.prefs.panels.contrast",
            "container": ".flc-prefsEditor-contrast",
            "classnameMap": {"theme": "@contrast.classes"},
            "template": "%prefix/PrefsEditorTemplate-contrast.html",
            "message": "%prefix/contrast.json"
        }
    },
    "tableOfContents": {
        "type": "fluid.prefs.tableOfContents",
        "enactor": {
            "type": "fluid.prefs.enactors.tableOfContents",
            "tocTemplate": "../../../../components/tableOfContents/html/TableOfContents.html"
        },
        "panel": {
            "type": "fluid.prefs.panels.layoutControls",
            "container": ".flc-prefsEditor-layout-controls",
            "template": "%prefix/PrefsEditorTemplate-layout.html",
            "message": "%prefix/tableOfContents.json"
        }
    },
    "emphasizeLinks": {
        "type": "fluid.prefs.emphasizeLinks",
        "enactor": {
            "type": "fluid.prefs.enactors.emphasizeLinks",
            "cssClass": "fl-link-enhanced"
        },
        "panel": {
            "type": "fluid.prefs.panels.linksControls",
            "container": ".flc-prefsEditor-links-controls",
            "template": "%prefix/PrefsEditorTemplate-links.html",
            "message": "%prefix/links.json"
        }
    },
    "inputsLarger": {
        "type": "fluid.prefs.inputsLarger",
        "enactor": {
            "type": "fluid.prefs.enactor.inputsLarger",
            "cssClass": "fl-text-larger"
        },
        "panel": {
            "type": "fluid.prefs.panel.inputsLarger",
            "container": ".flc-prefsEditor-inputsLarger",  // the css selector in the template where the panel
```

```
is rendered
            "template": "%prefix/PrefsEditorTemplate-inputsLarger.html",
            "message": "%prefix/inputsLarger.json"
        }
    },
    groups: {
        "linksControls": { // this defines a composite panel
            "container": ".flc-prefsEditor-links-controls",
            "template": "%prefix/PrefsEditorTemplate-linksControls.html",
            "message": "%prefix/linksControls.json",
            "type": "fluid.prefs.panel.linksControls",
            "panels": ["emphasizeLinks", "inputsLarger"] // the composite panel includes these two subpanels
        }
    }
}
```

Sharing data between panels and enactors

In some cases, panels and enactors may need to share data, such as a list of class names. In these cases, define the data at the root of the relevant preference block and reference it within the panel and enactor blocks, as seen in the example above (in the `contrast` and `textFont` preference blocks). The general structure and syntax is highlighted below:

```
{
    "prefBlockName": {
        "type": "typename",
        "dataToBeShared": {
            ...
        },
        "enactor": {
            "type": "enactor.component.name",
            "sharedData": "@prefBlockName.dataToBeShared"
        },
        "panel": {
            "type": "panel.component.name",
            "sharedData": "@prefBlockName.dataToBeShared",
            ...
        }
}
```

Configuring Multiple Panels and Enactors for a Single Preference

Each preference block can contain configuration for *at most* one enactor and one panel. If more than one enactor and/or panel needs to be configured for a given preference, you can create another preference block with a new namespace.

```
{
    ...
    // Standard preference block configuration
    "textSize": {
        "type": "fluid.prefs.textSize",
        "enactor": {
            "type": "fluid.prefs.enactors.textSize"
        },
        "panel": {
            "type": "fluid.prefs.panels.textSize",
            "container": ".flc-prefsEditor-text-size",
            "template": "%prefix/PrefsEditorTemplate-textSize.html",
            "message": "%prefix/textSize.json"
        }
    },
    // An additional panel and enactor configured for the same preference
    // Note: the namespace, "additional", can be any valid string
    "textSize.additional": {
        "type": "fluid.prefs.textSize",
        "enactor": {
            "type": "fluid.prefs.enactors.foo"
        },
        "panel": {
            "type": "fluid.prefs.panels.foo",
            "container": ".flc-prefsEditor-foo",
            "template": "%prefix/PrefsEditorTemplate-foo.html",
            "message": "%prefix/foo.json"
        }
    }
    ...
}
```

Configuring a Single Panel for Multiple Enactors and Preferences

Each preference block defines *only one* preference, even if multiple preferences (with their own enactors) are displayed in the *same panel*. In these cases, multiple preference blocks still need to be configured:

- Each preference block declares the common panel type, and
- the detail information for this panel is defined in any *one* (and only one) of these panel block.

The example below shows two preferences (`emphasizeLinks` and `inputsLarger`) sharing the same panel `fluid.prefs.panels.linksControls` (lines 9 and 22). The details for this panel are only defined in the preference block for `emphasizeLinks`, on lines 10-12.

```
{
    "emphasizeLinks": {
        "type": "fluid.prefs.emphasizeLinks",
        "enactor": {
            "type": "fluid.prefs.enactors.emphasizeLinks",
            "cssClass": "fl-link-enhanced"
        },
        "panel": {
            "type": "fluid.prefs.panels.linksControls",
            "container": ".flc-prefsEditor-links-controls",
            "template": "%prefix/PrefsEditorTemplate-links.html",
            "message": "%prefix/links.json"
        }
    },
    "inputsLarger": {
        "type": "fluid.prefs.inputsLarger",
        "enactor": {
            "type": "fluid.prefs.enactors.inputsLarger",
            "cssClass": "fl-text-larger"
        },
        "panel": {
            "type": "fluid.prefs.panels.linksControls"
        }
    }
}
```