Extending *iSEE*

Kevin Rue-Albrecht, Federico Marini, Charlotte Soneson, and Aaron Lun

2019-12-10

Contents

Preface

The Bioconductor *iSEE* package provides functions for creating an interactive graphical user interface (GUI) using the RStudio *Shiny* package for exploring data stored in *SummarizedExperiment* objects, including row- and column-level metadata (Rue-Albrecht et al., 2018). In this book we describe how to create web-applications that leverage builtin panels and develop new ones.

Chapter 1

Panel classes

1.1 Overview

The types of panels available to compose an *iSEE* app are defined as a hierarchy of S4 classes.

- `Panel`*
  - `DotPlot`*
    - `ColumnDotPlot`*
      - `RedDimPlot`
      - `ColDataPlot`
      - `FeatAssayPlot`
    - `RowDotPlot`*
      - `RowDataPlot`
      - `SampAssayPlot`
  - `Table`*
    - `RowTable`*
      - `RowStatTable`
    - `ColumnTable`*
      - `ColStatTable`
  - `HeatMapPlot`

Some of those classes are "virtual" (indicated by *), meaning that they cannot be directly instanciated as panels in the GUI. Instead, virtual panel classes define families of panels that share groups of properties. Virtual classes are meant to be used as the parent of one or more concrete classes. In contrast, concrete classes must define fully-functional panels that can be embedded in a GUI, interact with other panels, receive and process data, and generate an output such as a plot or a table, accompanied by a code chunk to display in the code tracker for reproducibility.

1.2 The Panel class

The top-most class is called `Panel`. It is a virtual class that defines the core properties common to any panel - existing or future - that may be displayed in the interface.

`PanelId`	Integer index indicating the i^{th} panel of a given type.
`PanelHeight`	Height of the panel, in pixels.
`PanelWidth`	Width of the panel, an integer value indicating the number of columns to use, from 1 to 12.
`SelectBoxOpen`	Logical value indicating if the *Selection parameters* box of the panel is open when the app starts.
`SelectByPlot`	Encoded name of the panel from which to receive a selection of data points.
`SelectMultiType`	Keyword indicating the method to deal with multiple incoming selections of data points.
`SelectMultiSaved`	Integer index indicating a single data point selection to use, among multiple incoming selections.

1.3 The DotPlot and Table panel families

The `Panel` virtual class is directly derived into two major virtual sub-classes:

- `DotPlot`
- `Table`

Those classes introduce properties that are specific to distinct subsets of panel types.

The `DotPlot` class introduce parameters specific to panels where the output is a `ggplot` object and each row in the data-frame is represented as a point in a plot.

The `Table` class introduce parameters specific to panels where the main output is a data-frame directly displayed as a table in the GUI.

In addition, the `HeatMapPlot` class defines a special panel class that directly extends the `Panel` class, as it introduces a set of parameters distinct from both the `DotPlot` and `Table` panel families. This panel type is described in further details in a separate section below.

1.4 The ColumnDotPlot and RowDotPlot panel families

1.5 Built-in ColumnDotPlot panel classes

1.6 Built-in RowDotPlot panel classes

1.7 The ColumnTable and RowTable panel families

1.8 Built-in ColumnTable panel classes

1.9 Built-in RowTable panel classes

1.10 The HeatMapPlot panel class

This type of panel introduces parameters specific to panels where the output is a heat map, with each row representing a feature and each column representing a sample in the `se` object.

Chapter 2

The app server

2.1 Reactive objects

2.2 Persistent (non-reactive) objects

2.3 The app memory

The app `memory` is a list of instances created from available panel classes, which defines the order in which individual panels are displayed in the GUI.

2.4 The panel API

2.4.1 .cacheCommonInfo

Each individual panel type (e.g., *Reduced dimension plot*) and family of panel types (e.g., *Column dot plot*) defines a `.cacheCommonInfo` function.

This function is called for each panel instance in memory when the app is initialized. It allows the app to efficienly compute a single time common information that only depends on the input `se` object, and may be frequently reused during the runtime of the app.

Following the hierarchy of panel types, each call to the signature takes a panel instance `x` and the `se` object, and caches common information relevant to all instances of that panel type in the `se` object itself, before calling `callNextMethod()` to invoke the next parent signature.

The top-most signature - for the `Panel` class - returns the `se` object that contains all the cached information.

Note that this function only populates the cache for the first panel of each type; it is a no-op if the common cache has already been initialized.

2.4.2 .refineParameters

Each individual panel type (e.g., *Reduced dimension plot*) and family of panel types (e.g., *Column dot plot*) defines a `.refineParameters` function.

This function is called for each panel instance in memory when the app is initialized, and also when a new panel is added to the GUI. It inspects the parameters of a given panel instance, and replaces invalid parameters with sensible values for a given `se` object.

Following the hierarchy of panel types, each call to the signature takes an instance `x` and the `se` object, and first calls `callNextMethod()` to invoke the next parent signature, to refine generic parameters before processing specific ones.

The called signature ultimately returns the updated instance panel `x`, or `NULL` if the panel instance is not available for this app.

2.5 Initialization of the app server

The app server is initialized as soon as a valid `se` object is provided. This can be either in the call to `iSEE(se)` or using the Shiny file upload button in apps that were launched without providing the `se` arguments, e.g., `iSEE()`.

The `initialize_server` function takes the `se` object and the list holding reactive values used to trigger re-rendering of the GUI, as described above.

The very first step invokes the function `.sanitize_SE_input` on the `se` object. This function coerces the `se` to `SingleCellExperiment`, flattens nested DataFrames, adds row and column names, and removes other non-atomic fields. In addition, it also sanitizes the `SingleCellExperiment` object by moving internal fields into the column- or row-level metadata, making them visible in the *Column statistics table* and *Row statistics table* panels, respectively. The function returns both the sanitized `se` object that will be used by the app, and the list of R commands that will be displayed in the code tracker for users.

Next, the server invokes the `checkColormapCompatibility` function. This function takes the `se` object and the optioanl `colormap` provided to `iSEE()`, and carries out a number of compatibility checks between the two objects. The function collects a character vector of issue messages that are displayed - if any - as warning messages in GUI during initialization.

Next, the `.cacheCommonInfo` and `.refineParameters` are successively invoked on each panel instance initialized in the app memory. As described in a separate section above, the first function precomputes and caches information specific to the `se` object and frequently used throughout the runtime of the app. The

second function ensures that each panel instance is initialized with valid parameters; it replaces any invalid parameters with sensible values for a given `se` object.

Next, persistent (non-reactive) objects are initialized:

- the app `memory` (see this section)
- the count of panels of each type, used to assign increasing ID to new panel instances
- the list of commands to display in the code tracker for each panel instance
- the list of data point coordinates selectable in each panel instance[1]
- a list of miscellaneous cached information[2]

[1]Data points downsampled for rendering speed performance remain selectable, even though they are not visible in the plot.

[2]The plot that contain the legend keys of *Heatmap* panels is currently cached as miscellaneous information retrieved separately when rendering the GUI.

Chapter 3

The plotting API

3.1 .getPlottingFunction

Each panel type available for use in the GUI defines a `.getPlottingFunction`.

This function is called within `.createRenderedOutput`, which is triggered by observers when the value of the panel input widgets are changed by users, or when a new panel is added to the GUI.

The `.getPlottingFunction` function inspects the parameters for a given panel instance, and uses the app `memory` of all active panels and parameters, the coordinates of data points in each plot panel, the `se` object, and the `colormap` to generate all the information necessary to render the outputs of this panel and those that depend on it.

For `DotPlot` panels, the output is a list that includes:

- the list of commands to display in the code tracker
- the coordinates of data points in the plot
- the `ggplot` object

For `Table` panels, the output is a `datatable`.

For the `HeatMap` panel, the function does not return any value. Instead it sets relevant elements in the `output` object of the Shiny session.

Chapter 4

Developing new panels

First things first, we need to load the *iSEE* package for this chapter.

```
library(iSEE)
```

4.1 Create a new S4 class

In the chapter Panel classes, we saw how each type of panel is defined as an S4 class, organised in a hierarchy that allows new panel classes to inherit sets of properties from parent classes.

Therefore, developing a new panel starts with the creation of a new class.

The choice of a parent class for the new panel depends on the properties that we want that new panel to have. For instance, let us say that we want to define a new panel that has all the functionality of the *Reduced dimension plot* panel type, but summarizes data into a layer of hexagonal bins instead of showing each individual data point. To do so, we declare our new class with a new unique name, *e.g.* `RedDimHexbinPlot`, that contains the builtin panel class `RedDimPlot`.

```
setClass("RedDimHexbinPlot", contains="RedDimPlot")
```

creation of a subclass, adding a UI, adding an observer, adding the XY data logic, adding the plot logic. I

Bibliography

Rue-Albrecht, K., Marini, F., Soneson, C., and Lun, A. T. L. (2018). isee: Interactive summarizedexperiment explorer. *F1000Res*, 7:741.