

# Customizing Blender

A lot of aspects of RenderMan for Blender (RfB) such as defaults and labels can be customized using JSON (.json files).

## Environment Variables

Environment variables can be set using an `rfb_envvars.json` file. This file is automatically read on start up from the user's config directory (see: [Blender's documentation](#)). Note, this method cannot be used to set any standard Blender environment variables, including OCIO.

An example of the format of the file:

```
{
  "$schema": "../schemas/rfbEnvVarsSchema.json",
  "environment": {
    "RFB_LOG_LEVEL": {
      "value": "INFO"
    },
    "RFB_SITE_PATH": {
      "value": "/path/to/rfb_config_files"
    }
  }
}
```

**Advanced:** if you would like RfB to ignore the `rfb_envvars.json`, set the environment variable `RFB_IGNORE_ENVVARS_JSON` to 1

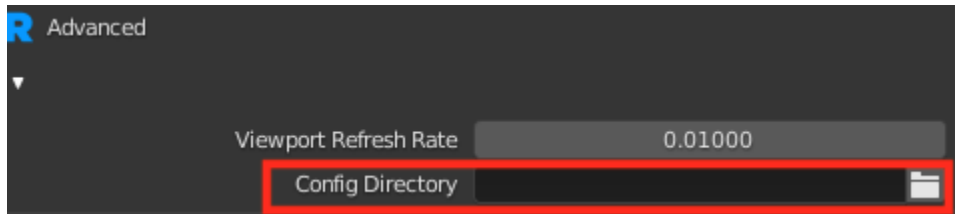
## Configuration Paths

To find configuration JSON files, RfB will check the following environment variables in order:

- `RFB_SITE_PATH`
- `RFB_SHOW_PATH`
- `RFB_USER_PATH`

Each environment can also contain multiple, color-separated, paths (on Windows, these are semi-colon separated paths).

You can also specify a config path in the RenderMan for Blender preferences, under the Advanced section (requires restart).



## rfb.json

This json file configure certain settings such as your site's Tractor engine configuration, directory mappings, or adding extra user tokens. Here's an example:

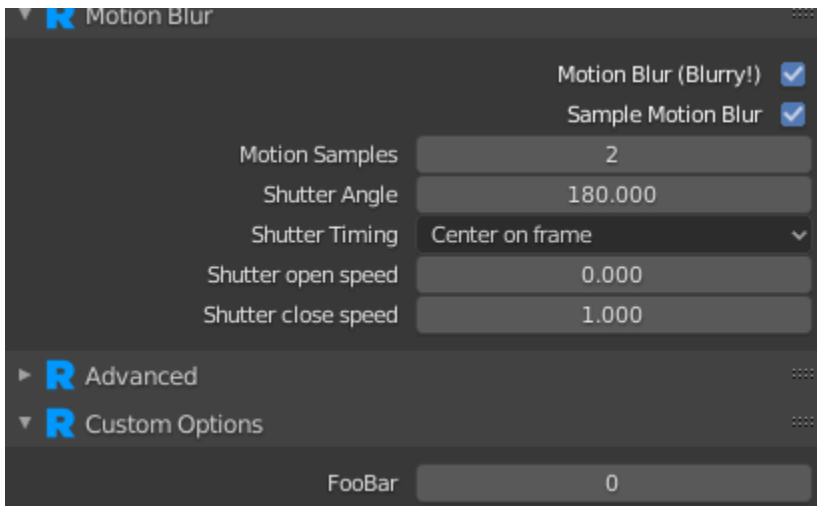
```
{
  "$schema": "../schemas/rfbSchema.json",
  "tractor_cfg": {
    "engine": "tractor-engine",
    "port": 80,
    "user": ""
  },
  "dirmaps": {
    "seahome_to_z": {
      "from": "/seahome/",
      "to": "Z:/",
      "zone": "UNC"
    }
  },
  "woffs": [
    "R90007"
  ],
  "user tokens": ["PROD", "SHOT"]
}
```

## Configuring Panel Properties

The properties in some RenderMan panels can be configured to use different defaults. Here's an example.

```
{
  "name": "rman_properties_scene",
  "params": [
    {
      "name": "motion_blur",
      "label": "Motion Blur (Blurry!)",
      "type": "int",
      "default": 1,
      "help": "TURNS ON MOTION BLUR!"
    },
    {
      "panel": "RENDER_PT_renderman_custom_options",
      "name": "user_foobar",
      "label": "FooBar",
      "riopt": "user:foobar",
      "type": "int",
      "default": 0,
      "help": "This is a test user option"
    }
  ]
}
```

"name", should be the name of the configuration we are interested in overriding (in this case it's the scene properties). "params" should follow name. In our example, the first property is an existing one and we are overriding the label, the default, and the description. If we are adding a new property, we also need to specify which panel this property should be drawn in; in this case we want to add it to the scene's custom options panel. This example also shows how we can add a RenderMan user option. Note, the name of the json file itself in this case does not matter; as long as the first "name" matches an already known configuration it will work. Here's what the result should look like after applying the above override:



As you can see, the label for Motion Blur has changed along with its default. There is also a new FooBar user option under Custom Options.

To get a list of configurable panels, you can run the following code in Blender's Python console:

```
import RenderManForBlender.rfb_api as rfbapi
rfbapi.GetConfigurablePanels()
```

## Shader Defaults

We can also override the defaults of any of RenderMan's shaders (bxd's, patterns etc.) Here's an example where we override parameters in PxrEnvDayLight:

```
{
  "name": "PxrEnvDayLight.args",
  "params": [
    {
      "name": "sunDirection",
      "default": [0.0, 0.0, 1.0]
    }
  ]
}
```

In this case we want to change the default for the sunDirection to default to pointing up in the Z-direction. "name" should be the name of the Args file we want to override (this can also be the name of an OSL shader), follow by the parameters we want to configure. Note, the name of the json file itself in this case does not matter; as long as the first "name" matches an already known shader Args file it will work.

## Display Channels


Finally, we can also add custom display channels that can be used in AOVs. Note, the name of this JSON file **does** matter; it must be named: **rman\_dspyc\_han\_definitions.json**. Here's an example where we add a new channel called MyAOV.

```
{
  "$schema": "./schemas/rfbDspyChansSchema.json",
  "channels": {
    "MyAOV": {
      "description": "This a custom AOV",
      "channelType": "color",
      "channelSource": "MyAOV",
      "group": "Pixar"
    }
  }
}
```

## Custom Shaders

You can also add your own custom shaders by using the environment variables `RMAN_SHADERPATH` for OSL shaders and `RMAN_RIXPLUGINPATH` for C++ shaders. For the latter, an accompanying Args file is required.

## Other Examples

You can also look at some other examples of JSON config files in Blender's Text Editor , under the Templates menu.

