CloudHaven UI API Programmers Guide

API

Client Script Functions

Overview

CloudHaven provides authentication, user management and a UI-as-a-Service (UIaaS). Organizations can register Applications or Components with CloudHaven. Registered Applications are then available in the “App Store”. A User may subscribe to an application in the App Store by clicking the “SUBSCRIBE” link for that application. Once subscribed that application will appear under “My Apps”. Clicking on an application under “My Apps” will launch that application. If it has one, the applications main menu will supersede the CloudHaven main menu available from the hamburger menu in the upper left of the main CloudHaven title bar.

Organizations

(Applications, Components, Mixins & Groups)

The Organization record has the following fields:

1. Name
   Descriptive Name for the Organization.
2. Id
   A hidden alphanumeric id with no spaces used as the programmatic identifier of this organization.
3. Components URL
   The URL where this organization’s components can be found. A POST handler should be registered at this URL which expects an object containing a “componentIds” member which is an array of component Ids as defined below under the Components Tab.
4. Applications Tab
   An application has the following fields:
   a. Name
      Descriptive name of the application. Appears as the title of the App in the App Store or My Apps.
   b. Application Id
      A hidden alphanumeric id with no spaces used as the programmatic identifier.
   c. URL
      This the URL for the organization’s backend server. This server should be configured to respond to the following http GETs and POSTs at this URL:
      i. Application Pages
         Application pages are retrieved with http GETS from subpaths using the following convention: “/apppages/<page>”, where the first page accessed must be “home”. For example, the home page is then found at “/apppages/home” and if there was a page “secondpage” it would be found at “/apppages/secondpage”.
      ii. Other GETs and POSTS
         The application may define any other GET and POST handlers as needed which can

then be accessed via the CloudHaven client script "_appGet" and "_appPost" functions.

5. Components Tab
   This is where components are registered. It has the following fields:
   a. Name
      A descriptive name for the component
   b. Id
      The programmatic id used to by other applications that want to use the component to refer to that component.
6. Mixins Tab
7. Groups Tab

API

An App "page", accessed via the "/apppages/<page>" subpath (as described above under Organization, Applications Tab, URL, Application Pages), is defined by the "Page" object.

Page Object

The page object contains all the information required to dynamically generate the page as well as customize the CloudHaven frame and main menu. The Page object contains the following properties (note that the properties, dataModel, methods, computed, filters are named the same as the corresponding options of Vue.component):

```
{
 components: [< array of component objects>],
 dataModel: {<dataModel object as would be returned by the Vue dataModel function>},
 methods: {<object specifying methods for this Vue "page" (see "methods" section below)},
 computed: {<object specifying "computeds" for this Vue "page" (see "computeds" section below)},
 filters: {<object specifying "filters" for this Vue "page" (see "filters" section below)},
 watch: {<object specifying "watches" for this Vue "page" (see "watches" section below)},
 mixins: [<array of mixin objects>],
 appFrame: {
  name: <String: Application Name>,
  appBarStyle: {<css styles>},
  appBarTextClass: <String: vuetify text class names>,
  nameTextClass: <String: vuetify text class names>,
  menuItems: [
   {page: "<String: page name>", title: "<String: title of page>"},
   …
  ],
 },
 uiSchema: {
  <object containing nested Vue/Vuetify component specifications
 }
```

components

This property contains an array of "internal" component objects supplied by this App's organization (as opposed to "external" components which are supplied by other organizations). Component objects are the same as page objects except they may have a props property and do not have an appFrame property. See Component Objects below. Components are included in the UI using the "dynamicComponent" (see Special Components section).

dataModel

This is an object containing all the elements of the data model, formatted identical to the object that would be returned by the dataModel function of a Vue component.

methods

The "methods" member of the page object is an object. The properties of that object are method names and the value of those properties can be either a string containing the javascript body of that method or a "function" object containing the following properties:

{
    args: ["arg1", "arg2", …], //an array of function argument names as referenced in the body
    body: "…" //a string containing the javascript for the body of the function
}

For example, the methods "currentDate" and "add" can be defined as:

```
methods: {
    currentDate: "return new Date():",
    add: {
        args:["p1", "p2"],
        body:"return p1+p2;"
    }
}
```

In functions defined as methods, computed, watch or filters, references to objects of the dataModel, computeds and methods must be prefaced with "this." (same as for Vue). However, javascript used inline in component properties need not be prefaced with "this." (again, same as for Vue).

The function body javascript is somewhat sandboxed (interpreted with window.Function.apply, not the risky "eval"). It doesn't allow references to the following objects or functions:

window, document, history, location, navigator, XMLHttpRequest, WebSocket, WindowOrWorkerGlobalScope, XMLDocument, ServiceWorkerContainer, Worker, localStorage

computed

This specifies the computed property of the Vue component options in the same manner as Vue except the functions are specified with a "function" object but with no args property (see methods section above). Note that a computed object can also be defined with an object containing get and set properties which in turn contain function objects for getting and setting the computed property respectively (same as for Vue component options).

filters

This specifies the filter property of the Vue component options in the same manner as Vue except the functions are specified with a "function" object (see methods section above).

watch

This specifies the watch property of the Vue component options in the same manner as Vue except the functions are specified with a "function" object (see methods section above).

appFrame

The appFrame property contains an object whose properties affect the CloudHaven "frame" when the App is started from My Apps, including the style of the title bar, the style of the title bar text, the style of the user's name and the application main menu which replaces the CloudHaven main menu while the App is active.

uiSchema

This uiSchema object is the meat of the page containing a nested hierarchy of UI components including all properties, attributes, slots, events, etc. The component object definition closely parallels the Data Object parameter of the Vue createElement function – it also has the same class, style, attrs, props, domProps, on, nativeOn, scopedSlots, key and ref properties (however, there is no CloudHaven component equivalent for the directives, slot, and refInFor properties). UI Components include all Vuetify components, some CloudHaven-native components, and external and internal components defined with CloudHaven component objects.

Child components of a component are defined either in the "contents" property of a component (either as an array of components or a single component) or in the component's scopedSlot property.

The component object closely mirrors a Vue component object with the exception that v-for directive is replaced with the "loop" property and the v-if/v-else/v-else-if directive family is replaced by the "show" and "omit" properties. Note that there is direct v-show equivalent, however the same effect can be done through manipulation of the display style.

The v-model directive is replaced with the component "vmodel" property.

Any v-on directives are implemented with the "on" property of the component (similar to the on property for the Vue.component data

There isn't currently any CloudHaven equivalent for the following Vue directives:

v-text	
v-html	Can probably be accomplished with templates
v-show	Use css display style instead or "show" and "omit" properties that act like v-if/v-else
v-on	all event handlers are defined in the component "on" property
v-bind	binding is handled by prefacing property names with the ":" character
v-model	replaced with the "vmodel" property
v-slot	all slots are defined in the component "scopedSlots" property
v-pre	
v-cloak	
v-once	

Component properties

- component
  This is the name of the component which references a "special component" (see below), a CloudHaven-native component or a vuretify component

  The following CloudHaven-native components are currently available:

  conversation (the CloudHaven, context-sensitive conversation component)
  dateField (a text field where the text must be in a date format like MM/DD/YYYY)
  fileViewer (for viewing text, pdf, image or docx files)
  (See Additional Components section below)

  The following vuetify components are available ("Component name" is the value to use in the component property and "Vuetify Component" is the Vuetify component that will be rendered):

Component name	Vuetify component
alert	VAlert
autocomplete	VAutocomplete

avatar	VAvatar
badge	VBadge
banner	VBanner
bottomNavigation	VBottomNavigation
bottomSheet	VBottomSheet
breadcrumbs	VBreadcrumbs
col	VCol
button	VBtn
buttonToggle	VBtnToggle
calendar	VCalendar
calendarDaily	VCalendarDaily
calendarMonthly	VCalendarMonthly
calendarWeekly	VCalendarWeekly
card	VCard
cardSubtitle	VCardSubtitle
cardTitle	VCardTitle
cardText	VCardText
cardActions	VCardActions
carousel	VCarousel
carouselItem	VCarouselItem
chip	VChip
chipGroup	VChipGroup
checkbox	VCheckbox
colorPicker	VColorPicker
combobox	VCombobox
container	VContainer
dataIterator	VDataIterator
dataFooter	VDataFooter
datePicker	VDatePicker
dataTable	VDataTable
dataTableHeader	VDataTableHeader
dialog	VDialog
divider	VDivider
expansionPanel	VExpansionPanel
expansionPanelHeader	VExpansionPanelHeader
expansionPanelContent	VExpansionPanelContent
expansionPanels	VExpansionPanels
editDialog	VEditDialog
fileInput	VFileInput
footer	VFooter
form	VForm
hover	VHover
icon	VIcon
image	VImg
input	VInput
item	VItem
itemGroup	VItemGroup
lazy	VLazy
list	VList
listGroup	VListGroup
listItem	VListItem

listItemAction	VListItemAction
listItemActionText	VListItemActionText
listItemAvatar	VListItemAvatar
listItemContent	VListItemContent
listItemGroup	VListItemGroup
listItemIcon	VListItemIcon
listItemSubtitle	VListItemSubtitle
listItemTitle	VListItemTitle
menu	VMenu
navigationDrawer	VNavigationDrawer
overflowButton	VOverflowBtn
overlay	VOverlay
pagination	VPagination
parallax	VParallax
progressCircular	VProgressCircular
progressLinear	VProgressLinear
overlay	VOverlay
overlay	VOverlay
radio	VRadio
rangeSlider	VRangeSlider
radioGroup	VRadioGroup
rating	VRating
responsive	VResponsive
richTextField*	TiptapVuetify
row	VRow
select	VSelect
sheet	VSheet
simpleCheckbox	VSimpleCheckbox
simpleTable	VSimpleTable
skeletonLoader	VSkeletonLoader
slider	VSlider
slideGroup	VSlideGroup
slideItem	VSlideItem
snackbar	VSnackbar
spacer	VSpacer
sparkline	VSparkline
stepper	VStepper
stepperContent	VStepperContent
stepperHeader	VStepperHeader
stepperItems	VStepperItems
stepperStep	VStepperStep
subheader	VSubheader
switch	VSwitch
tab	VTab
tabs	VTabs
tabsItems	VTabsItems
tabItem	VTabItem
tabsSlider	VTabsSlider
textarea	VTextarea
textField	VTextField
timePicker	VTimePicker

timeline	VTimeline
timelineItem	VTimelineItem
toolbar	VToolbar
toolbarTitle	VToolbarTitle
toolbarItems	VToolbarItems
tooltip	VTooltip
treeview	VTreeview
virtualScroll	VVirtualScroll
window	VWindow
windowItem	VWindowItem

* Note: RichTextField is TiptapVuetify. You can get the extensions for it by calling

this._getTiptapExtensions()

For example, the component specification would like the following:

{component: “richTextEditor”, props:{“:extensions”:”_getTiptapExtensions()”}

_getTiptapExtensions currently is “hard-coded” to return the following:

[History, Blockquote, Link, Underline, Strike, Italic, ListItem, BulletList, OrderedList,
[Heading, {options: {levels: [1, 2, 3] }}], Bold, Code, HorizontalRule, Paragraph, HardBreak]

- props
  The “props” component property mimics the “props” property of the Vue createElement Data Object specifying the properties of the object. A props property takes a literal value, however, if the property name is prefaced with a “:” character the value is not taken literally but evaluated as an expression (dataModel objects, computed and methods in this expression do not need to be prefaced with “this.”).

  For example:
  {component:”textField”, props:{outlined: false, label:”First Name”}, vmodel:’formData.firstName’}

  or

  {component:”textField”, props:{outlined: false, “:label”:”firstNameLabel”},
  vmodel:’formData.firstName’}
  (where “firstNameLabel” is a property in the dataModel)

- vmodel
  This component property mimics the Vue v-model directive. The value of the vmodel property must be an object in the dataModel and does not need to be prefaced with “this.”

- attrs
  The “attrs” component property mimics the “attrs” property of the Vue createElement Data Object specifying the normal html attributes of the object (like “id”). An attrs property takes a literal value, however, if the property name is prefaced with a “:” character the value is not taken literally but evaluated as an expression (dataModel objects, computed and methods in this expression do not need

to be prefaced with “this.”).

- class
  The “class” component property mimics the “class” property of the Vue createElement Data Object specifying the css classes for the object. A class property takes a literal value, however, if the property name is prefaced with a “:” character the value is not taken literally but evaluated as an expression (dataModel objects, computed and methods in this expression do not need to be prefaced with “this.”). As with the createElement Data Object class property, the class value can be a string, object, or array of strings and objects.

- style
  The “style” component property mimics the “style” property of the Vue createElement Data Object specifying the css style for the object. A style property takes a literal value, however, if the property name is prefaced with a “:” character the value is not taken literally but evaluated as an expression (dataModel objects, computed and methods in this expression do not need to be prefaced with “this.”). As with the createElement Data Object style property, the style value can be a string, object, or array of strings and objects.

- domProps
  The “domProps” component property mimics the “domProps” property of the Vue createElement Data Object specifying the DOM properties of the object (like “innerHtml”). A domProps property takes a literal value, however, if the property name is prefaced with a “:” character the value is not taken literally but evaluated as an expression (dataModel objects, computed and methods in this expression do not need to be prefaced with “this.”).

- on, nativeOn
  The “on” (and nativeOncomponent property mimics the “on” (and nativeOnproperty of the Vue createElement Data Object specifying the event handlers of the object (like “click”). The value of the “on” (or nativeOnproperty is an object where each property of that object corresponds with an event of that object. Mimicking Vue, eventhandlers can have one or more of the following modifiers: ctrl, shift, alt, meta, stop, and prevent or the keycode modifiers sec, tab, enter, space, up, left, right, down or delete. For example:

  {component:”button”, on:{‘click.stop’:”doButtonAction”}, …}

  The value of a click handler can be one of the following:

  string – then name of a method
  function object
  string expression (gets evaluated – does not require prefacing identifiers with “this.”

- contents
  The value of the “contents” component property can be any of the following:
  - A string
    treated as the template contents for a “template” component or else treated as a text vnode for all other components
  - A CloudHaven component
  - An array of CloudHaven components or strings (the latter treated as text vnodes)

- template
  contains the text string template to be “template-compiled”. See the “template” special component below.

- scopedSlots
  The value of the scopedSlots property contains an object where each of its properties references the name of a scopedSlot applicable to that component. The value of each scopedSlot is a component or array of components with the slot properties injected into the current scope. For example, given a datatable component with items defined by an array of objects containing a “firstName” property:

  ```
  {component:”dataTable”, scopedSlots:{
    item: {
      component:”template”, template:”<tr><td>{{item.firstName}}</td></tr>”
    }
  }
  ```

Component Object

The Component Object is identical to the Page Object except it doesn’t have an “appFrame” property and can contain a “props” property the value of which is in a format that mimics the props of a Vue Component except that the type of a property is given as a String not an Object. For example, use

{type: “String”} instead of {type:String}

or

{type: “Object”} instead of {type: Object}

Special Components

template

The template component provides “html” template-compiled from the value in the “template” property. For example:

{component:”template”, template:”{{someDataModelVar}}”}

dynamicComponent

The dynamicComponent name references an internally (this organizationor externally (different organizationCloudHaven uiSchema-defined component. For example, the component with id “example-component”, which could be either an internal (this organizationor external (different organizationcomponent would have a component specification that looked like:

{component: “dynamicComponent”, organizationId: “example-org”, componentId:“example-component”}

loop

The loop component will iterate the array specified in the dataList property of this component, generating the component specified in the contents property while injecting the current dataList element of that iteration into the current scope.

Client system properties

this._currentUser – the current user

this._moment – the momentjs object that can be used for date manipulations

this._ - the _ object is an instance of lodash available to client script.

Client Script Functions

Application Data Storage

_writeAppData(params, cb)

This function provides simple storage capability for application data. A string value can be saved by table and key and later retrieved with _readAppData.

params:

> **table**: This enables application data to be stored in a specified "table".
>
> **key**: Used to specify a specific record (document) within the "table" for the specified key value.
>
> **dataString**: This is the data value to be stored. To store json objects, simply JSON.stringify the object and store the resulting string.

cb: This is an optional callback function that is executed after the write operation. It expects one parameter containing the results of the write operation. It executes in the Vue scope for the current page (or component).

_readAppData (params, cb)

This function is used to read back application data stored with _writeAppData.

params:

> **table**: This enables application data to be read for this "table" – see corresponding table parameter in _writeAppData).
>
> **key**: Used to specify a specific record (document) within "table". This parameter is optional. Omitting it returns all records (documents) for the specified "table".
>
> **searchOperator**: This parameter is optional. Omitting causes only records (documents) to returned where their key property exactly matches the "key" parameter. Valid values are "startswIth" or "contains".

cb: This is an optional callback function that is executed after the data is read. It expects one parameter which contains the records (documents) found for the given search criteria. It executes in the Vue scope for the current page (or component).

_deleteAppData (params, cb)

This function is used to delete a data record given the table and id (actually, the id is all that's really needed to delete the record but the table is required as an extra precaution).

params:

table: This enables application data to be read for this "table" – see corresponding table parameter in _writeAppData).

id: the internal "_id" of the record to be deleted.

cb: This is an optional callback function that is executed after the data is read. It expects one parameter which contains the results of the delete operation. It executes in the Vue scope for the current page (or component).

Application Communication

_pdfGet(params, cb)

Performs and http GET to the organization backend (similar to _appGetwith a responseType of 'blob').

params:

operationId: this is a subpath that is appended to the application URL to provide the target URL for this GET operation.

cb: This is an optional callback function that is executed after the GET operation responds. It expects one parameter, a Blob containing the pdf data. It executes in the Vue scope for the current page (or component).

_appGet(params, cb)

Performs and http GET to the organization backend with the following headers:

Accept: 'application/json'
Content-Type: 'application/json'

params:

subpath: this is a subpath that is appended to the application URL to make the target URL for this GET operation.

cb: This is an optional callback function that is executed after the GET operation responds. It expects one parameter which is the data portion of the response. It executes in the Vue scope for the current page (or component).

_appGetFile(params, cb)

Performs and http Get to the organization backend at the subpath "/<postId>/<fileId>". The has a "content-disposition" header with an "attachment" value containing the filename. The "content-type" header will indicate the MIME type of the returned file. The data is returned as a Blob.

params:

operationId: this is a subpath that is appended to the application URL to provide the target URL for this GET operation.

fileId: this is the fileId to be used as defined by the application for retrieving the file from the organization backend. It is appended at the end of the path to complete the HTTP GET URL.

cb: This is the callback function that executed after the GET operation responds. It expects one parameter which is the Blob containing the file data returned from the HTTP GET. It executes in the Vue scope for the current page (or component). The data passed to the callback can be used to create an object URL – for example, given the parameter is called "data" an object URL could be created as follows:

```
var objURL = URL.createObjectURL(data);
```

_appPost(params, cb)

Performs and http POST to the organization backend with the following headers:

Accept: 'application/json'
Content-Type: 'application/json'

params:

> **operationId**: the application server can use this to determine which operation to perform.
>
> **postData**: the json object to be POSTed. If postData contains a "files" member the content type of the post will be "multipart/form-data" and postData will be treated as FormData object with "files.<filename>" properties append for each <filename> key in "files".

cb: This is an optional callback function that is executed after the POST operation responds. It expects one parameter which is the data portion of the response. It executes in the Vue scope for the current page (or component).

User Data

_lookupUser(searchSpec, cb)

Searches for **a CloudHaven user** using a filter of the same format as the mongodb find operation filter. searchSpec: The mongodb-style filter used to search for a CloudHaven user. The following fields are available to be searched:

searchSpec has the following optional properties (requires one or the other):

> **email**
> **ssn**

cb: This is an optional callback function that is executed after the function completes. It expects one parameter which is the user record found or null if no match was found. It executes in the Vue scope for the current page (or component).

_usersSearch(searchCriteria, cb)

Searches for a **list** of CloudHaven users given a search phrase in the searchCriteria parameter. It will find any user where the firstName, lastName or ssn contains that search phrase.

searchCriteria: a text phrase to be searched in the firstName, lastName or ssn fields of the user.

dateOfBirth: javascript Date object – specifying this will find all users with this date of Birth.

cb: This is an optional callback function that is executed after this function completes. It expects one parameter which is an array containing the users matching the specified searchCriteria. It executes in the Vue scope for the current page (or component).

_writeUserData(userId, userDataIdToValueMap, cb)

This function is used to write user data to the CloudHaven database. If the user record doesn't exist it will be created.

userId: The internal CloudHaven (mongodb _id) of the user.

userDataIdToValueMap: An associative array keyed by the user data id containing the values to be written:

```
{
  userDataIdA: <value1>,
  userDataIdB: <value2>, …
}
```

cb: This is an optional callback function that is executed after this function completes. It executes in the Vue scope for the current page (or component).

_getUserData(userIds, userDataIds, cb)

This function is used to fetch user data for multiple users given a list of "User Data Ids".

userIds: A list of internal user Ids (from the mongodb user "_id" field).

userDataIds: A list of "User Data Ids" to fetch. These can be core user data ids (email, name, firstName, middleName, lastName, dateOfBirth, ssn, language or non-core user data ids (added automatically by the system as needed).

cb: This is an optional callback function that is executed after this function completes. It expects one parameter which is the result. The result is an associative array keyed by user _id with values that are a nested associative array of user-data-id: value pairs. It executes in the Vue scope for the current page (or component).

Multi-Instance User Data

This is for data like comments to a conversation that can be submitted by any user. They are stored by organization and an application-defined key (multiple instances can be stored for the same organization and key and are distinguished by a "created_at" timestamp (and the internal _id).

_writeMultiInstanceUserData (params, cb)

This writes one instance of the Multi-Instance User Data to the database for the specified owner, organization and key value.

params:

> **owner:** the internal id of the owner of this data.
>
> **organizationId:** the id of the organization.
>
> **key:** an application defined key value to distinguish this set of multi-instance user data from other sets.
>
> **Content:** the content to be written (a string)

cb: This is an optional callback function that is executed after the function completes. It expects one parameter which contains the new record created. It executes in the Vue scope for the current page (or component).

__getMultiInstanceUserData (params, cb)

This function will retrieve the collection of multi-instance user data for a given organization and key.

params:

owner: the internal id of the owner of this data.

organizationId: the id of the organization.

key: an application defined key value to distinguish this set of multi-instance user data from other sets.

cb: This is an optional callback function that is executed after the function completes. It expects one parameter which contains found collection. It executes in the Vue scope for the current page (or component).

__deleteMultiInstanceUserData (params, cb)

This function deletes multi-instance user data give either an _id (single delete) or the pair organizationId and key (multiple deletes).

params:

owner: the internal id of the owner of this data.

_id: id of the record to delete

organizationId: the id of the organization.

key: an application defined key value to distinguish this set of multi-instance user data from other sets.

cb: This is an optional callback function that is executed after the function completes. It expects one parameter which contains the results of the delete operation. It executes in the Vue scope for the current page (or component).

User Files

_getUserFiles(userId, cb)

Fetches all User Files locally stored in CloudHaven for this userId.

userId: the internal id of the user who is the owner of this file.

cb: This is an optional callback function that is executed after the GET operation responds. It expects one parameter which is an array of CloudHaven file objects containing "_id", "name", "fileName" and "mimeType" properties (use _getUserFile to get a files data/contents). The callback executes in the Vue scope for the current page (or component).

_getUserFile(userId, fileId, cb)

Fetches a User File locally stored in CloudHaven for this userId and fileId.

userId: the internal id of the user who is the owner of this file.

fileId: This is the internal Id of the file to be fetched.

cb: This is an optional callback function that is executed after the GET operation responds. It expects one parameter which is the file fetched. The data type of the file will be String for text or docx files (docx converted

to html for display and a File object for pdf or image files. The callback executes in the Vue scope for the current page (or component).

_userFileOperation(params, cb)

This function is add, update or delete a User File record. The parameters for this function are contained in the params parameter:

userId: The internal userId of the owner of the file record.

operation: valid operations are add, update or delete

fileId: the internal Id of the User File record containing the file. Only supplied when updating or deleting.

name: The name of the file. This is the user-friendly descriptive name for the file that may differ from the filename.

fileName: the filename of the file.

fileType: This parameter is required so that CloudHaven knows whether to treat the file as text or binary Blob. If the mimeType is text or the docx mime type, the callback "cb" is called with a String parameter containing the file, otherwise it is a File object.

file: the contents of the file as a String for text or docx-type mimetypes otherwise a File object. Only required when operation is add or update.

cb: This is an optional callback function that is executed after this function completes. It executes in the Vue scope for the current page (or component). It will have one parameter indicating the success or failure of the operation.

Application Navigation

_gotoAppPage(page, appParams)

Applications use this function to navigate to a specific application page.

page: Specifies the page to navigate to. If the application source is set to "App Server" CloudHaven will retrieve the page Object json from the app server. Otherwise, if the source is set to "CloudHaven", the page object json will be retrieved from the corresponding page stored in CloudHaven. If the source is App Server, CloudHaven will make an http GET to the application URL with a subpath of "/apppages/<page>" and will expect a Page Object to be returned which it will use to dynamically generate the application page.

Notification

_showNotification(msg)

Displays the text in the "msg" parameter in the "notification" format in the notification area of the CloudHaven "title bar" at the top of the screen.

_showError(msg)

Displays the text in the "msg" parameter in the "error" format in the notification area of the CloudHaven "title bar" at the top of the screen.

Messaging/Tasks/Workflow

CloudHaven offers the following user-oriented subsystems with the ability to embed Applications from any organization directly in the corresponding content.

- Messaging (like email)
- Tasks/Task Queues
- Calendar

_sendMessage(params, cb)

This function is used to send a message to another CloudHaven user. Messages in CloudHaven can have embedded applications. Messages are marked "read" after users view them. Embedded applications have the ability to be "completed" with associated result content (similar to tasks).

params can have the following properties:

senderId: internal _id of user sending this message

senderEmail: usermame (email) of user sending this message.

recipients: an array of **Recipient** json objects to specify who receives this Message. The **Recipient** object contains the following properties:

> **type**: to, cc, bcc
>
> **email**: CloudHaven username (email) of the person to receive the message

subject: a short subject synonymous with an email subject

message: The text content of message.

application: a json object specifying which application page or component to embed in the Message

> **organizationId**: The organization id of the component or application page to be embedded in the Message.
>
> **applicationId**: The internal id of the application containing the page to be embedded in the Message. Either an applicationId or a componentId must be specified.
>
> **componentId**: [NOT YET IMPLEMENTED] The internal id of the component containing the page to be embedded in the Message. Either an applicationId or a componentId must be specified.
>
> **appConfigData**: An application defined json object containing information needed by the embedded application page.

cb: This is an optional callback function that is executed after this function completes. It executes in the Vue scope for the current page (or component). It will have one parameter containing the internal id of the Message created (which can be used to delete this message (see _deleteMessageOrTask).

_queueTask(params, cb)

This function is used to queue a task to an Organization Group. This task will then appear in the Unassigned Task List for members of this group. Unassigned tasks can be "grabbed" by a user which removes it from the Unassigned Task List for all members of that group and places it in that user's "My Tasks" list. Tasks can be "completed" which moves them from the "My Tasks" list to the Archived Tasks list.

params can have the following properties:

senderId: internal _id of user sending this task.

subject: a short subject synonymous with an email subject

message: The text message to send along with this task.

taskGroup: a json object indicating the group to receive the task. It contains the following properties:

organizationId: The organization to which the group belongs.

groupName: The name of the group

application: a json object specifying which application page or component to embed in the Task

organizationId: The organization id of the component or application page to be embedded in the Task.

applicationId: The internal id of the application containing the page to be embedded in the Task. Either an applicationId or a componentId must be specified.

componentId: [NOT YET IMPLEMENTED] The internal id of the component containing the page to be embedded in the Task. Either an applicationId or a componentId must be specified.

appConfigData: An application defined json object containing information needed by the embedded application page.

cb: This is an optional callback function that is executed after this function completes. It executes in the Vue scope for the current page (or component). It will have one parameter containing the internal id (Task Id) of the task created. The Task Id can be used to delete the task with _deleteMessageOrTask

_setTaskOutcome(params, cb)

This function is used to indicate the disposition of a task. Note that "tasks" can be delivered via an Organization Group with _queueTask or embedded in a message sent with _sendMessage. The embedded application page or component can use this function to indicate completion and set a result message/content. If an explicit task (created with _queueTask then the task is moved from "My Tasks" to "Archived Tasks".

params: has the following properties:

taskId: The internal Id of the CloudHaven Message/Task

resultStatus: Incomplete or Completed

resultMessage: the text message to be set on the originating Message/Task

cb: This is an optional callback function that is executed after this function completes. It executes in the Vue scope for the current page (or component). It will have one parameter indicating the success of the operation.

_deleteMessageOrTask(itemId, cb)

[NOT YET IMPLEMENTED]

Use this function to delete a User Message created with _sendMessage or a task created with _queueTask. This will only succeed if the Message/Task has not already not been read or "grabbed".

itemId: The internal Id of the CloudHaven Message/Task

cb: This is an optional callback function that is executed after this function completes. It executes in the Vue scope for the current page (or component). It will have one parameter indicating the success of the operation.

Calendar

_addCalendarEntry(params, cb)

Adds an entry in the specified user's calendar.

params can have the following properties:

ownerId: internal _id of user whose calendar will receive this event. (Either ownerId or ownerEmail must be specified)

ownerEmail: username (email) of user whose calendar will receive this event. (Either ownerId or ownerEmail must be specified)

title: a descriptive title of this calendar entry.

content: the text, richtext or html content of this calendar entry.

durationType: valid values are "allday" or "timed". "allday" implies one or more days, "timed" is only for a single day (for timed, start and end must be on the same day).

start: if durationType is "allday" this specifies the start date, otherwise if the durationType is "timed" this is the start datetime of the event

end: if durationType is "allday" this specifies the end date, otherwise if the durationType is "timed" this is the start datetime of the event

application: a json object specifying **which application page or component will be used to create custom organization-specific content for this calendar entry**

> **organizationId**: The organization id of the component or application page to be embedded in the calendar event.
>
> **applicationId**: The applicationId of the application containing the page to be embedded in the calendar event.
>
> **appConfigData**: An application defined json object containing information needed to by the embedded component or page (including info specifying the page if the embedded object is an application page instead of a component – it's up to the application to "navigate" to that page).

Misc

_deepGet(object, path)

Retrieves the value at the specified path in the specified object.

_deepSet(object, path, value)

Sets the specified value at the specified path in the specified object.

Additional Components

fileViewer

The fileViewer component can be used to view docx, pdf, image or text files. Image files have some scrolling and zooming capability.

props:

mimeType: {type: String, required: true},
filename: {type: String}, //optional
dataString: String, //only applies to text and docx files
dataBlob: undefined, //only applies to image and pdf files
label: {type: String}, //to show an optional label
width: {type: String, default: '1000px'} //set the width of the viewing space

mimeType must be one of the following:

application/vnd.openxmlformats-officedocument.wordprocessingml.document (docx)
pdf
image/* (valid web images)
text/plain'

dateField

This is a simple text date input field that takes a Date object as an input property.

props:

value – a Date object
label – a text label

conversation

This is a simple conversation component where the added comments are saved for the specified user.

props:

organizationId – the internal id of the organization this conversation should be associated with.
application – This is an object containing applicationId and page properties that can be used to associated a conversation with a specific application or even page.
topic – the conversation can also be associated with an arbitrary topic as well