New SSF APIs Proposal

Overview

In order for desktop wrappers to properly handle opening, refocusing, and restoring of chat window states, some additional APIs are needed, this proposal covers how we could deliver that functionality while still limiting dependencies between the Symphony Application and any desktop wrapper implementation.

Critical gaps in ssf api functionality are:

- Desktop wrappers must resort to brittle techniques and reverse-engineering to maintain state of pop-out windows.
- From a notification, a chat in the main window cannot be called into focus.

Proposed APIs

ssf.openWindow(url, name, windowOptions)

Summary

This api will allow the desktop wrapper to open a window on behalf of the Symphony Application. While the *window.open* API is a long standard feature in the browser, desktop wrappers often require custom logic for window management. This api allows a desktop wrapper to use its own functions to open a window and create its own hooks on the handling of window state and additional window features.

Expected Behavior

The *ssf.openWindow* api will prompt the desktop wrapper to open a new window using the given URL, name, and specified options. This API follows the shape of the standard browser *window.open* API closely.

For example:

let winOptions = *“chrome=yes,width=662,height=660,centerscreen=yes,menubar=no,toolbar=no,location=no,status=no,scrollbars=no,resizable=yes,titlebar=no”;*

ssf.openWindow("/float.html?floaterId=2&floaterStreamId=J6CFT7GMyrqHetuDmadTE3///qDOzZbKdA==&x=95&y=38", "window2", winOptions);

ssf.setChatFunction(*callback*)

ssf.setChatFunction(function(iDs, isPopout);

Summary

In order to maintain the interface between the desktop wrapper and the Symphony Application within the SSF API, this proposed method would create a hook whereby the Symphony Application would pass a function via the SSF API for the desktop wrapper to use.

The *setChatFunction* API exposes a hook where the Symphony Application can provide the desktop wrapper with a function to initiate a chat. Supported workflows include restoring a previous state of chat windows or responding to a user action - such as clicking on a notification.

Expected Behavior

- On startup, the Symphony Application would call *ssf.setChatFunction* and provide a callback function. This function would accept an array of user ids and a boolean flag indicating if the chat is a pop-out.
- When *ssf.setChatFunction* is called, the desktop wrapper stores the provided function. The function will be called when:
    - Previously stored state is re-hydrated on startup of symphony
    - A notification is clicked on
    - Other cases where the desktop wrapper needs to initiate a chat as the result of some user action
- When the function defined by the Symphony Application is called, it should do one of the following:
    - If the is*Popout* flag is set then the Application will use the *iDs* args to generate a full URL with a streamId and pass this into a call to *ssf.open* to launch the pop-up.
    - If the *isPopout* flag is false, then the Application will do the following:
        - If a corresponding chat is in the main window, the Application will bring the chat into focus in the main window.
        - If a corresponding chat is open but not in the main window, the Application will call *ssf.activate* to bring the popout window into focus.
        - If there is no corresponding chat, a new chat will be opened in the main window.

Workflow Examples

Opening a Pop-out

1. User clicks on the “popout” icon in a chat in the Symphony main window.
2. The Application calls *ssf.open* and passes in the URL for the chat popout and the window name
3. The desktop wrapper stores the window name along with other relevant metadata and opens the window and loads the URL.

Restoring Pop-out State

1. The desktop wrapper retrieves its stored pop-out window state on startup. This would be metadata describing window names, ids for the chats, and window positioning.
2. For each pop-out window, the wrapper calls the handler registered on *ssf.setChatFunction* passing in the identifiers for the chat and true to indicate the chat is a pop-out.
3. The Symphony Application handler resolves the identifiers to a URL to open the chat popout and calls *ssf.open* with the URL, along with window name, etc.
4. The desktop wrapper stores the window name along with other relevant metadata and opens the window and loads the URL.

Bringing a Chat into Focus on a Notification

1. A user clicks on a notification.
2. The desktop wrapper calls the handler registered on *ssf.setChatFunction* passing in the identifiers for the chat.
3. If the chat is in the main window, the Symphony Application will bring the chat into focus and call *ssf.activate()* on the main window.
4. If the chat is in a pop-out window, the Symphony Application will call *ssf.activate()* on the pop-out window with the chat.