PathSimR – Shiny Documentation

Contents

[1 Overview 2](#_Toc19543343)

[2 Code Structure & Breakdown 2](#_Toc19543344)

[2.1 Shiny UI Code Block 2](#_Toc19543345)

[2.1.1 Introduction and Overview & Glossary Tabs 2](#_Toc19543346)

[2.1.2 Wizard 1 – Setup Tab 2](#_Toc19543347)

[2.1.3 Wizard 2 – Data Entry Tab 2](#_Toc19543348)

[2.1.4 Wizard 3 – Final Wizard Tables & Download Tab 2](#_Toc19543349)

[2.1.5 Service Distribution Tool Tab 3](#_Toc19543350)

[2.1.6 Tool 1 – Network Import & Visualisation Tab 3](#_Toc19543351)

[2.1.7 Tool 2 – Simulation Setup & Run Tab 3](#_Toc19543352)

[2.1.8 Tool 3 – Simulation Outputs Tab 3](#_Toc19543353)

[2.1.9 Tool 4 – Download Outputs Tab 3](#_Toc19543354)

[2.2 Shiny Server Code Block 3](#_Toc19543355)

[2.2.1 Figures for Modals 3](#_Toc19543356)

[2.2.2 Navigation Buttons 4](#_Toc19543357)

[2.2.3 Name Input tables and checks 4](#_Toc19543358)

[2.2.4 Data Entry Service Point Tabs UI 4](#_Toc19543359)

[2.2.5 Creates the transition probability inputs & delay departure entry (dynamic based on number of nodes & exits) 4](#_Toc19543360)

[2.2.6 Defines 'tabs' layout (dynamic based on number of nodes & exits) 4](#_Toc19543361)

[2.2.7 Trial Variable & Calendar Inputs & Tables 4](#_Toc19543362)

[2.2.8 Input Checklist 4](#_Toc19543363)

[2.2.9 Length of Service Model Fits Tab 5](#_Toc19543364)

[2.2.10 Length of Service Scaled Means Tab 5](#_Toc19543365)

[2.2.11 Template upload and checks 5](#_Toc19543366)

[2.2.12 Network Visualisation 5](#_Toc19543367)

[2.2.13 Simulation Code 6](#_Toc19543368)

[2.2.14 Output Rendering 6](#_Toc19543369)

# Overview

Along with being a standalone R script (described in the document “PathSimR – Technical Documentation”), PathSimR is also implemented as part of a Shiny application (app). The app includes a number of features not included in the script version, all of which are highlighted in this report. The app is split into 2 key sections: Shiny User Interface (UI), which defines all of the visuals and navigation of the tool, and the Shiny Server, which includes all of the computation components of the tool and any extra dynamic elements of the UI. The following section will go through each part of the code, highlighting how the code sections fit together and where any changes could take place. Section titles below correspond to bookmarks in the R script.

# Code Structure & Breakdown

## Shiny UI Code Block

This section covers all of the static UI design (text, buttons, tabs, etc.) and is where the vast majority of the UI is created. The following subsections go through each UI tab in the tool. Most of the tabs have a side panel/main panel layout, with the sidebar always being defined first.

Throughout the app, blue text accompanied by an “i” in a blue circle can be clicked on to bring up a pop-up window (a “modal”) which contains further information and/or help, for example clicking the icon or text in figure 1



Figure 1

Launches the pop-up in figure 2

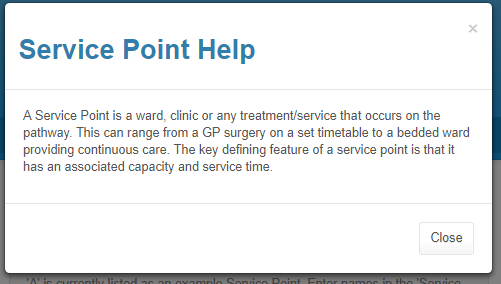


Figure 2

These modals are created using the function *bsmodal* from the package *shinyModal.* They are created in the same format throughout the app.

### Introduction and Overview & Glossary Tabs

Both these tabs contain text information about how to use the tool and reinforce some of the key terms used throughout. The R code specifies the HTML layout of the page and the text to be displayed, and in the case of the Introduction tab, code to create buttons which the can be used to navigate to either a data entry “Pathway Wizard”, or the simulation tool proper. Additional terms can be entered into the Glossary by finding the correct category (Overview, Wizard & Setup Terms, Outputs Terms) and then copying the text input from a previous item.

### Wizard 1 – Setup Tab

This tab contains two matrix input boxes created using the function *matrixInput* from the package *ShinyMatrix*. They take the names of the Service Points and the Exits. These then render into two data tables that contain formatted versions of the name lists (see section 2.2.3). An error message will appear if there are matching names in the Service Point & Exit entry columns.

### Wizard 2 – Data Entry Tab

The UI code for this tab only defines the sidebar. The input boxes in the main panel are dynamically created on the Server side (see sections 2.2.4, 2.2.5 & 2.2.6 of this documentation).

### Wizard 3 – Final Wizard Tables & Download Tab

The four dynamic tables in this tab - Issues log, LoS Means, Network Information and Calendar Information – are created in the Server code (sections 2.2.7 & 2.2.8). The download and progress buttons will only appear under certain conditions, and are created using the experimental feature *uiOutput* from the core Shiny package.

### Service Distribution Tool Tab

This tab contains 2 sub-tabs which provide different methods for estimating appropriate length of stay (LoS) distributions and parameters for pathway service points: Model fits to user data & Scale data by mean.

The first sub-tab, “Model fits to user data”, provides a tool which can be used to derive distribution types and parameters from actual row-level service data (uploaded by the PathSimR user) using the package *fitdistrplus*. The distribution fitting itself is handled on the server side (see section 2.2.9). The UI side code simply creates a CSV upload button (using base Shiny functions) and increases the maximum permitted file upload size from its default value to 30Mb, by setting *options(shiny.maxRequestSize = 30\*1024^2)*.

The second sub-tab provides a simpler way of choosing distribution types and parameters for service point LoS: the code specifies a pre-populated table of service point types along with associated distributions and parameters[[1]](#footnote-1). The user picks service point types from a drop down menu, types the relevant mean LoS for their local service into a numerical input box, and clicks an activation button (all defined using standard Shiny input functions) and functions are then applied on the server side (see section 2.2.10 for details) to return distribution parameters appropriate to that mean.

### Tool 1 – Network Import & Visualisation Tab

This is the first tab of the proper simulation tool and is used for both template users and wizard users. There is a checkbox to select whether or not to pull through wizard outputs and if not, then two csv upload boxes to input the two templates. The sidebar also includes checkboxes to show/hide the uploaded templates and also change the network figure.

The main panel contains a network visualisation, issues log for uploaded templates, Network Information and Calendar Information, the creation of which can be found in section 2.2.11 & 2.2.12.

### Tool 2 – Simulation Setup & Run Tab

The UI design on this tab is limited and is primarily designed to bridge the user between data inputs and simulation outputs. It contains conditional entry boxes in the sidebar depending on whether Trial Simulation or Full Simulation is chosen. The *shinyalert* that appears when the user presses “Run Simulation” is coded in the server at the start of the simulation loop.

### Tool 3 – Simulation Outputs Tab

Simulation Outputs uses a *navlistPanel* to have tabs listed in the sidebar. The content of these tabs is dictated by whether the user ran a Trial simulation or a Full Simulation. The majority of tabs simply show tables and graphs rendered from the simulation and the Output Interpretation tab is solely for generic information.

### Tool 4 – Download Outputs Tab

Tab contains 3 download buttons and descriptions about what each one will render once pressed.

## Shiny Server Code Block

This section covers all of the dynamic UI design (dynamically named tabs and inputs) and all simulation computation. The following subsections go through each server bookmark in the code. For details about how the simulation code operates, consult the technical documentation.

### Figures for Modals

Some of the modals in the UI side contain graphs/tables that require code to render rather than relying on an external png format image file. There are two draft calendars for the Data Entry Wizard and an example distribution fit for the Service Distribution Tool.

### Navigation Buttons

Throughout PathSimR, there are a number of navigation buttons to move between tabs. This is achieved using the *show/hideTab* functions along with *updateTabsetPanel* function which allows the code to change the visible tabs and which tab should be shown. There are *observeEvent* statements for each of the buttons that updates the view.

### Name Input tables and checks

This set of code tidies up the Service Point and Exit names entered into the *ShinyMatrix*. It removes all duplicates, trims all trailing whitespace and replaces any internal spaces with underscores. There is also a check to see if there are any duplicate names between the lists and raises a warning if there is.

### Data Entry Service Point Tabs UI

Rather than defining the UI design of this tab in the UI section of the App, the Data Entry page is dynamically created to have a tab for each named Service Point and then the Transition Matrix entry will also include all of the named Exits as options.

The code block begins by bringing through each of the Service Point and Exits names into the *renderUI* function.

### Creates the transition probability inputs & delay departure entry (dynamic based on number of nodes & exits)

The first major operation is to create the transition inputs for each of the named Service Point tabs that include a box for each Service Point and Exit. This is done using a *lapply* statement that for each Service Point *j* and node *i*, creates a transition box that includes a proportion *numericInput* and then the delay distribution and parameters, each of which have a *conditionalPanel* to change the associated text and where the variable is saved. All the dynamic elements are saved as shinyInputs which are then unravelled using a *lapply* function later.

### Defines 'tabs' layout (dynamic based on number of nodes & exits)

A dynamic tab is created for each Service Point to allow the user to enter information about each one. This is created through a *lapply* function that loops over each of the Service Points and saves the resulting UI code down into the *renderUI* statement. The tabs include *conditionalPanels()* to dynamically update the Service Parameter section depending on which distribution is chosen, *numericInputs()* for the queue lengths, the transition & delays UI created in the previous section and finally a blank *shinyMatrix* for each of the external arrivals and capacity calendars. This is then all passed through the *tabsetPanel* function that is then rendered by *renderUI*.

### Trial Variable & Calendar Inputs & Tables

This section of code brings together all the data entered in the Data Entry tab and replicates the original PathSimR templates on which the wizard is based. The code uses *lapply* to collate the results from each tab without having to hard code for each service point.

### Input Checklist

The Input Checklist is a series of tests on the two templates which adds an issues statement to the issues vector if there check is triggered. The vector is then checked at the end to see if there are any statements, if not then the issues log shows completed.

The following checks are implemented (and again in the later template input checks):

* Testing if the transition matrix has rowsums of 1
* Testing if the distribution parameter inputs are correct
* Testing if the Queue inputs are correct
* Testing if the delay parameter inputs are correct
* Testing if there is at least 1 row of capacity and ext\_arrival rate for each service point
* Testing that every line in the caledar template has a value entry
* Testing that nodes that have 2+ lines in the calendar have any values in the start and end columns
* Testing that nodes that have a zero in the first start line in the calendar
* Testing that nodes that have 2+ lines in the calendar have matching values in the start and end columns
* Testing that nodes that have ascending start and end values
* Testing that there are arrivals to at least one node

On successful completion of the issues log, there is also a table rendered that includes that calculated mean LoS for each unit, based on the data input through the wizard.

### Length of Service Model Fits Tab

The first half of the Service Distribution tool allows the user to fit a distribution and parameters to their own data. The app uses the *fitdistrplus* package and uses a simple framework to find the suitable model. The code runs a model fit against the 5 core distributions and then performs AIC weighting to find the best fitting model to produce the *los\_fit\_table*. This table is then sorted into a ranked order so the user takes the top row model.

### Length of Service Scaled Means Tab

This code section works from a table loaded in which includes model fits to different POD and speciality levels based on BNSSG data. The table only includes the distribution name and one of the two parameters, the other is calculated based on the mean value input by the user. The code finds the corresponding information and produces a plot to show the scaled distribution along with the altered data in a table

### Template upload and checks

Performs the same checks as in section 2.2.8 but on the CSV uploads.

### Network Visualisation

PathSimR makes use of the ‘dot’ language and its implementation within the *DiagrammeR* package to create network visualisations. The code reads in the two templates and proceeds to create a service point data frame (ndf) that contains all the information regarding the service points and exits and the edge data frame (edf) that contains all the information about connections between service points. Various aesthetics are edited within the ndf to distinguish between service points and exits (colour and shape) and within the edf to highlight external arrivals and transition delays (arrow colour). Tooltips are also assigned and contain a capacity calendar for the service points and delay parameters for transition delay arrows. Overall, a html representation of the network is produced and can be called into the R viewer on successful runs of the code.

### Simulation Code

The Inputs and Initialisation section manipulates the reps, warm-up, simulation time and computation core variables depending on which simulation mode is used. This is where changes to the core clusters can be made. The code then follows the same pattern as the scripted version, parallelising execution of the replications using the *parLapply* function within the *Parallel* package which then has a large variable wrap up at the end to send the outputs back to the shiny app.

For more detail about the simulation code, consult the technical documentation.

### Output Rendering

The final section of code renders all of the tables and graphs created in the simulation loop for use in the application. They all follow the same pattern, pulling the metric/table/plot of interest from the *sim\_out()* reactive variable (which is created in *eventReactive* containing the simulation loop) and then formatting it for use in the app. The downloadable outputs are each nested within a *downloadHandler* and have *shinyalerts* that pop-up when the buttons are clicked that simply remind the user that the process might take a small amount of time.

1. Estimated by the PathSimR development team from national Hospital Episode Statistics (HES) data, or where not available, regional data, by maximum likelihood estimation for a standard set of candidate probability distributions, selected by Akaike Information Criterion (AIC) [↑](#footnote-ref-1)