Knack Toolkit Library

(v0.1.0 - pre-release)

Contents

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

Introduction

Knack Toolkit Library, henceforth referred to as **KTL**, is a collection of open-source Javascript utilities that eases Knack application development and add several features that are not easily created from the ground up. It is 100% view-based, so your API key is never exposed.

*** NOTE*** The code and documentation are currently in a pre-release phase. Thank you for being patient and understanding that everything may not be complete or perfectly organized.

Features

The code is categorized in different general features:

Here's a quick list:

- Bootloader
- Core
- Storage
- Scenes
- Views
- Fields
- Bulk Operations
- User Filters
- Form Persistence
- Accounts
- User Preferences
- iFrame Window
- Debug Window
- Logging
- Windows Messaging
- System Info
- System Colors

Now, let's go through each one and see what they can do, with the list of all available functions.

Bootloader

The bootloader is the entry point of all code, including KTL and your app-specific code, and does two things:

External library loading

First, I wish to say a big ***thank you*** to **Soluntech** for their gracious permission to use a portion of their code to manage the dynamic library loading. To be honest, I don't fully understand it, but it sure does a wonderful job, and allowed me to start the KTL project. In short, it uses a list of libraries your App will need, and automatically loads them with the Lazyload function. Don't worry, you don't need to understand all of this since the setup is already done.

Normally, without the Bootloader, all your App code resides in the the Builder's Javascript pane. But when using it, you have another cool option of loading **MyAppCode.js** (for example) file locally, from your hard drive, just as if it was another library, being loaded at run-time. This

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

means you can now code and save directly on your workstation, without having to copy/paste the code to the Builder every few seconds, for every change you make.

Developing code locally

This mode enables you (the developer) to work more efficiently by using your favorite code editor with all its bells and whistles, instead of the Builder's Javascript pane. You must install **Node.js** on your computer and run the **NodeJS_FileServer.js** script provided. Then, each time you save your code, all you have to do is refresh the browser to see the changes take effect immediately. In this mode, writing and testing code simply doesn't get any faster.

Another great advantage is that it opens the possibility of teamwork. Currently, only one developer at a time can edit the code. With the bootloader and Node.js server, there is no conflict because each developer works with his own "sandboxed" copy and pulls external changes whenever he/she chooses to do so.

Functions

- Bootloader has no exposed functions.

Core

This contains generic utility functions, not related to any category.

Functions

- **setCfg**: This is where you can enable the features you want.
- **getCfg**: To get the config and read the flags.
- **knAPI**: Knack API wrapper with retries and error management.
- **isKiosk**: For support of kiosk mode applications. You decide the trigger conditions for kiosk mode in a callback function.
- **hideSelector**: To move away elements off the screen to hide or save real-estate.
- **waitSelector**: When you need to wait until an element exists or is visible.
- **waitAndReload**: Waits for a specific delay, then reloads app
- **enableDragElement**: To add basic drag and drop to an element.
- **splitUrl**: Creates an array containing the path and parameters of the URL.
- **getMenuInfo**: Retrieves the menu and sub-menu items.
- **isHex**: For valdation of hex characters only.
- **ipFormatOk**: For IP format validation.
- **getSubstringPosition**: Returns the index of the Nth occurrence of a string within a string.
- **addZero**: Adds leading zeros to 2 or 3-digit numbers, typically for logs alignment.
- **getCurrentDateTime**: Generates a local or UTC date/time string.
- **dateInPast**: Compares the first date to the second one and returns true if it's in the past, but ignore the time component. If second date is not provided, it uses today.
- **selectElementContents**: Selects all element's text.
- **timedPopup**: Generates a brief, auto-delete popup with status text and color.
- **removeTimedPopup**: To remove the timedPopup.

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

- **infoPopup**: Similar to timedPopup, but without a delay. Removal must be done manually. Useful for progress indicator.
- **setInfoPopupText**: To indicate progress in infoPopup.
- **removeInfoPopup**: To remove infoPopup.
- **insertAfter**: To insert a node after an existing one, but as sibling, not as a child.
- **setContextMenuPostion**: Upon right-click, ensures that a context menu follows the mouse, but without overflowing outside of window.

Storage

Provides non-volatile storage utilities using the localStorage and cookies objects. It is the base of most other features.

Functions

- **hasLocalStorage**: Returns whether or not localStorage is supported.
- **lsSetItem, lsGetItem, lsRemoveItem**: Saves, loads and deletes text item in app-specific keys.
- **saveUserSetting, loadUserSetting, setCookie, getCookie, deleteCookie, deleteAllCookies**: Same but using cookies.

Scenes

Provides scene-related features.

Functions

- **setCfg**: To setup your parameters and callbacks to your app.
- **autoFocus**: By default, Knack does not set the focus to a field. But this enables you to choose when and how to do it.
- **addKioskButtons**: In kiosk mode, most of the time there is no mouse or keyboard. This enables adding specific buttons, typically for navigation: Back, Done, Refresh. I've also added Work Shift and Messaging buttons, if ever you need them.
- **spinnerWatchdog**: This is a timer that checks if the App is in a waiting state. If the spinner takes more than a specified amount of time (default is 30s), you can gain back control, typically by reloading the page. Most of the time, this solves the "infinite waiting" problem after a Submit, especially for kiosks without a keyboard, where users would otherwise have to reboot the device.
- **getSpinnerWdStatus**: Returns true if page is busy and spinner is shown.
- **flashBackground**: Simple attention getter, useful on small devices monitored from a distant area, to show status like success or failure.
- **resetIdleWatchdog**: The idle watchdog is an inactivity timer. Each time a mouse click/move or a key press is detected, this is called. After a given amount of time without activity, a callback in your App is called and an action can be taken, like reloading the page.
- **findViewWithTitle**: Returns the first view id containing specific text in its title.
- **scrollToTop**: Scrolls the page all the way up.
- **addVersionNumber**: Adds the App and KTL version numbers on the page.

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

- **onSceneRender**: Callback to your app's handler of a scene render.

Views

Provides view-related features.

Functions

- **setCfg**: To setup your parameters and callbacks to your app. Callback appProcessTitleFlags allows you to process your own special title flags.
- **refreshView**: Robust view refresh function with retries and error handling. Supports most types of views including tables, details, searches, forms, rich text and menus.
- **refreshViewArray**: Calls refreshView in sequence from an array of view ids as parameter.
- **autoRefresh**: You can now add auto refresh to any view without a single line of code. It is done from the Builder, by simply adding AUTOREFRESH=30 at the end of your view's title and it will refresh itself every 30 seconds. Values from 5 (seconds) to 86500 (24 hours) are accepted. Of course, the flag is truncated so only your title remains visible. Also, you can start and stop the process at will.
- **addViewId**: Convenient for developers who want to see the view id next to or near the title.
- **addCheckboxesToTable**: Will add checkboxes to a table, including the top one in the header to check all at once. Used by bulk operations.
- **addTimeStampToHeader**: Useful to see when the last refresh date/time occurred and assess that all is working fine.
- **processTitleFlags**: This is an internal function that is not exposed. But it's worth explaining nonetheless. It parses the view's title for special flags. Here's the list:
    - AUTOREFRESH=*value*
    - HIDDEN_VIEW
    - HIDDEN_TITLE
    - ADD_REFRESH
    - ADD_BACK
    - ADD_DONE
    - NO_BUTTONS
    - NO_INLINE
    - USER_PREFS_CUR
    - USER_PREFS_UPD
    - USER_PREFS_SET
    - ACCOUNT_LOGS
    - UTC_HEARTBEAT
    - USER_FILTERS_MENU
    - USER_FILTERS_CODE

    You can also add your own app-specific flags in the callback function appProcessTitleFlags.
- **hideField**: Moves a field away from the screen to hide it or save space.

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

- **searchDropdown**: Searches text in a dropdown or a multiple choices object, with these options: exact match, show popup for outcome. Supports all 4 states of the dropdown: single selection, multiple selections, less than 500 and more than 500 entries. Auto-selects item if found is an exact match. Otherwise returns found items and lets you manually choose from the results list.
- **findInSearchView**: Uses a Search view to find text, with exact match. Very useful to prevent duplicate entries on a connected field, for example, by doing a hidden search on that view before submitting a new connected record.
- **removeTableColumns**: Will hide or remove columns from a table. Pass it an array of field ids, and/or array of columns indexes to remove. Also works with action links, which is useful to remove a Delete action if the logged-in role shouldn't be allowed for example.
- **modifyTableSort**: Inverts the sort order if the data type is Date/Time. In several apps, I found that users always need to click the header twice because they want to see the most recent entries. You can also do a Ctrl+Click to sort it ascending like it is now.

Fields

Provides field-related features like auto-select all text on focus, convert form text to numeric and enforce numeric validation.

Functions

- **setCfg**: Set a callback to your keypress event handler. Specify which fields must be considered as numeric even though Knack sets them as text.
- **convertNumToTel**: All numeric fields will automatically be converted to telephone type. This has no negative or perceptible impact by all users, except that it allows mobile devices to switch the keyboard to telephone type for a more convenient numeric layout and also auto-selection of all text upon focus.
- **enforceNumeric**: For all numeric fields, plus some specified by user (optional), validation will be performed. If non-numeric values are found, the submit button will be disabled and grayed-out, and the field will be colorized with Knack's "pink" error indicator.
- **addButton**: Will add a button to a specified div element. You can specify the label, style, classes, id, etc., and it will return a button object to which you can attach your event handlers.
- **addCheckbox**: Similar to addButton, but for a checkbox.
- **Barcode reader specific functions**: **addChar, clearBuffer, getBuffer, setUsingBarcode,** and **getUsingBarcode**. Useful in the context of business and industrial projects.
- **addChznBetter**: The chznBetter object is a custom solution that fixes a few problems with the Knack dropdown object. The most annoying being the following: When you have more than 500 items in the list, the dropdown switches mode and offers a search field. But most of the time, when 3 or 4 characters are typed, the last one or two are erased, rendering the selection very tedious. I got so many complaints about this that I decided to code my own solution. The other problems are that you can't set the threshold delay time nor the number of characters to type before the search starts. Now, you have control over both.

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

- **searchChznBetterDropdown**: chznBetter's wrapper to searchDropdown.

Bulk Operations

Provides the ability to perform several record modifications or delete operations in batches. There are two types of bulk operations: Bulk Delete and Bulk Edit. Both work with table views, and have a global flag to enable each of them separately.

Bulk Edit

You must enable the bulkEdit flag in the ktl.core.setCfg function. Then create an account role named "Bulk Edit" and assign it diligently to very trusty and liable users.

For each applicable table, enable Inline editing and be sure to disable all the fields that should be protected against unintended modifications. Where Inline Editing would not be desirable, you can disable it for any given table adding the view title flag **NO_INLINE**. This allows the API to work properly, while disabling the mouse actions.

These field types are supported: all text fields, connected fields, date time picker, checkboxes, radio buttons and multiple choices.

Usage: In the table, select all the checkboxes for the records to be modified. Then click on a cell to edit its value (inline). After submitting the change, a prompt will ask you if the value should also apply to all selected records. Click yes to apply to all. A confirmation message will pop-up after completion.

** Important note** the table's sort may cause your changes to disappear due to being out of scope. This is normal. You can prevent this by first choosing a sort that will not cause this, or filtering as much as possible to show a very restricted number or records, including the ones you need. Experimenting with only a few records at a time (less than 10) is recommended as a starting point. If you've made an error, the process can be interrupted (but not undone) at any time be pressing F5 to reload the page.

Bulk Delete

You must enable the bulkDelete flag in the ktl.core.setCfg function. Then create an account role named "Bulk Delete" and assign it diligently to very trusty and liable users. For each applicable table, a Delete action link must be added. You will see two buttons appear:

- **Delete Selected**: Is enabled when at least one record is selected
- **Delete All**: Is enabled when "Add filters" is used. The checkboxes are ignored, and the process will keep deleting records until none is left, flipping through pages automatically.

Functions

- **enableBulkOperations**: If the user has one of the special roles above, then Bulk Ops are automatically enabled for all table views. Note that Bulk Edit will only be possible when Inline Edit is enabled for the table and for each field of interest.
- **deleteRecords**: To delete and array of records. Used internally by bulk delete, but may be used elsewhere by your App if ever needed.

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

User Filters

When "Add filters" is used in tables, it is possible to save them to a named button. Your configuration is save in localStorage, but can be saved/restored to/from Knack for backup or migration to other devices. The button colors will have matching variations based on the App's theme.

Functions

- **setCfg**: Sets the allowUserFilters callback to your App to allow user filters based on specific conditions.

Form Persistence

When user types-in data in a form, values are saved to localStorage and restored in case of power outage, accidental refresh, loss of network or other mishaps. Data is erased when the form is submitted or user navigates away from page.

Functions

- **setCfg**: To define scenes and fields to exclude.

Account

Provides features for the currently logged-in account.

Functions

- **getRoleNames**: Returns a list of all roles, in text format.
- **isDeveloper**: Check if the list of role names contains "Developer"
- **isLoggedIn**: Returns false if Knack.getUserAttributes() is not "No user found" (not logged-in).

User Preferences

Provides various settings for the currently logged-in account. Some are built-in, and more can be added by your app. You can control which settings can be modified by the user and they can access them in the Account Settings page. See the User Preferences setup procedure below.

Functions

- **setCfg**: Creates a callback allowShowPrefs() where you can control what preferences you give access to specific roles. Typically, this is used to give access to more advanced flags to developers.
- **getUserPrefs**: Reads the user preferences from localStorage.

iFrame Window

Referred to as the **iFrameWnd**, it's a hidden utility page at the bottom of the main App page that contains various views for the support of user preferences, system status and logging features. You may also add your own tables there if you need any. The idea is to be at two places at the

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

same time: The main App page that changes as the user navigates around, and that invisible iFrameWnd that stays with us to serve various functions in the background. When the user logs-in, the authentication token is conveniently shared with the iFrameWnd, allowing us to log-in seamlessly and do API calls. See the iFrameWnd setup procedure below.

Usage

- It is used to monitor the current SW version on all devices, perform remote SW updates, send UTC timestamps called *heartbeats* from devices to the system to assess sanity/presence.
- The user preferences are also read here, for various flags and the work shift.
- A logging table is used to send all logs to Knack via an API call. It contains the 5 most recent logs with a unique identifier (Log ID) to confirm the transaction.
- To enable the iFrameWnd feature, see the procedure described in the section Advanced Features.

Functions

- **setCfg**: Your App calls this at startup to specify the field IDs that are required to do their functions, as taken from the Builder.
- **getCfg**: Returns the iFrameWnd config about field and view IDs.

Debug Window

Provides a window to see local logs on mobile devices where we don't have the luxury of a console log output. Useful for simple tracing/debugging without the complexity of USB tethering and the learning curve that comes with all the tools. Works on all device types (not just mobile), and the window can be moved around. The logs are stored in a ring buffer of 100 elements.

Functions

- **lsLog**: Adds a log to localStorage, with timestamp to millisecond resolution. These logs can be shown in the debugWnd when visible, and optionally, in the console.log if you have one.
- **showDebugWnd**: Show or hide the debugWnd.

Logging

Provides enhanced local logging functions, but also remote recording of user activity information and system status. Logs are aways saved in localStorage, with their timestamp. This is to prevent losing any of them in case of power loss or other reason. At certain intervals, they are sent to Knack and upon confirmation (todo…), they are erased from localStorage.

Logging categories: User login, Navigation, Activity (count of keypresses and mouse clicks), Critial Events, App Errors, Server Errors, Warnings, Info and Debug.

Functions

- **setCfg**: Allows setting a callback logCategoryAllowed() that returns whether or not a category should be logged, based on specific conditions.
- **clog**: Just an enhanced version of console.log(), with colorized, bold font text

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

- **objSnapshot**: Converts an object to a string and back to an object. This is used to *freeze* the content of an object in time.
- **addLog**: Adds a log to the localStorage for deferred processing. All log categories are not created equal. Here's how each work:
    - Critical: Sent to Knack within 1 minute. An email is also sent to the Sysop.
    - Login, Warning, App Error: Sent to Knack within 1 minute.
    - Activity, Navigation, Server Error: Data is accumulated in an object in localStorage, then sent as a single bundle to Knack every 3 hours to reduce API calls.
    - Info, Debug: in progress… to be determined.
- **getLogArrayAge**: Used internally by iFrameWnd and returns the oldest log's date/time from array within a resolution of 1 hour.
- **monitorActivity**: Entry point that starts the user activity logging. Every 5 seconds, the mouse clicks and key presses counters are updated in localStorage, and counters from all opened pages and tabs are merged (added) together.
- **resetActivityCtr**: Resets both counters.

Windows Messaging

Provides a framework that handles messaging between windows. It uses a queue and supports automatic retries and error handling. The windows can be App window, the iFramWnd, or any other window that the App creates and needs to communicate with. This is where your App can implement a heartbeat message that notifies Knack about your account (or device) being online and running properly.

Functions

- **setCfg:** Allows setting a callback to your app's handler of failed message types. (TODO: Allow setting number of retries and msg timeout delay before ack.)
- **send**: To send a msg to a specific window. May contain a payload or not.
- **removeAllMsgOfType**: Cleans up the msg queue of all those of a specific type.
- **processFailedMessages**: Callback to your App to handle failed msg.

System Info

Retrieves information about the operating system, browser, device model, processor, whether or not we are on a mobile device, and public IP address.

Functions

- **getSysInfo**: Returns an object with the mentioned properties.

System Colors

Retrieves information about Knack's colors and generates a few variations for KTL features.

Functions

- **initSystemColors**: Parses the Knack colors and generates a sysColors object.
- **getSystemColors**: Get the sysColors object.

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

- **rgbToHsl**, hslToRgb, rgbToHsv, hsvToRgb, hexToRgb: Various color conversion routines.

How to use KTL

There are two methods to use KTL: ACB and CLS modes. If you're in hurry to test it, and experiment with all the features, you should go for the ACB mode.

ACB Mode - All Code in Builder

This is the traditionnal mode that we're used to, i.e. when all the code resides in the Builder's Javascript pane. In this mode, the Bootloader will load the external library files required, but not your App and KTL's code since they will be included inside the Javascript pane itself.

Pros

- Easy and quick to setup, no need to install anything.
- Other users can always see your changes.
- You can test your code on any device, not limited to your workstation.

Cons

- Slower and more tedious to work when using an external editor, due to the copy/paste of the code required each time you make a change.
- Can be risky if used in production - which means the App is being used in a live and "serious" context - since your development code always take effect immediately. You must have good coding experience and know exactly what you're doing.
- If you want to keep multiple separate files (App and KTL) for independent revision control, it won't be so trivial. Whenever you want to go switch from ACB to CLS Mode (more on this later), you will need to migrate your changes back and forth to each files.

Setup

1) In a code editor, create a new file named *AppName*.js. Using the actual name of your App is strongly recommended. This is because, if ever you want to switch to the CLS mode eventually, the Bootloader will need that the file name matches your App name in order to recognize it.
2) Add code from file KnackBootloader.js.
3) Add code from file KnackToolkitLibrary.js.
4) Add the code from file KTLSetupTemplate.js
5) Copy your existing code from the Builder at the end, where you see ***//My App code*** (between the begin and end markers). Save *AppName*.js.
6) In the Builder's Javascript pane
    a. Wipe all code.
    b. Add code from *AppName*.js.
    c. Locate the ***//*App constants** section and add any const (scenes, views, field IDs) that KTL may need. If not sure, just ignore for now.

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

   d. Locate the **//KTL Setup** section and go through all the flags and settings to match your needs.
   e. Locate the **//KTL callbacks to your App** section and go through each function, adapting them to match your needs.
   f. Locate the **//Setup default preferences** section and go through all the flags and settings to match your needs.
   g. Copy all that code to another file named **ACB_*AppName*.js** and save it. This will be your daily work file for code updates, used for revision control, and when you want to switch from CLS to ACB modes eventually. Keeping this file will save you from going through this merge procedure everytime.
7) In the CSS pane, add the CSS code from file KnackToolkitLibrary.css to yours.
8) Copy all that CSS code to another file named **ACB_*AppName*.css** and save it, for the same reasons as stated above for .js.
9) Open a browser to your Knack App.
10) Check console logs to see if all is looking good.

CLS Mode – Code in Local Server

This mode is for advanced users and provides much faster code-save-test cycles and is typically used for longer stretches of code development, where you won't need to show your results to others until a milestone is reached. It requires the installation of Node.js as a basic local file server that the Bootloader uses to fetch the KTL files and your App's code. The Builder's Javascript pane only contains the Bootloader.

IMPORTANT: Only use this mode in a temporary copy of the production app, where the developers can freely experiment without the fear of serious consequence or disruption. Using it in a production environment is not possible, and would simply defeat the purpose anyways. On the other hand, it is possible at any point in time (though not trivial) to switch back and forth between the ACB and CLS modes. See Switching Modes below.

Pros

- Allows very fast "code-save-test" cycles.
- Allows multi-developer collaboration without conflict.
- Allows keeping separate files, one for the App and two for the KTL (.js and .css). This is useful if you want to add features or debug while maintaining independent revision control on each file.

Cons

- Requires a one-time Node.js installation and setup.
- Requires a temporary copy of the actual project for development.
- Other developers can't see your changes unless they pull/merge your new code with theirs.
- If you are working with collaborators and want to see their updates, you must pull and merge their code with yours.

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

- Other users or clients can't see the updates until you merge all code and switch to the ACB Mode.
- You can't test on devices other than your workstation, running the file server.

Setup

Install **Node.js** on your workstation. Just the basic install, no optional component is needed.

Validate installation by typing **node -v** in a command prompt or terminal window. You should see the version number displayed.

1) In a code editor, create a new file named *AppName*.js. Using the actual name of your App is required for the Bootloader to recognize it.
2) Add the code from file KTLSetupTemplate.js
3) Copy your existing code from the Builder at the end, where you see ***//My App code*** (between the begin and end markers). Save *AppName*.js.
4) In the Builder's Javascript pane
    a. Wipe all code
    b. Add the code from file KnackBootloader.js
5) Copy the KnackToolkitLibrary.js and KnackToolkitLibrary.css files to the folder as per recommendation below (Lib\KTL).
6) Back in *AppName*.js
    a. Locate the **//App constants** section and add any const (scenes, views, field IDs) that KTL may need.
    b. Locate the **//KTL Setup** section and go through all the flags and settings to match your needs.
    c. Locate the **//KTL callbacks to your App** section and go through each function, adapting them to match your needs.
    d. Locate the **//Setup default preferences** section and go through all the flags and settings to match your needs.
7) Open command prompt or a terminal window, go to the **code** folder (see folder sdtructure below) and launch **node NodeJS_FileServer.js**. Each time you refresh your app's page, you will see logs showing the path and file name requested.
8) Open a browser to your Knack App.
9) Check console logs to see if all is looking good.

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

Folder Structure

The following structure is recommended, to keep each apps' code separated, a single set of shared libraries, and everything easy to maintain with a revision control tool like GIT.

```
.code\MyKnackApps\App1\App1.js
                  \App2\App2.js
                  \App3\App3.js
.code\Lib\KTL\KnackBootloader.js
             \KnackToolkitLibrary.css
             \KTLSetupTemplate.js
.code\Lib\SomeOtherCoolLib\CoolCode.js
```

Switching Modes

If you plan to be switching modes at least once in a while, which is likely, you should go with the ACB mode since it is easier to start with and is more versatile.

Don't worry, if this procedure sounds complicated and tedious, it's a bit normal, but it's not that bad either. Typically, I have to switch modes about once or twice a month, and it takes about five minutes. But the effort is absolutely worth it, just to fully benefit from the CLS mode.

From ACB to CLS

Let's assume that you're curently in ACB mode, i.e. you've merged all the code in two files from the procedure above - *AppName*.js and *AppName*.css, you've copy/pasted them to the Builder. You can show your updates to external users, or test on various devices. All fine.

But now, you want to pull the plug and work locally in a sandboxed development copy of the app, using the fast CLS mode.

1) Go to the Builder's Javascript pane and uncomment the three lines below this line: **//Uncomment the three lines below to switch to CLS mode.**
2) Make sure that all file names and paths match your system's, as per CLS mode instructions above.
3) Locate **//End of KTL's Bootloader** and delete all code beyond that and save.
4) You probably want to migrate (and remove) any new KTL-related CSS code to the KnackToolkitLibrary.css file.
5) Run **node FileServe.js**
6) Open your browser to your Knack App.
7) Check console logs to see if all is looking good.

From CLS to ACB

Let's assume that you've been in CLS mode for a while, the files *AppName*.js and *AppName*.css contain your latest and greatest code, and it's time to show the cool updates to your client.

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

1) Migrate any changes you've made during the CLS mode session to their respective ACB counterparts: **ACB_*AppName*.js** and **ACB_*AppName*.css**.
2) Copy the files contents to their respective Javascript and CSS panes.
3) In the Javascript pane, comment the three lines below **//Un-comment the three lines below to switch to CLS mode** to enable ACB mode.
4) Open a browser to your Knack App.
5) Check console logs to see if all is looking good.

Basic Features

KTL offers a set of ready to use, out of the box features that require no setup. They are disabled by default, but you can enable them easily by setting their flag to true in the function **ktl.core.setCfg**, in the **//KTL Setup** section.

Here's the list:

1) **showAppInfo**: Displays the App version number.
2) **showKtlInfo**: Displays the KTL version number.
3) **showMenuInTitle**: Adds the menu to the browser's tab title.
4) **selTextOnFocus**: Selects all text in an input field when a mouse or keyboard sets focus on it.
5) **autoFocus**: When a scene is rendered, a field will be selected to automatically place the focus on it, ready for text input. You can have control of the logic with the callback **function autoFocus()** in the **//KTL callbacks to your App** section.
6) **userFilters**: Allows saving the Add filters settings to a button. See User Filters.
7) **persistentForm**: Allows saving the form data to localStorage. See Form Persistence.
8) **spinnerWatchDog**: Will detect when the spinner runs for too long, based on your timeout value, and allows your App to take action - typically reload the page.

Advanced Features

These features are considered "advanced" in the sense that they are not trivial and require some additional setup. Also, some of them can provide communication between various parts of your App, thus leveraging quite powerful administration features.

Here's the list:

1) **iFrameWnd**
    a. **Heartbeat Monitoring**
    b. **User Preferences**
    c. **Account Logging**
    d. TODO… System Info (SW version, Sys Filters, more…)
2) **Bulk Operations**
3) **User Filters**

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

Setup

Invisible Menu

This shall be your default place for any future utility hidden pages. For now, the iFrameWnd will be its first resident.

1) Create a menu named Invisible Menu.
2) In settings, uncheck Include this page in the Page Menu.

iFrameWnd

Create a new Login Page and give permission to all users. Set Page Name to: **iFrameWnd**. Its URL should automatically be set to **iframewnd**. This page will be the placeholder for the next features. For now, leave it blank as we need to create a few objects first. Now, go back in the Invisible Menu and move the iFrameWnd to it.

Heartbeat Monitoring and SW Update

If your App requires Heartbeat Monitoring to assess an account's presence and generate alarms, or perform remote SW updates, follow this procedure:

1) In the Accounts object, add these fields:
    a. **SW Version**: Type: Short text
    b. **UTC HB**: Type: Date/Time, Date Format: mm/dd/yyyy, Default Date: none, Time Format: military, Default Time: none.
    c. **TimeZoneOffset**: Type: Equation, Equation Type: Date, Date Type: hours, Result Type: number, Equation Editor: currentTime()-{UTC HB}
    d. **LOC HB**: Type: Equation, Equation Type: Date, Date Type: hours, Result Type: Date, Equation Editor: {UTC HB}+{TimeZoneOffset}, Date Format: mm/dd/yyy, Time Format: military
2) In the iFrameWnd page created above, add a Form view that updates the currently logged-in account. Once the view is added, remove all fields, then add: SW Version, UTC HB, LOC HB (set as read-only). Move all 3 fields on a single line to save space. Set the view title to **UTC_HEARTBEAT**. Enable the form's auto reload in the Submit rule.

User Preferences

If your App requires User Preferences (aka settings), there are some already built-in, and you can also add your own. Follow this procedure:

1) In the Accounts object, add a Paragraph Text field named User Prefs.
2) In the iFrameWnd, add a view: Type: Details, For: Logged-in Account. Once the view is added, remove all fields, then add User Prefs. Set the view title to **USER_PREFS_CUR AUTOREFRESH=10**.

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

3) Add a Form view that updates the currently logged-in account. Once the view is added, remove all fields, then add User Prefs. Set the view title to **USER_PREFS_UPD**. Enable the form's auto reload in the Submit rule.
4) Align both view on same row to save space.
5) In your app, locate the function ktl.iFrameWnd.setCfg and set all required fields to match those in your Account object.

Account Logging

If your App requires Account Logging, follow this procedure:

1) Create an object named Account Logs and add these fields:
    a. **Account**: Type: Connection to Accounts, all settings at default.
    b. **Date/Time**: Type: Date/Time, Date Format: mm/dd/yyyy, Default Date: Current Date, Time Format: military, Default Time: Current Time.
    c. **Log Type**: Type: Short Text
    d. **Details**: Type: Paragraph Text
    e. **Log Id**: Type: Short Text
    f. **Log Nb**: Type: Auto-Increment
2) In the iFrameWnd, add a view: Type: Table, For: Account Logs, connected to the logged-in Account.
    a. Once the view is added, remove all fields, then add Log Type, Date/Time, Details, Log ID and an Custom Email action with these settings, as from the screen capture **KTL Account Logs Email Settings.jpg**
    b. Set the view title to **ACCOUNT_LOGS AUTOREFRESH=30**, disable keyword search, enable Inline editing, 10 records at a time, no filter,
    c. Sort by Log Nb: high to low, limit to 5 records.
3) In your app, locate the function ktl.iFrameWnd.setCfg and set all required fields to match those in your Account Logs object.

Bulk Operations

If your App requires Bulk Edit and Bulk Delete operations, follow the procedure described in this section: Bulk Operations.

User Filters

In addition to be able to create named buttons for the User Filters that are save in localStorage, it is possible to upload the settings to Knack and download them back wherever and whenever needed. This can be seen as a backup method, or also to migrate them to other devices. Note that if you migrate filters from one App to another, typically a temporary development copy, some filters will not work due to the record IDs that have changed for connected fields. This is a normal behavior, and the only way to fix this is to redo their settings and save back to the same button name.

To support upload and download, follow this procedure:

1) Create an object named Filters and add these fields:

NORMAND DEFAYETTE
PRESIDENT
nd@ctrnd.com
www.ctrnd.com

   a. **Account**: Type: Connection to Accounts, all settings at default.
   b. **Date/Time**: Type: Date/Time, Date Format: mm/dd/yyyy, Default Date: Current Date, Time Format: military, Default Time: Current Time.
   c. **Filters Code**: Type: Paragraph Text
2) Go to User Pages and edit the Account Settings page.
3) Add a menu named My Settings and move it at the top of the page.
4) Add a link to a new page named Filters Backup and Restore, and enter to edit that page.
5) Add a menu named My Filters and add two links to a URL named Upload and Download. Both URLs are: **javascript:void(0);** This unusual URL is actually a method that means "do nothing" and let the KTL handle click events with special processing.
6) Add a table that displays Filters connected to the logged-in account, with two fields: Date/Time and Filters Code.
7) Enable Inline Editing, remove Title, disable filtering and search, 10 records at a time.
8) In your app, locate the function ktl.userFilters.setCfg and set all required fields to match those in your Account object

Future Improvements

- Use JSDoc to have an adequate auto-generated and detailed API documentation, for each function with parameter description, etc.
- Geofencing and other map-based features, with geo-based events and Google Maps integration.
- The sky's the limit! Let's see what we can come up with…

Conslusion

That's about it for now, thanks for reading this and testing the library. Hope you enjoy it as much as I did writing it.

All code and documentation written by:

Normand Defayette

nd@ctrnd.com

Cortex R&D Inc.

Blainville, Québec, Canada