

Guideline for publishing R Shiny apps as on-demand applications

The main goals of this guide are to:

- provide a toolbox of prebuilt Shiny modules that third-party R Shiny developers can plug into their apps and connect with the Seven Bridges Platform
- define the steps for third-party R Shiny developers or Velsara developers for creation and deployment of Shiny applications as on-demand web apps.

On-demand web applications

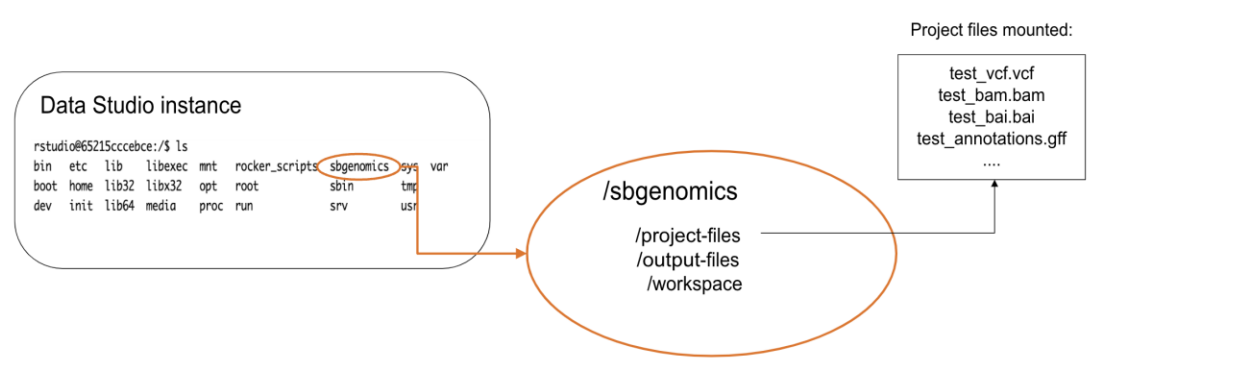
On-demand web apps utilize our Data Studio infrastructure for hosting Shiny or any other web applications.

Hosting an R Shiny app as an on-demand application means making it available to run from the [Public Web Apps gallery](#). When accessing the on-demand app from the gallery, a dedicated cloud instance will be allocated. The Docker image containing the application will be copied and run as a container. The entire process of fetching the instance and starting the app can take 2-3 minutes, like when starting a Data Studio analysis. Read more about our [Data Studio feature](#) here.

One advantage of this approach is that our [Seven Bridges File System](#) is available allowing users to access their project files directly. This means that project files are mounted on the local file system and can be imported into your analysis without having to download them.

The mounted directory is called **sbgenomics** and contains these subdirectories:

- Project-files
- Output-files
- Workspace



The ***/project-files*** subdirectory contains all your project files (the project from which you started the application). The users can only have “read” permissions on this directory.

However, the ***/output-files*** subdirectory aims to “save” the files you wish to write/export from your analysis/application into your project; therefore, you would have “write” permissions here. The files you store here will be exported to the project files once you stop your analysis/application.

The ***/workspace*** subdirectory can serve as a working directory where you can store temporary files produced during the web app execution, which you do not want to export. The content of this subdirectory will be saved and available the next time the app is initialized, but we strongly suggest not storing anything here.

Developing Shiny applications for publishing

We have two main use cases to cover:

1. Application developers develop Shiny apps on their local machines and want to publish them to our Platform as on-demand apps
2. Application developers develop Shiny apps in Data Studio environment and want to publish them to our Platform as on-demand apps

Local development

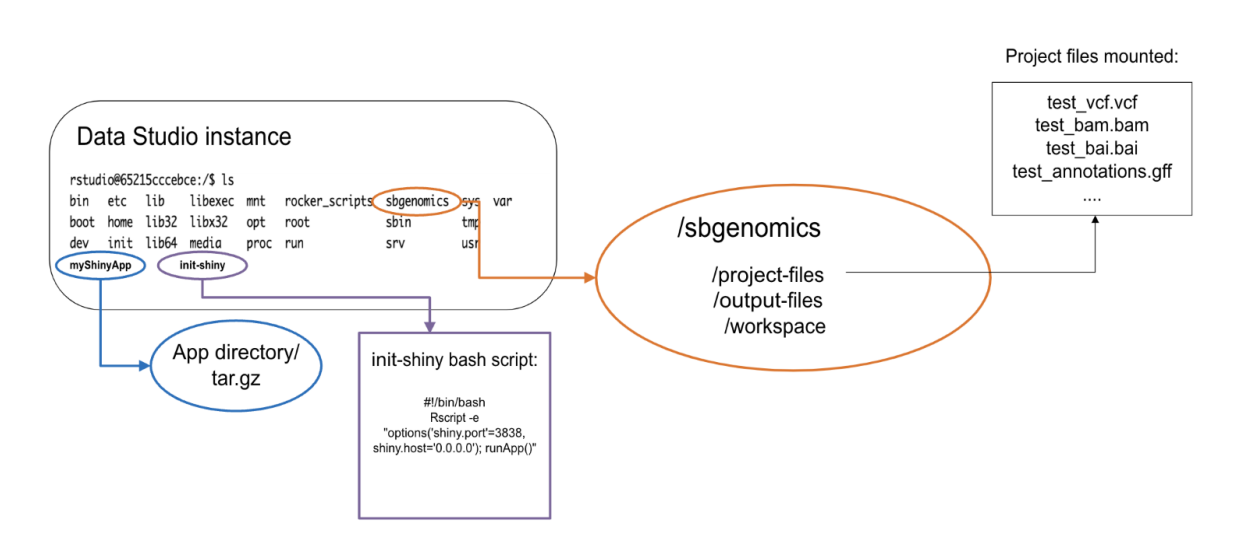
Developers have freedom to develop the app however they want, especially if the app does not have to communicate with the Platform (accessing project files, exporting files, etc). It is, however, recommended to follow some standard principles like using *renv*, *git*, *golem* framework, writing tests, etc.

The final output of the development should be a **Docker image** containing the application, which should then be pushed to the platform's [internal Docker registry](#) (cgc-images.sbgenomics.com).

Below you will find some guidelines on what to consider when building Docker images.

It is recommended to use the appropriate base image built on the **amd64 linux architecture**.

When building the docker image and adding app files, the app project directory should be placed in the instance root directory containing all necessary app files, or the app should be installed as a package (if using *golem*).



Please make sure to install all necessary dependencies for your web app.

If you want to use Platform files in the app or export outputs of the app to the Platform, you will have to assume the existence of the **sbgenomics** directory in the root directory of the instance where the app would run during local development. As mentioned before, this folder contains subdirectories `sbgenomics/project-files`, `sbgenomics/output-files`, `sbgenomics/workspace`. You would have to set absolute paths to these folders to work with them on the instance:

- `/sbgenomics/project-files`
- `/sbgenomics/output-files`
- `/sbgenomics/workspace`

Besides this interaction with the Platform, a **command for running the app** needs to be provided, including the port and the `runApp` command. The specifics of this command may vary depending on whether you are using the Golem framework or developing a standard - "vanilla" Shiny app.

This command should be included in a bash file named **init-shiny**, which should be created separately - outside the app project directory so that it ends up in the instance root directory. The app should be running on port **3838**.

An example of the **init-shiny** file:

```

#!/bin/bash
R -e "options('shiny.port'=3838,shiny.host='0.0.0.0');OmicCircosShiny::run_app()"

```

This can also be achieved via a Dockerfile build. You can add this segment to your Dockerfile (to be modified in accordance with the specific use case):

```

RUN touch init-shiny

EXPOSE 3838
# Create the init-shiny file with the shebang and the R code
RUN echo "#!/bin/bash" > /init-shiny && \
    echo "Rscript -e \"options('shiny.port'=3838, shiny.host='0.0.0.0'); OmicCircosShiny::run_app()\"" >> /init-shiny

# Make init-shiny file fully readable, writable, and executable
RUN chmod 777 /init-shiny

```

This example demonstrates how we created init-shiny file for running the OmicCircos application.

Since the app is developed using the R Golem framework, we must call the run_app() function and set options beforehand. For non-golem apps, where the command to run the app is runApp(), you would have to set something like:

```

"Rscript -e \"options('shiny.port'=3838, shiny.host='0.0.0.0'); runApp(appDir =
'<appDirName>')\"

```

Lastly, as stated before, the output of the development should be a Docker image pushed to the platform's [internal Docker registry](#) and made **public** (read more on [making your image public](#)).

Below you will find an example of the Dockerfile we used for OmicCircos app deployment.

```

FROM rocker/shiny:4.1.3

RUN apt-get update && apt-get install -y git-core libcurl4-openssl-dev libgit2-dev libicu-dev libssl-dev libxml2-dev make pandoc pandoc-citeproc zlib1g-dev curl && rm -rf /var/lib/apt/lists/*

RUN echo "options(repos = c(CRAN = 'https://packagemanager.rstudio.com/all/_linux_/focal/latest'), download.file.method = 'libcurl', Ncpus = 4)" >> /usr/local/lib/R/etc/Rprofile.site

RUN touch init-shiny

RUN R -e 'install.packages("remotes")'

RUN mkdir /omiccircos

ADD renv / omiccircos

```

```
ADD renv.lock /omicccircos

WORKDIR /omicccircos

ENV
RENV_CONFIG_REPOS_OVERRIDE=https://packagemanager.rstudio.com/all/__linux__/focal/latest

RUN Rscript -e 'remotes::install_version("renv", version = "0.16", force = TRUE);
renv::restore(lockfile = "renv.lock", library = .libPaths())'

ADD . /omicccircos

RUN R -e 'install.packages(".", type = "source", dependencies=FALSE, repos = NULL)'

EXPOSE 3838

RUN echo "options('shiny.port'=3838,shiny.host='0.0.0.0');OmicCircosShiny::run_app()"
>> /init-shiny

CMD R -e "options('shiny.port'=3838,shiny.host='0.0.0.0');OmicCircosShiny::run_app()"
```

Please note that at the end of Dockerfile, we suggest you put CMD. This command will be overwritten by init-shiny command anyway.

The final step in app publishing requires the involvement of our Support team. You should submit a [request](#) and provide us with the information required for publication (will be explained below).

Development in the Data Studio and Custom Environments

If you are developing an app within the Data Studio RStudio environment, for example, you would have access to our SB File system and could read project files or export data to projects.

Moreover, you will be able to save your environment with installed dependencies using the Custom Environments feature. This functionality allows users to save the image of their environment and continue where they left off.

By using Data Studio for development, you do not have to assume the existence of the `sbgenomics` directory and mock it, since it will be present on the instance for accessing the project files or exporting them. Next, by using the Custom environments, you do not have to worry about tracking the required dependencies, since the whole environment would be saved. It is important to note that **only files (and installed packages) outside of the /sbgenomics folder** will be saved in the Custom environment.

However, there are some rules you must follow to save the app project properly and setup everything for publishing:

- By default, when starting a new Data Studio analysis, the working directory on the instance is set to **sbgenomics/workspace**. You can create a project, develop the app within this directory, and save the environment as Custom environment **during the app development**.
- If the interaction with the Platform is needed (importing and exporting files), setting the paths in the app to project-files, output-files or workspace directories should be absolute:
 - /sbgenomics/project-files
 - /sbgenomics/output-files
 - /sbgenomics/workspace
- When you are ready to publish the app, depending on the approach you have chosen, most probably you would have to move the app project directory from the sbgenomics/workspace to the root directory (using sudo terminal commands) for example:
 - `cd /` - go back to root directory
 - `sudo mv sbgenomics/workspace/<myAppName> /<myAppName>`
 - If the app is created via golem, then you would have to build the `tar.gz` file, move it to the root directory and install it like any other package from source.
 - Next, you would have to create a bash file `init-shiny` that would contain the run app command (as explained before).
 - It is important to **set the port to 3838** in the run command. This is **where** the app will be running in the container.
- When everything is ready, you should save the environment as a new Custom Environment.
- Finally, you should create a publishing [request](#) for our Support team specifying the name of the Custom environment you have created that contains your app.

App registration

When you are ready to publish your app, you should submit a [request](#) to our Support team for releasing the app to our Platform and provide the following information:

- The app name
- Description
- Thumbnail image (size 160x160px)
- Recommended instance size (CPU and RAM)
 - You may review the list of available [AWS](#) and [GCP](#) instances.
- Simple (init-shiny) shell script containing the command for running the application copied from the Dockerfile for review
- Dockerfile to review
- Built docker image that is pushed to developer's docker image repository and made **public** (<https://docs.cancergenomicscloud.org/docs/the-cgc-image-registry>) or if built via Custom Environment, name of the Custom environment created
 - The prerequisite is to already have an account on the Platform. Read more about the procedure for [creating an account](#).
- How should it be shared – with specific users, teams, or the entire CGC user base.

Network restrictions information

When deploying R Shiny apps on the Seven Bridges Platform, it is important to be aware of the network access settings in the Platform project that contains your app. These settings determine whether your app can communicate with external network hosts, which can affect its functionality, especially if it needs to upload or download files, contact a licensing server, or install additional libraries.

Network access control allows you to define a more restrictive network access policy per project, thus ensuring high security and compliance standards in the execution environment. As a project administrator, you can define network access permissions for project executions during project creation or afterwards (via the visual interface, or via the API).

There are two available settings:

- **Block Network Access:** By default, new projects are created with network access blocked. This means that your app will be isolated from the internet and any attempts

to connect to external network hosts will be denied. This includes operations such as file uploads or downloads from external sources, interactions with licensing servers, and accessing external libraries.

- **Allow Network Access:** If the project network access setting is configured to allow network traffic, the app can freely communicate with external network hosts without any restrictions. The existing projects created before this feature was introduced may have unrestricted network access unless changed manually by a project administrator.

Execution Settings

Network access

Block network access

Execution within the project won't have network access.

Allow network access

Execution will have unrestricted network access.

Network access permissions control can be used to further optimize the security of your execution environment.

[Learn more](#)

Important Considerations

- If your application requires communication with external hosts or URLs, ensure that the project has network access enabled – inform the users to run the application from a project where network access has been previously granted. Otherwise, the application may not be able to perform the necessary actions.
- For cases where certain external hosts or URLs need to be accessible despite network restrictions, you should state this in your request for our Support team and have these addresses whitelisted. The request will be reviewed to assess whether the external hosts or URLs can be permitted.

Please ensure that your app network access requirements are clearly communicated to users, advising them to adjust their network access as needed in their Platform projects. If you have specific connectivity needs, consult with our Support team to ensure that these requirements are met.

sbShinyModules Package Usage Guidelines

The package includes modules for file selection, plot export, and generic file export, each of which is detailed in the following chapters. Designed specifically to integrate with the Seven Bridges Platform, these modules facilitate seamless interactions between your Shiny application and Platform projects. The file selection module enables users to pick files directly from the Platform project from which the app was initiated, while the plot exporter and generic file export modules allow users to save and export plots and other generated files back to the Platform project. Each module is designed to be easily incorporated into Shiny applications, providing both UI and server-side components that work seamlessly together. Additionally, the package features the **get_all_project_files()** utility function, which enhances file management by efficiently retrieving detailed information about files within a specified directory. In this guide, we will walk you through each module and function, explaining their purpose, configuration, and usage with illustrative examples.

Installation

To install the **sbShinyModules** from [GitHub](#), please use the remotes package. Run the following command in your R console:

```
# Install remotes package if not already installed
install.packages("remotes")
```

```
# Install sbShinyModules from GitHub
remotes::install_github("sbg/sbShinyModules")
```

Note on Dependencies

The sbShinyModules package depends on the xattr package, which in turn requires the **libattr1-dev** system library. If the installation of xattr fails, it could be due to the absence of libattr1-dev on your system. To resolve this, please install libattr1-dev system library using your package manager.

Once you have successfully installed the missing system library, retry the installation of sbShinyModules package using the command provided above.

Key Modules

- **File Picker Module:** Facilitates the selection of Platform project files within a Shiny app, offering options for single or multiple file selections through a user-friendly interface.
- **Plot Exporter Module:** Allows users to save and export plots generated within a Shiny app, supporting various output formats, and integrating with the Seven Bridges Platform for seamless project file management.
- **Generic File Exporter Module:** Provides a versatile solution for saving and exporting diverse types of files from a Shiny app into the Platform project, with support for various file formats and export functions.

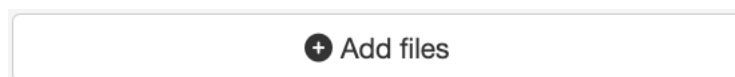
Each module is designed to integrate smoothly with Shiny applications, helping you streamline your workflows and enhance the functionality of your apps. In the following chapters, we will delve into the specifics of each module and function, offering detailed instructions and examples to illustrate their use.

File Picker Module

The File Picker Module is designed to facilitate the selection of Platform project files. It displays a table of available files, allowing users to choose files based on different criteria. It supports both single and multiple file selections and integrates seamlessly with the server-side logic of your Shiny app. By leveraging this module, you can streamline file management tasks, enhance user experience, and ensure that file selection is handled consistently across various parts of the application.

UI Function

The module UI function creates an action button that triggers a modal dialog for file selection.



The UI function also allows you to customize the action button icon (**button_icon** parameter) and width (**button_width** parameter) to suit the needs of your application.

UI function call:

```
mod_file_picker_ui("file_picker_1")
```

Server Function

The server function handles the logic for displaying the file picker modal and managing file selection. It provides features such as table filtering, pagination, and selection modes. The most crucial argument for this function is `files_df`, which is a data frame that developers must prepare and provide. This data frame should include columns for file paths and other relevant metadata.

Parameters:

- **id**: A unique identifier for the module instance.
- **files_df**: A data frame containing the file information. This is the most important argument, and it must include columns for the file paths and any other relevant metadata.
- **selection**: Specifies the selection mode ('single' or 'multiple'). Defaults to 'single'.
- **file_identifier_column**: The column in `files_df` used to identify selected files. Defaults to `path`.
- **default_page_size**: The number of rows per page in the file table. Defaults to 10.
- **use_bslib_theme**: Choose if the bslib theme is used for the modal UI. Defaults to `FALSE`.

Tip: You can use the utility function **get_all_project_files()** to fetch all project files along with their metadata from the Seven Bridges File System (SBFS) that is expected to exist on the instance where the app will be hosted. This function returns a data frame containing comprehensive file information, making it an ideal input for the **mod_file_picker_server()** function.

Note: If your Shiny app utilizes the bslib theme, ensure that **use_bslib_theme** is set to **TRUE** to maintain consistent styling across the modal dialog.

Server function call:

```
mod_file_picker_server("file_picker_1", files_df, selection = "single")
```

Or

```
mod_file_picker_server("file_picker_1", files_df, selection = "multiple")
```

File Picker Modal Dialog

When using the File Picker Module, you need to specify which information should be returned for the selected row(s). This is done through the **file_identifier_column** parameter. By default, `file_identifier_column` is set to *path*, as this is typically the most useful information for subsequent steps in the application. However, you can set this parameter to any other column in the *files_df* data frame that contains the desired information.

It is crucial that the specified column, whether *path* or another value, exists in the *files_df* data frame provided to the module. The module will return a reactive expression containing the information from the specified `file_identifier_column` for the selected file(s).

When the modal dialog for file selection opens, the module displays a preview table of files and allows users to select files based on the specified selection mode ('single' or 'multiple'). Depending on the selection mode, the table will show either radio buttons (for single file selection) or checkboxes (for multiple file selections), enabling users to choose one or more files. Below, you will find examples illustrating both single-file and multi-file picker interfaces.

File Selection

File Selection Guide

To efficiently manage files, follow these steps:

- Review file details.
- To select a file, click the radio button in the leftmost column of the desired file's row.
- Use the search bar above the table to quickly locate a file by its name or metadata.
- Utilize the filter options to narrow down the file list based on specific criteria.
- Click on column headers to sort files in ascending or descending order.
- Use pagination controls at the bottom of the table to navigate through multiple pages of files.
- Once you're ready with your choice, click **Submit** located below the table.

Files Table Preview

Search

	name	id
<input type="radio"/>	ribosomal_RNA_gene.fasta	0544ba8f82ca1ccc0ba259
<input type="radio"/>	Sars_cov_2_ASM98388v3_dna_topline.fa	0544ba8f82ca1ccc0ba259
<input type="radio"/>	GRCh38/CC_ensembl95_transcriptome.fasta	66030c7b50a2c00d06dd174
<input type="radio"/>	genode.v37.transcripts.fa	66030c7b50a2c00d06dd174
<input checked="" type="radio"/>	human_g3k_v37_decay.fasta	0544ba8f82ca1ccc0ba259
<input type="radio"/>	normal_decay.bam	581770d0507c1714734a6124
<input type="radio"/>	example_human_Ulmina.pe_1.fastq	5a20ba65470c79880d0c0e7
<input type="radio"/>	example_human_Ulmina.pe_2.fastq	5a20ba65470c79880d0c0e8

Selected file:

root/public/human_g3k_v37_decay.fasta

X Discard

✓ Submit

File Selection

File Selection Guide

To efficiently manage files, follow these steps:

- Review file details.
- Use the checkboxes in each row to select multiple files simultaneously.
- Use the search bar above the table to quickly locate a file by its name or metadata.
- Utilize the filter options to narrow down the file list based on specific criteria.
- Click on column headers to sort files in ascending or descending order.
- Use pagination controls at the bottom of the table to navigate through multiple pages of files.
- Once you're ready with your choice, click **Submit** located below the table.

Files Table Preview

Search

<input type="checkbox"/>	name	id
<input type="checkbox"/>	ribosomal_RNA_gene.fasta	0544ba8f82ca1ccc0ba259
<input type="checkbox"/>	Sars_cov_2_ASM98388v3_dna_topline.fa	0544ba8f82ca1ccc0ba259
<input type="checkbox"/>	GRCh38/CC_ensembl95_transcriptome.fasta	66030c7b50a2c00d06dd174
<input type="checkbox"/>	genode.v37.transcripts.fa	66030c7b50a2c00d06dd174
<input type="checkbox"/>	human_g3k_v37_decay.fasta	0544ba8f82ca1ccc0ba259
<input type="checkbox"/>	normal_decay.bam	581770d0507c1714734a6124
<input checked="" type="checkbox"/>	example_human_Ulmina.pe_1.fastq	5a20ba65470c79880d0c0e7
<input checked="" type="checkbox"/>	example_human_Ulmina.pe_2.fastq	5a20ba65470c79880d0c0e8

Selected files:

root/public/example_human_Ulmina.pe_1.fastq
root/public/example_human_Ulmina.pe_2.fastq

X Discard

✓ Submit

Table options

Table columns are equipped with various filter options depending on the type of data:

- Numeric Columns:** A range slider filter allows users to narrow down the table by selecting a range of numeric values.

size

30489...3495578872

43669

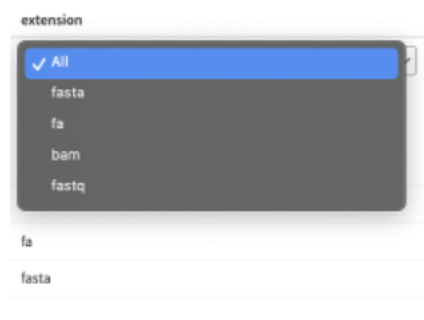
30489

320579869

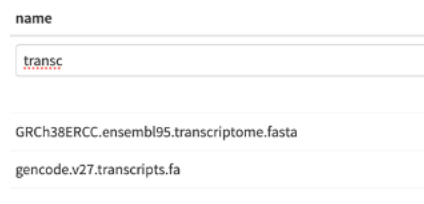
329083254

3189750467

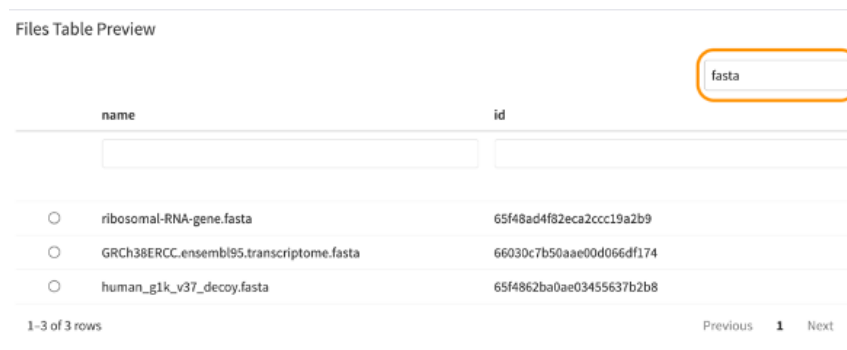
- **Factor Columns:** A drop-down filter enables users to filter the table by selecting specific factor levels.



- **Other Data Types:** A basic filter with case-insensitive text matching is available for other data types.



Additionally, a **general search box** allows users to search the table by entering a term. This search will return all rows that contain the search term in any of their columns.



Check out the example application showcasing the file picker module here: [inst/demos/file_pickers_demo_app.R](#).

Plot Exporter Module

The Plot Exporter Module enables users to save and export plots generated within a Shiny app to the Seven Bridges Platform. This module provides a user-friendly interface for exporting plots in various formats and integrates seamlessly with the server-side logic of your Shiny app.

UI Function

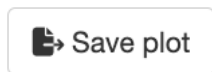
The UI function for the Plot Exporter Module creates an action button that triggers a modal dialog for exporting plots. This modal dialog includes settings for selecting output formats and managing export options.

Parameters:

- **id:** A unique identifier for the module instance.
- **save_button_title:** The label for the button that triggers the export modal.

UI function call:

```
mod_plot_exporter_ui("plot_exporter_1", save_button_title = "Save Plot")
```



Server Function

The server function handles the logic for exporting plots. It integrates with reactiveValues to manage the plot object and provides settings for output formats and directory paths. This function is crucial for enabling users to export plots in various formats and save them to the Seven Bridges Platform.

Parameters:

- **id:** Module's ID.
- **plot_reactVals:** A reactiveValues object containing a *plot* slot with the plot object created using recordPlot(). This is the plot which will be exported.
- **output_formats:** The supported output formats for exporting the plot. Options include "png", "pdf", "svg", "jpeg", "bmp", and "tiff".
- **module_title:** The title displayed in the modal dialog top-left corner.

- **sbg_directory_path:** Path to the mounted */sbgenomics* directory, which should include *project-files*, *output-files*, and *workspace* sub-directories. This directory structure must exist on the instance where the app runs. For local testing, create a mock directory with the same structure and populate it with test files.
- **btns_div_width:** Width of the `div()` containing the buttons for saving plots in the modal dialog. Default is 12.

Server function call:

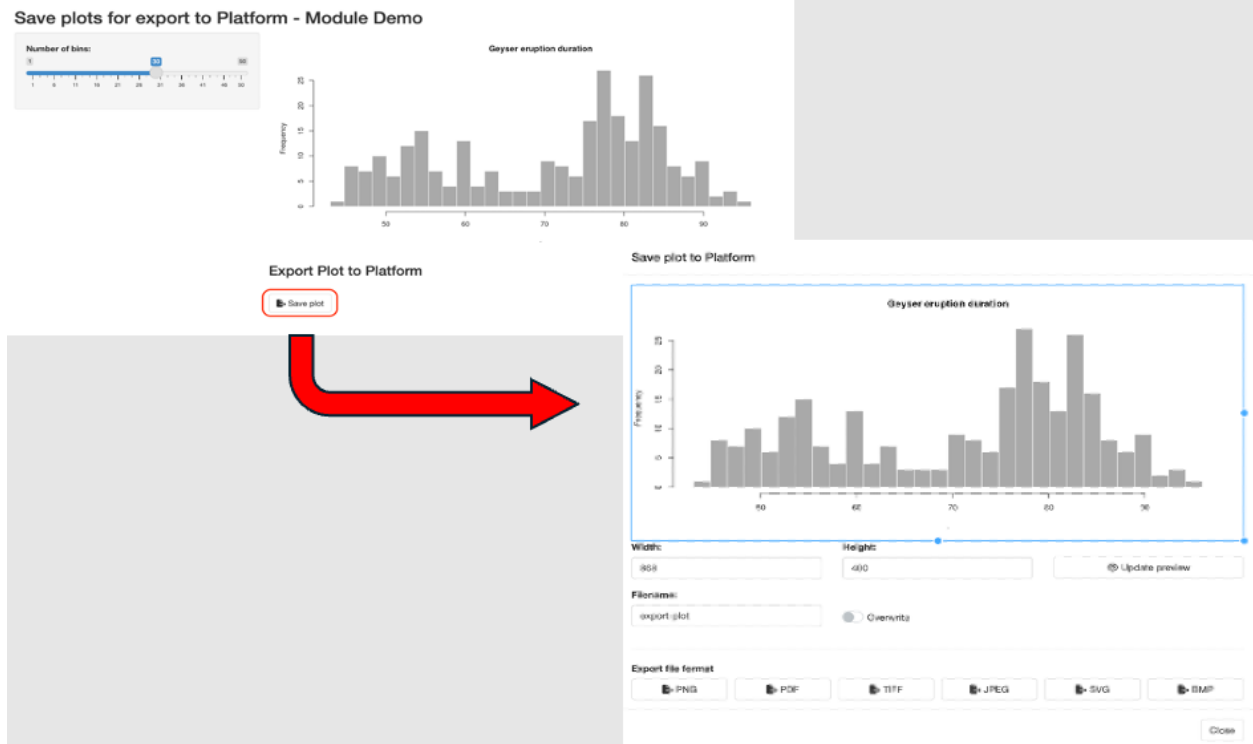
```
mod_plot_exporter_server(
  "plot_exporter_1",
  plot_reactVals = helper_reactive,
  output_formats = c("png", "pdf"),
  module_title = "Export Plot",
  sbg_directory_path = "/sbgenomics",
  btns_div_width = 12
)
```

Note: If you are developing a Shiny application that will be hosted on-demand on the Seven Bridges Platform, the `sbg_directory_path` parameter should be set to the default value of `"/sbgenomics"`. This path points to the */sbgenomics* directory, which already includes the *project-files*, *output-files*, and *workspace* sub-directories. For local testing of your application, you can create a mock *sbgenomics* directory with the same structure—containing *project-files*, *output-files*, and *workspace* sub-directories. Populate this mock directory with test files that replicate the project file structure on the Platform.

Usage Example

To use the Plot Exporter Module in a Shiny app, follow these steps:

- **Add the UI Function:** Place `mod_plot_exporter_ui("plot_exporter_1", save_button_title = "Save Plot")` in the UI section where you want the export button to appear.
- **Add the Server Function:** Include `mod_plot_exporter_server("plot_exporter_1", plot_reactVals = helper_reactive)` in the server section of your app. Ensure that `plot_reactVals` contains the plot object created with `recordPlot()`.



Plot Exporter Module: Modal Dialog Features

When users click the **Save Plot** button, a modal dialog appears, offering several advanced features for customizing and exporting the plot. Here is a detailed overview of these features:

Plot Exporter Modal Dialog

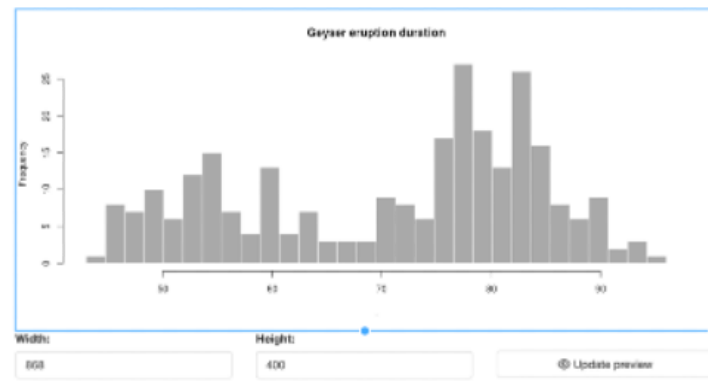
1. Plot Resizing

The modal dialog provides two options for resizing the plot before saving:

- **Interactive Resizing:** Users can drag and resize the plot box using a cursor. This functionality leverages the React Moveable library, allowing users to adjust the plot

dimensions directly. As users resize the box, the plot within it automatically adjusts to fit the new size, and the modal dialog resizes accordingly to accommodate the updated plot dimensions.

- **Manual Resizing:** Users can also specify the plot width and height manually. By entering the desired dimensions and clicking the "Update Preview" button, the plot and modal dialog will adjust to reflect the new size. This option provides precise control over the plot appearance.




2. File Naming and Validation

The modal dialog includes a text input element where users can set the name (base name) for the plot file. Key aspects of this feature include:

- **File Name Validation:** The module performs real-time validation to check if a file with the same name (base name + extension) already exists in the project directory or if it is previously saved within the same session (stored in the output-files). If a conflict is detected, a warning message is displayed near the text input field, prompting users to choose a different name.

Filename:



Please change the name and try again.

- **Overwrite Option:** For cases where users prefer to overwrite an existing file, an overwrite switch is available. This option allows users to save the plot with the same name, regardless of existing files in the project directory.

Filename:

export-plot

 Overwrite

3. File Format Selection

At the bottom of the modal dialog, users can choose from various file formats for exporting the plot. This section includes:

- **Format Buttons:** The dialog features buttons for the default file formats: PNG, PDF, SVG, JPEG, BMP, and TIFF. The specific formats available are controlled by the `output_formats` parameter in the `mod_plot_exporter_server()` function. By default, all six formats are shown, but you can limit the options to a subset of these formats.



- **Saving the Plot:** Clicking a format button triggers the plot export process, using the specified base name and the selected file extension. This action also invokes the file name validation mechanism described above, ensuring that users are alerted to any potential conflicts before saving.

Please Note: By default, plot files are saved in the `/sbgenomics/output-files` directory. Once the Shiny app is stopped, these files will be accessible from the "Files" tab of the project from which the app was launched.

Explore a sample application utilizing the plot exporter module at this location: [inst/demos/plot_exporter_demo_app.R](#).

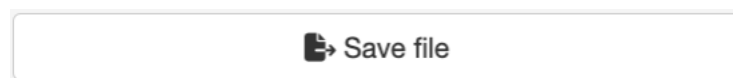
In this example, `helper_reactive` variable is used to store the plot object. The UI function creates the button for triggering the export, and the server function handles the export settings and file saving.

Generic File Exporter Module

The Generic File Exporter Module offers a flexible and versatile solution for saving diverse types of files within a Shiny application. Unlike the specialized Plot Exporter Module, which focuses specifically on plot exporting and includes options for resizing and format selection, this module is designed to handle a broad range of file types and formats, catering to diverse file-saving needs.

UI Function

The UI function for this module creates a button within the Shiny app interface. Clicking this button initiates the file-saving process. This interface is minimalistic, providing a straightforward means for users to trigger file-saving operations.



UI function call:

```
mod_save_file_generic_ui("file_exporter_1")
```

Server Function

The server function of the Generic File Exporter Module is responsible for executing the file-saving logic. When the user clicks the save button, this function performs necessary checks and executes a user-defined file-saving function with specified parameters. The flexibility of this module allows you to define your own file-saving functions and parameters, providing greater control over the file-saving process.

Parameters:

- **id:** A unique identifier for the module instance.
- **reac_vals:** A reactive values list that must include the following fields:
 - FUN: The function used to create the file (e.g., write.table, save, write_json, write_xml, SaveH5Seurat...).
 - args: A list of arguments for the provided function.
 - filename: The name of the file to be saved.
 - extension: The expected file extension for proper validation.
 - overwrite: Boolean indicating if the file with the same name should be overwritten.
- **sbg_directory_path:** Path to the mounted */sbgenomics* directory, which contains *project-files*, *output-files*, and *workspace* sub-directories. This directory structure should mirror the Platform file system for proper integration.

You as a developer are responsible for defining these parameters and integrating them with the module. You can also add additional UI components, such as file name inputs or format selectors, and use their values to populate the parameters provided to the server function.

Note: For testing locally, you can create a mock **sbgenomics** directory with the required structure.

Feedback: Users receive clear alerts regarding the success or failure of the file-saving process, ensuring they are informed of any issues or confirmations.

Server function call:

```
mod_save_file_generic_server(
  id = "file_exporter_1",
  reac_vals = list(
    FUN = write.table,
    args = list(x = my_data_frame, file = "my_file.csv"),
    filename = "my_file",
    extension = "csv",
    overwrite = TRUE
  ),
  sbg_directory_path = "/sbgenomics")
```

Integration

To integrate the Generic File Exporter Module into your Shiny application, follow these steps:

- **Add the UI Function:** Insert `mod_save_file_generic_ui("file_exporter_1")` in the UI section of your app where you want the save button to appear.
- **Add the Server Function:** Include `mod_save_file_generic_server("file_exporter_1", reac_vals, sbg_directory_path)` in the server section of your app. Ensure that `reac_vals` is a properly defined reactive values list.

To see the generic file exporter module in use, refer to the demo app available at: [inst/demos/generic_file_exporter_demo_app.R](#).

Please note that the generic file picker module only adds basic action button to the Shiny app interface. You are responsible for defining additional UI elements, such as filename text inputs, separator selection elements, file extension dropdowns, etc. These elements should be integrated with the module and their values used to configure the parameters passed to the server function.

Additional features available in the package

get_all_project_files() Utility Function

The **get_all_project_files** function provides an efficient way to retrieve detailed information about all files within a specified directory on the Seven Bridges File system. This utility function is ideal for preparing data frames that can be integrated with the File Picker Module, enhancing file selection and management in Shiny applications.

Function Overview

This function fetches all files from a given directory path recursively, including their metadata if available. It constructs a comprehensive data frame containing file names, paths, sizes, and associated metadata, making it ready for use with file picker components.

Parameters:

- **path**: The path to the project-files directory. This is a required parameter specifying the location from which to retrieve file information.
- **...:** Additional parameters passed to the *list.files()* function. Options such as *pattern* and *include.dirs* can be used to customize the file search.

The function returns a data frame with detailed file information, including:

- File names
- File paths
- File sizes
- Metadata fields, if available. The function uses the `get_xattr_df()` function from the `xattr` package to extract metadata for each file.

Integration with File Picker Module

The data frame produced by **`get_all_project_files()`** can be directly used as input for the **`files_df`** parameter in the File Picker Module. This streamlines the process of populating the file picker with accurate and up-to-date file information.

Explore the usage of the `get_all_project_files()` utility function with the provided Rmd script by following this link: [inst/demos/load_files_and_metadata.Rmd](#).

By utilizing **`get_all_project_files()`** function, you can efficiently gather and prepare file information, enhancing the functionality of Shiny applications that involve file management and selection.