

Customize the Code Widget Properties Panel

Overview

The Code Widget Properties panel is broken down into three components.

- Parameters
- Sections
- Panels

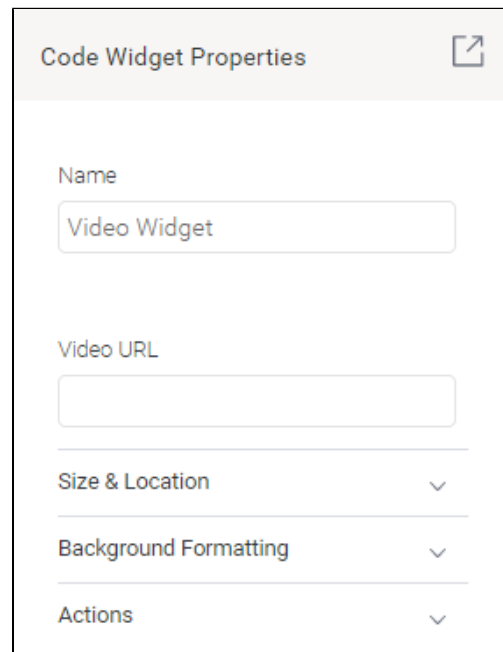
Parameters are the controls that a user will interact with and use to modify values. In the following example "Name" and "Video URL" are both parameters. The JavaDoc for the parameters can be found [here](#).

Sections are the containers of parameters, and they usually contain a group of similar parameters, such as formatting options.

Sections can typically be expanded or collapsed, but in the following screenshot, there are actually five sections:

Three of the sections in the screenshot — Size & Location, Background Formatting and Actions — are all expandable sections.

The other two sections — Name and Video URL — are both in "Always Expanded" sections, so they look like a single section to the user. They are, however, two separate sections from a code perspective.



The Code Widget panel allows you to add your own sections where you wish. To do this, implement a class that extends `CanvasWidgetPanel`, which by default provides the following sections:

- Name
- Size & Location
- Background Formatting
- Actions

Build a Code Widget Properties panel

The full code for the following example can be found here: <https://github.com/YellowfinBI/VideoWidgetExample>

The goal of this example is to build a panel that looks like this:

Code Widget Properties


Name

Video URL

Size & Location 

Background Formatting 

Actions 

To create a panel:

1. extend `CanvasWidgetPanel`;
2. override the function ***buildSections***; and,
3. add any sections that are to be available to the user.

VideoWidgetPanel.java

```
package com.hof.video;

import com.hof.mi.widgetcanvas.panelcollection.CanvasWidgetPanel;
import com.hof.mi.widgetcanvas.panelcollection.CanvasWidgetPanelInfo;

public class VideoWidgetPanel extends CanvasWidgetPanel {

    public VideoWidgetPanel(CanvasWidgetPanelInfo info) {
        super(info);
    }

    /**
     * Helper function that allows
     */
    @Override
    protected void buildSections() {
        super.buildSections();
        sections.add(1, new VideoWidgetURLSection());
        sections.add(2, new CollapsedVideoOptionsSection());
    }
}
```

In this example we are overriding *buildSections*, which is called when Yellowfin generates the panel for display in the front end. Calling *super.buildSections()* automatically adds the default sections, so if your panel doesn't require the default sections, don't call *super.buildSections()*.

Since we want the URL and Options sections to be added towards the top of the panel, we can use *section.add(Integer, Section)* to add to the relevant location.

The Java for the two sections is below.

VideoWidgetURLSection.java

```
package com.hof.video;

import com.hof.parameters.GeneralPanelOptions;
import com.hof.parameters.InputType;
import com.hof.parameters.Parameter;
import com.hof.parameters.ParameterDisplayRule;
import com.hof.parameters.ParameterImpl;
import com.hof.parameters.ParameterSection;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

public class VideoWidgetURLSection extends ParameterSection {

    /**
     * SectionKey is used by the front-end to create a unique panel/section combination to allow
     * quick removal, what is returned here should be unique
     */
    @Override
    public String getSectionKey() {
        return "youtube-url";
    }

    /**
     * The name of the Section that will be displayed to the user, if the section has display name
     * turned on
     */
    @Override
    public String getName() {
        return "Video URL";
    }

    /**
     * Parameters that will be available to a user in this section
     */
    @Override
    public List<Parameter> getParameters() {
        List<Parameter> parameterList = new LinkedList<>();

        ParameterImpl p = new ParameterImpl();
        p.setName("Video URL"); //Name that will be displayed to the user when they are editing this.
        p.setProperty("videoURL"); //Property that this will be stored as in the database, and the name of
the property to access when using getFormatValue
        p.setInputType(InputType.TEXTBOX); //Define this option as a textbox
        p.setModelKey("formats"); //All of the options are saved in formats.
        p.addViewOption("width", "240px"); //Set the display width of the textbox to 250px
        parameterList.add(p);
        return parameterList;
    }

    @Override
    public List<ParameterDisplayRule> getDisplayRules() {
        return null;
    }

    @Override
    public GeneralPanelOptions getSectionOptions() {
        return null;
    }

    @Override
    public Map<String, ?> getData() {
        return null;
    }
}
```

CollapsedVideoOptionsSection.java

```
package com.hof.video;

import com.hof.parameters.GeneralPanelOptions;
import com.hof.parameters.InputType;
import com.hof.parameters.Parameter;
import com.hof.parameters.ParameterDisplayRule;
import com.hof.parameters.ParameterImpl;
import com.hof.parameters.ParameterSection;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

public class CollapsedVideoOptionsSection extends ParameterSection {

    /**
     * SectionKey is used by the front-end to create a unique panel/section combination to allow
     * quick removal, what is returned here should be unique
     */
    @Override
    public String getSectionKey() {
        return "video-collapsed";
    }

    /**
     * The name of the Section that will be displayed to the user, if the section has display name
     * turned on
     */
    @Override
    public String getName() {
        return "Video Options";
    }

    /**
     * Parameters that will be available to a user in this section
     */
    @Override
    public List<Parameter> getParameters() {
        List<Parameter> parameterList = new LinkedList<>();

        ParameterImpl p = new ParameterImpl();
        p.setName("Video Start Time"); //Name that will be displayed to the user when they are editing this.
        p.setProperty("videoStartTime"); //Property that this will be stored as in the database, and the
name of the property to access when using getFormatValue
        p.setInputType(InputType.TEXTBOX); //Define this option as a textbox
        p.setModelKey("formats"); //All of the options are saved in formats.
        p.addViewOption("width", "240px"); //Set the display width of the textbox to 250px
        parameterList.add(p);
        return parameterList;
    }

    @Override
    public List<ParameterDisplayRule> getDisplayRules() {
        return null;
    }

    /**
     * Create the options for this section so that we can display in a
     * collapsed or expanded mode.
     */
    @Override
    public GeneralPanelOptions getSectionOptions() {
        GeneralPanelOptions options = new GeneralPanelOptions();
        options.setShowName(true);
        options.setExpandable(true);
        options.setExpanded(false);
    }
}
```

```
        return options;
    }

    @Override
    public Map<String, ?> getData() {
        return null;
    }
}
```

Next steps

By now, you should have the [back-end code](#) written, the [front-end code](#) written. Once you've customized your Code Widget Properties pane using the steps above, you're ready to [bundle the code into a jar file, then upload it to Yellowfin](#).

To further assist you during code widget development, click on the [code widget reference page](#) for samples, API links and detailed descriptions.

[Customize the Code Widget Properties Panel#top](#)