NextGen TV Run3TV Implementation Guidelines: Run3TV SDK APIs

Version: 2.0
Date: Dec 15th, 2023
Doc: R3TV-IG-0203

Framework version: 2.2

Revision history

Version	Date	Framework version	Update
v1.0 (Final)	15 December 2023	Version 2.2	Initial release covering Framework version 2.2
v2.0 (Final)	15 December 2023	Version 2.2	Updates and clarifications

Copyright notice

Contents

1. Glossary

Glossary of unfamiliar words and acronyms.

Term	Definition

2. References

ID	Publisher	Document
[A/331:2023-10]	ATSC	ATSC Standard: “Signaling, Delivery, Synchronization, and Error Protection The version depends upon the targeted receiver
[A/344:2023-03]	ATSC	ATSC Standard: “ATSC 3.0 Interactive Content” A/344:2023-03 28 March 2023
[A/344:Various]	ATSC	ATSC Standard: “ATSC 3.0 Interactive Content” A/344 The version(s) depend upon the targeted receiver. Any section references or examples provided used in relation to this reference relate to [A/344:2023-03]
[R3TV-IG-0204]	Run3TV	NextGen TV Run3TV Implementation Guidelines: Run3TV Application Management
[R3TV-IG-0231]	Run3TV	NextGen TV Run3TV Implementation Guidelines: RSS Feed Specification
[R3TV-IG-0234]	Run3TV	NextGen TV Run3TV Implementation Guidelines: DASH Event Streams Specification
[R3TV-IG-0250]	Run3TV	NextGen TV Run3TV Implementation Guidelines: A344 Emulator
[AEA-IT-024r31]	ATSC	ATSC Implementation Team Document ATSC 3.0 Advanced Emergency Information System Implementation Guide. Available at: https://www.atsc.org/wp-content/uploads/2020/03/AEA-IT-024r32-Advanced-Emergency-Information-Implementation-Guide.pdf
[R3TV-IG-201]	Run3TV	RUN3TV Implementation Guidelines: IOP for Receivers

3. Introduction

The A3FA Framework provides the following features for a common platform to launch ATSC 3.0 Broadcast Applications.

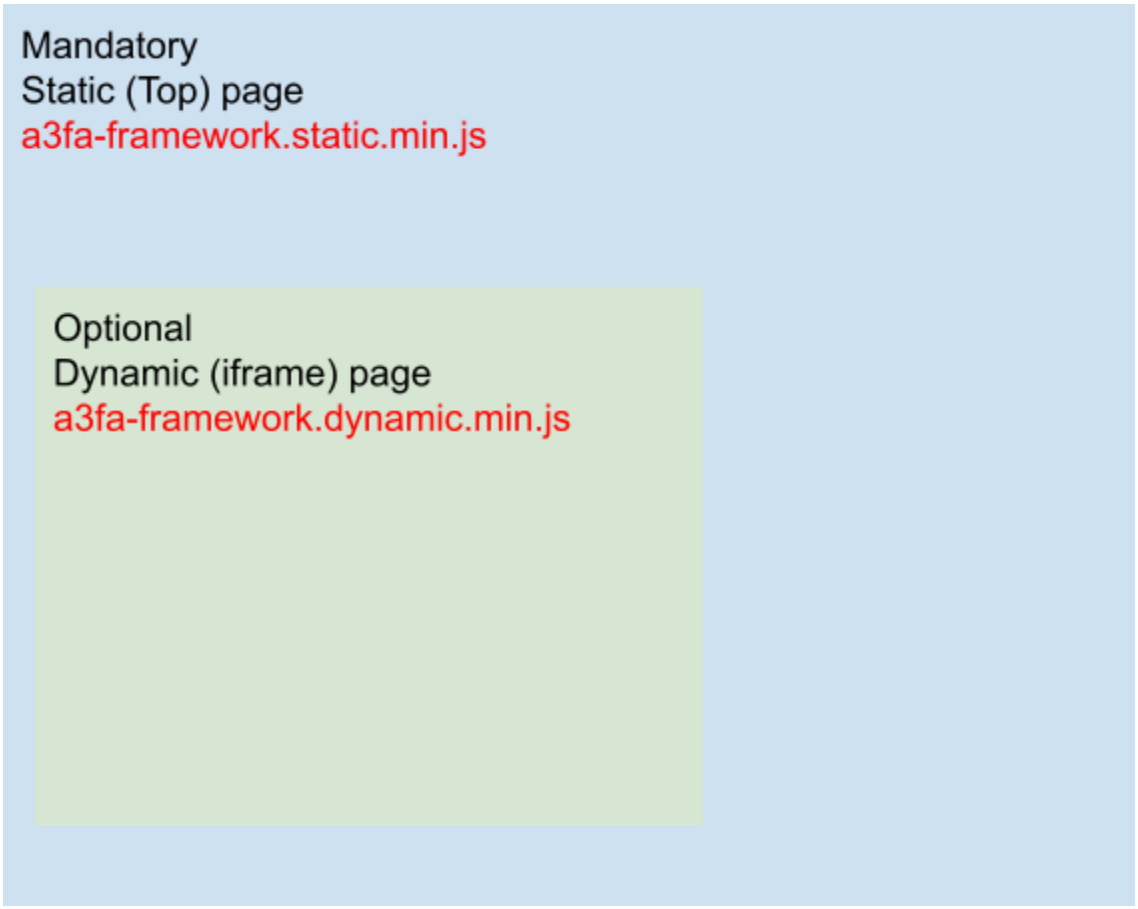
- Integration with ATSC3.0 Receivers, using secure web sockets and standards based API (A/344).
- Common toolset for creating menus with consistent look and feel.
- Flexible Branding, with ability to import different color schemes, skins, and logo images.
- Modular approach - Use functionality that is required, add or extend modules for future needs. e.g. DRM protection modules.

This document describes the SDK interface for v2.2 of the Framework.

4. Application Structure

The framework application typically consists of two pages, the top (static) page and a dynamic (iframe) page as a child.

The Static page orchestrates all of the Framework functions and processes, whilst the dynamic page provides the entry point and interfacing of the Broadcaster Application with the framework.



5. Kickstarting the development process

This section provides general development guidance for Run3TV framework applications.

5.1. How to best make use of this document

To get the most out of this document, it is recommended to try out the examples using the Run3TV emulator and starter kit.

Please refer to the Step by step guide for installing the emulator and starter kit.

Throughout this document you will find complete examples that can be copied and pasted into your browser's Dev Tools console. It is recommended to play with each example to understand how it works.

☐ If you find that the emulator is running, but the applications do not load, ensure that any ad-blocking plug-ins on your browser are disabled

Once you have the emulator and starter kit up and running on your development machine, follow these steps to use the example JavaScript snippets:

1. Open *Dev Tools* by right-clicking on the page and selecting *Inspect*.
2. Click on *Console*.
3. Make sure that the Console's context is set to the **`ifr`** value (below `run3tv-common/`). If it is not, you can change it at the top of the Console pane.
4. You can now copy and paste any examples in this document directly into the console to discover how each method and event can be used.
   Consider running the functions described in the Debug related section first to understand how to use the on-screen Framework log.

Please refer to the Run3TV Emulator documentation [R3TV-IG-0250] for more information.

5.2. Targeting specific web standards

The various web standards (HTML, JavaScript, etc.) versions to target depend upon the receiver base of the television market(s) that your application will be deployed to.

Over time, the ATSC 3.0 standard has moved to keep up with browser standards. As such, older receivers may not offer the same functionality as more modern receivers.

The table below shows a highly simplified summary of the web standards supported by older and newer devices.

ATSC 3.0 Spec	Year of publication	Required CTA Specification	EMCAScript version
A/300:2019	2019	CTA-5000	ECMAScript Language Specification, Edition 5.1
A/300:2020	2020		
A/300:2021	2021	CTA-5000-B	ECMAScript Language Specification, Edition 7
A/300:2022-04	2022		
A/300:2023-03	2023	CTA-5000-D	ECMAScript 2021 Language Specification

☐ Application developers are strongly encouraged to review the linked ATSC specifications (and specifications referenced therein) to better understand the web development feature-sets available.

The CTA-5000 range of documents describe in detail the features expected to be available on receivers of that era.

The Field-Test application (part of the Run3TV framework) can be used to analyze the make-up of receivers in the market to better understand how best to target web standards.

Please speak with your Run3TV representative if you have any questions.

6. API Summary

The table below summarizes the Framework's API functions and events.

API Section	Methods	Events

Framework, Application and Browser related	`fmw.appName()`
	`fmw.baseURI`
	`fmw.input.digits()`
	`fmw.input.preference()`
	`fmw.input.settings()`
	`fmw.isloaded()`
	`fmw.notify.send()`
	`fmw.package()`
	`fmw.getState()`
	`fmw.setState()`
	`fmw.getTopLocation()`
	`fmw.top()`
		`onload()`
		`onfocus()`

Debugging related	`fmw.list()`
	`fmw.log()`
	`fmw.info()`
	`fmw.warn()`
	`fmw.error()`
	`fmw.assert()`
	`fmw.navConsole()`
	`fmw.clearLog()`
	`fmw.rpcLogs()`
	`fmw.dateRND()`
	`fmw.readyState()`	

Websocket	`fmw.rpc.disConnect()` `fmw.rpc.connect()`	`onopen()` `onclose()`

Navigation	`fmw.keyCodeSwitch()`
	`fmw.navClick()`
	`fmw.navNumeric()`
	`fmw.navNext()`
	`fmw.navBack()`
	`fmw.navUp()`
	`fmw.navDown()`
	`fmw.navEnter()`
	`fmw.navExit()`
	`fmw.navHome()`
	`fmw.navQuit()`
	`fmw.BAAppear`
		`onNavEnter()`
		`onNavNext()`
		`onNavBack()`
		`onNavUp()`
		`onNavDown()`
		`onNavExit()`
		`onNavHome()`
		`onNavQuit()`
		`onNavClick()`
		`onNavNumeric()`

Device and viewer related	`fmw.ccEnabled()`
	`fmw.getDevice()`
	`fmw.deviceInput()`
	`fmw.languages()`
	`fmw.language.*()`
	`fmw.location.*()`
	`fmw.timeZone()`
		`onUiLang()`
		`onCaptionState()`

Application launch and selection related	`fmw.load()`
	`fmw.loadApp()`
	`fmw.launchApp()`
	`fmw.unload()`
	`fmw.appPrev()`
	`fmw.appBack()`
	`fmw.appEvent()`	

API Section	Methods	Events

Internal storage and Inter-app communication related	`fmw.setItem()`
	`fmw.getItem()`
	`fmw.removeItem()`
	`fmw.set()`
	`fmw.get()`
	`fmw.del()`
	`fmw.setData()`
	`fmw.getData()`
	`fmw.removeData()`
	`fmw.clear()`	

Navigation related	`fmw.acceptedKeys()`
	`fmw.requestKeys()`
	`fmw.relinquishKeys()`
	`fmw.requestMouseEvents()`
	`fmw.relinquishMouseEvents()`	

ESG related	`fmw.programs()` `fmw.serviceGuide()`	

Broadcast/IP service related	`fmw.rmpAudioTracks()`
	`fmw.audioTracks()`
	`fmw.trackSelect()`
	`fmw.rmp.start()`
	`fmw.rmp.play()`
	`fmw.rmp.pause()`
	`fmw.rmp.stop()`
	`fmw.rmp.resume()`
	`fmw.rmp.scale()`
	`fmw.rmp.time()`
	`fmw.rmp.state()`
	`fmw.queryService()`
	`fmw.acquireService()`
	`fmw.serviceConf()`
		`onServiceChange()`
		`rmpMediaTimeChange()`
		`rmpPlaybackStateChange()`
		`rmpPlaybackRateChange()`

Events stream related	`fmw.events.subscribe()`
	`fmw.events.unsubscribe()`
	`fmw.events.list()`
		`onStreamEvent()`
		`onAppEvent()`

Streaming content (AMP) related	`fmw.amp.start()`
	`fmw.amp.play()`
	`fmw.amp.stop()`
	`fmw.amp.state()`
	`fmw.amp.getCurrentTime()`
	`fmw.amp.setCurrentTime()`
	`fmw.amp.setposter()`
	`fmw.amp.getposter()`
	`fmw.amp.loop()`
	`fmw.amp.isLoop()`
	`fmw.dai()`
		`ampMediaTimeChange()`
		`ampPlaybackStateChange()`

Alerting related	`fmw.aeat()` `fmw.alerting()`	`onAlert()`

Data processing related	`fmw.ajax()`
	`fmw.mediaParse()`
	`fmw.legals()`
	`fmw.tagReplace()`	

7. Framework, Application and Browser related

appName()

The `appName()` method of `fmw` returns the name of the currently running application.

Syntax

```
fmw.appName(callbackFn(appName));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

`appName <String>`

The name of the currently running application, as defined in the appsList.json file for the service. For more information, see [R3TV-IG-0204].

Return value

None

Description

`appName()` is a function of `fmw`.

This function returns a callback containing a new string of the application's `appName`, as defined in the appsList.json file for the current service. For more information, see [R3TV-IG-0204].

Basic usage

The example below prints the name of the currently running application to the console log.

```
fmw.appName(function (result) {
  console.log("App name is: " + result);
});

// App name is: Q-Bar
```

baseURI

The `baseURI` property of `fmw` contains the base URI of the Framework.

Syntax

```
baseURI;
```

Description

`baseURI` is a String property of `fmw`.

This property contains the base URI of the Framework.

For NRT (Non-RealTime broadcast) applications, this value is determined by the `org.atsc.query.baseURI` ATSC 3.0 JSON-RPC message.

For IP-supplied applications, this value is calculated using the following logic:

```
(window.location.origin + window.location.pathname).split("/run3tv-common")[0]
```

When using the emulator, the baseURI can be configured in part using the a3fa-held field inside the **emulator/atscCmd*.mc.json** configuration file

Basic usage

The example below prints the base URI to the console log.

```
console.log("The base URI is: " + fmw.baseURI);

// The base URI is: http://localhost:5001/tv%3Aa3fa-apps.yottamedialabs.com
```

getState()

The `getState()` function of `fmw` returns the current state of the currently running application.

Syntax

```
fmw.getState(callbackFn(state));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

> **`state <Boolean>`**
>
> A new string containing the current state enum of the currently running application. The various states are described below.

Return value

None

Description

`getState()` is a function of `fmw`.

This function returns a callback containing a new string of the application's current state. States include:

- `loaded` - The application has loaded, but has not yet displayed anything on screen
- `unloaded` - The application is no longer available
- `visible` - The application is visible to the viewer
- `hidden` - The application was previously visible, but is not not visible to the viewer

Basic usage

The example below prints the state of the currently running application to the console log.

```
fmw.getState(function (result) {
  console.log("State is: " + result);
});

// State is: visible
```

setState()

The `setState()` function of `fmw` sets the current state of the running application.

Syntax

```
fmw.setState(newState,
             callbackFn(state)
            );
```

Parameters

`newState <Boolean>`

The new state for the application. If set to true, the application will become visible; otherwise it will be made invisible.

`callbackFn`

A function that will be executed once the state has been set. This function has the following parameters:

> **`state <Boolean>`**
>
> A new string containing the current state enum of the currently running application. The various states are described in getState().

Return value

None

Description

`setState()` is a function of `fmw`. It allows the application to inform the Framework of the visible state of the currently running application.

Applications should use this method when moving to and from being visible and hidden.

Basic usage

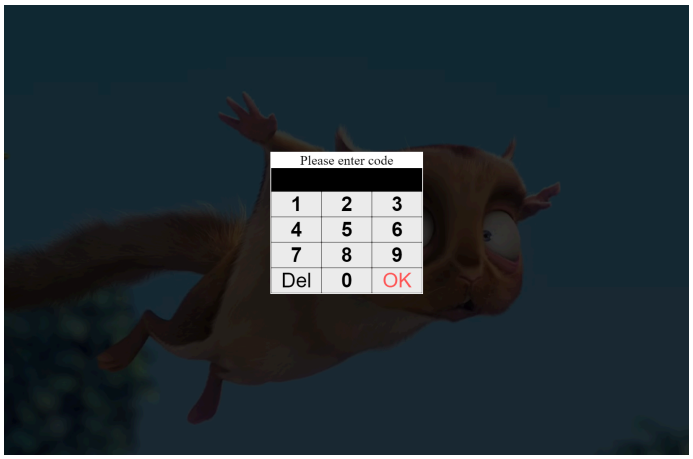
The example below sets the state of the currently running application to invisible.

```
fmw.getState(function (state) {
  console.log("State is: " + state);
});

// State is: visible
```

input.digits()

The `input.digits()` method of `fmw` presents a pin-entry dialog box on screen (as shown below) and accepts pin entries from the viewer.



Syntax

```
fmw.input.digits( /* optional */ messageText,
                                  callbackFn(userInput, error));

fmw.input.digits( /* optional */ messageText,
                  /* optional */ validPINregex,
                                  callbackFn(userInput, error));
```

Parameters

`messageText <String>`

The title text to present at the top of the text entry box

`validPINregex <RegexObject>`

A regex object describing valid pin value(s).

`callbackFn`

If a `validPINregex` parameter is present, this function will be called once the viewer either enters a valid PIN, or they select the “Del” button with no numbers entered on screen.

If a `validPINregex` parameter is not present, this function will be called once the viewer either selects the “OK” button or selects the “Del” button with no numbers entered on screen.

Parameters

`userInput <String>`

The digits entered by the viewer. If the viewer exits the PIN entry dialog without entering a number, this will be an empty string.

`error`

If the popUp object has not been set in **fmw.json**, an error will be returned.

If this function is called a second time without closing the dialog, an error will be returned.

Otherwise null.

Return value

None

Description

`input.digits()` is a function of `fmw`. It presents a PIN entry popup and manages user input. Viewers can enter any number of digits, remove digits and press OK when ready to submit the PIN.

To make use of this functionality, the popUp object must be set correctly in **fmw.json**.

This function can handle PIN validation using the optional `validPINregex` parameter. Alternatively, the application developer can choose to handle PIN validation themselves by not including this parameter.

No matter whether `validPINregex` is present, On leaving the PIN entry popup, the callback function is run including the PIN entered.

Basic usage

The example below presents the PIN entry screen with a bespoke title. Only “1337” is a valid PIN.

```
fmw.input.digits("Please enter secret code",
                 "1337",
                 function (userInput, error) {
                        console.log("PIN entered:" + userInput + ". Error:" + error);
                 });
```

The example below presents the PIN entry screen with a bespoke title. Any four-digit number ending in an 8 is a valid PIN.

```
fmw.input.digits("Please enter secret code",
                 new RegExp("\\d\\d\\d8"),
                 function (userInput, error) {
                         console.log("PIN entered:" + userInput + ". Error:" + error);
                 });
```

input.preferences()

The `input.preferences()` method of `fmw` presents the viewer preferences dialog box on screen.

Syntax

```
fmw.input.preferences( /* optional */ selectedSection,
                       /* optional */ sectionToPresentArray,
                       /* optional */ callbackFn);
```

Parameters

`selectedItem <String>`

The section of the Preferences dialog to first highlight

`sectionToPresentArray <Array of Strings>`

The sections of the Preferences dialog to present on screen

`callbackFn`

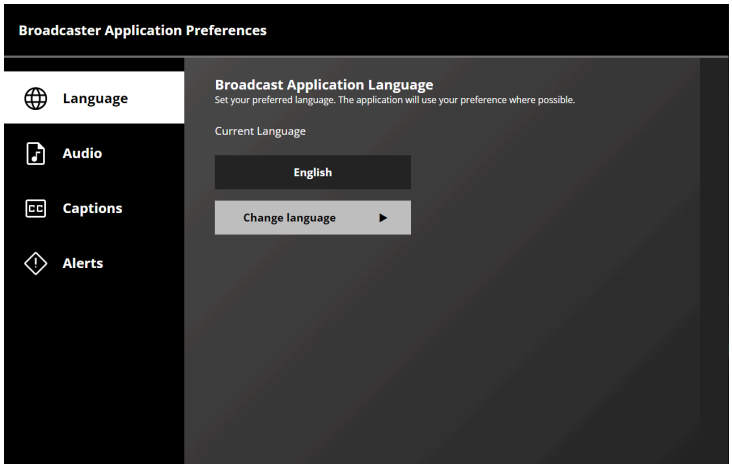
A function that is run when the viewer exits the preferences dialog. This function is called with no arguments.

Return value

None

Description

`input.preferences()` is a function of `fmw`. It presents the Broadcaster Application Preferences dialog box on screen, as shown below.



By including the optional parameters `selectedItem` and `itemsToPresentArray`, it is possible to present the viewer with a customized version of this dialog box.

The permitted values for `selectedItem` and `itemsToPresentArray` are:

- `language`
- `audio`
- `caption`
- `alert`

The dialog will present the menu items in the order defined by `itemsToPresentArray`. If this parameter is not included, all menu items will be presented in the order shown above.

If `itemsToPresentArray` is not included, the complete list of preferences items will be displayed as shown in the earlier screenshot.

If `selectedItem` is not included, the first menu item on the preferences screen will be highlighted at first.

If `selectedItem` contains a value not present in `itemsToPresentArray`, then `selectedItem` will be ignored and the first menu item on the preferences screen will be highlighted at first.

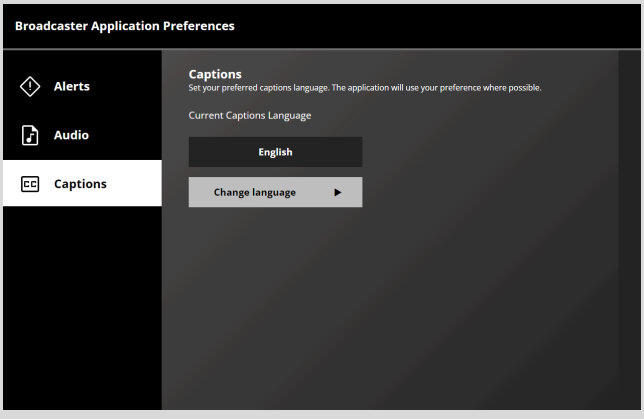
If `itemsToPresentArray` is an empty list, then all preferences menu items will be displayed in their default order.

Basic usage

The example below presents the viewer with three preferences sections and highlights the “Captions” section.

```
fmw.input.preference("caption", ["alert", "audio", "caption"]);

// Results in:
```



input.settings()

The `input.settings()` method of `fmw` presents the viewer settings dialog box on screen.

Syntax

```
fmw.input.settings( /* optional */ selectedSection,
                    /* optional */ sectionToPresentArray,
                    /* optional */ callbackFn);
```

Parameters

`selectedItem <String>`

The section of the Settings dialog to first highlight

`sectionToPresentArray <Array of Strings>`

The sections of the Settings dialog to present on screen

`callbackFn`

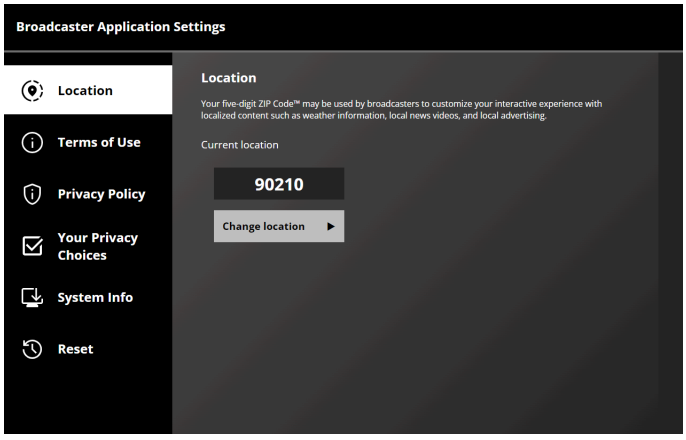
A function that is run when the viewer exits the preferences dialog. This function is called with no arguments.

Return value

None

Description

`input.settings()` is a function of `fmw`. It presents the Broadcaster Application Settings dialog box on screen, as shown below.



In a similar way to input.preferences(), input.settings() allows you to configure which sections are presented to viewers, and which section is highlighted first.

By including the optional parameters `selectedItem` and `itemsToPresentArray`, it is possible to present the viewer with a customized version of this dialog box.

The permitted values for `selectedItem` and `itemsToPresentArray` are:

- `location` - Location information
- `terms` - Terms of use
- `privacy` - Privacy Policy
- `analytics` - Your Privacy Choices
- `system` - System Info
- `forget` - Reset all Framework (and associated application) settings

The dialog will present the menu items in the order defined by `itemsToPresentArray`. If this parameter is not included, all menu items will be presented in the order shown above.

If `itemsToPresentArray` is not included, the complete list of preferences items will be displayed as shown in the earlier screenshot.

If `selectedItem` contains a value not present in `itemsToPresentArray`, then `selectedItem` will be ignored and the first menu item on the preferences screen will be highlighted at first.

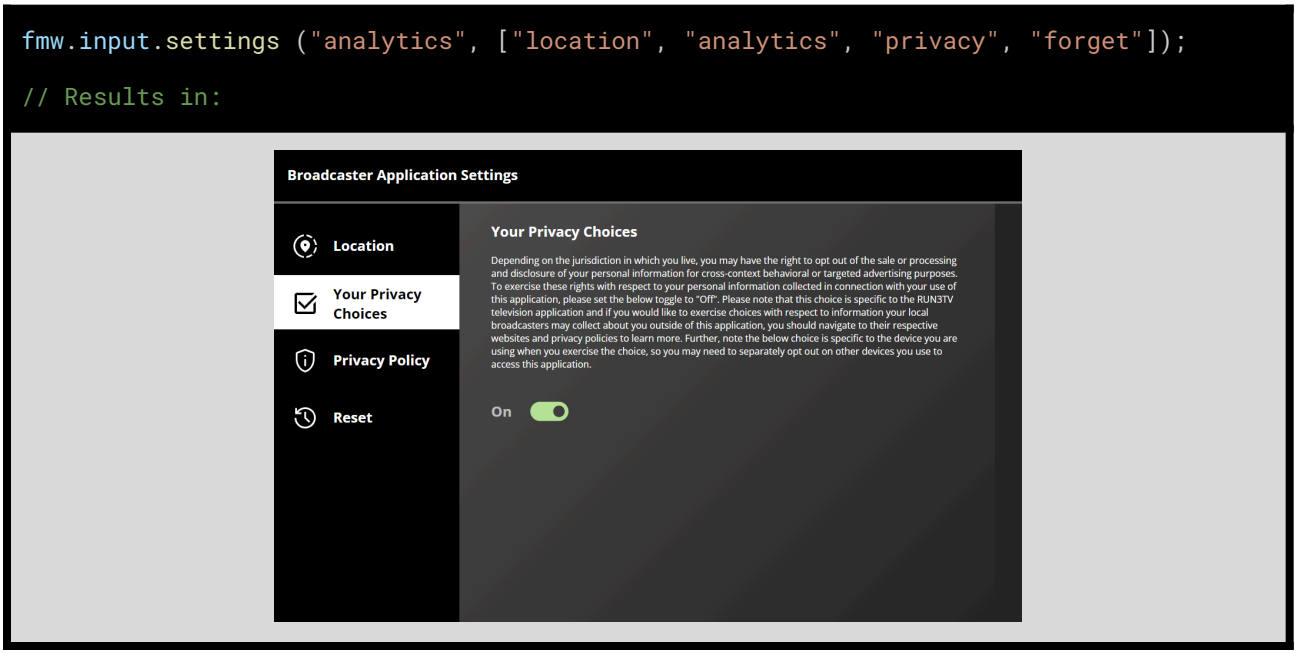
If `itemsToPresentArray` is an empty list, then all preferences menu items will be displayed in their default order.

Basic usage

The example presents the settings popup with four sections, and the “analytics” section highlighted.

```
fmw.input.settings ("analytics", ["location", "analytics", "privacy", "forget"]);

// Results in:
```



isLoaded()

The `isLoaded()` function of `fmw` returns whether the Framework is loaded or not.

Syntax

```
fmw.isLoaded();
```

Parameters

None

Return value

`Boolean`

If the Framework is loaded, true; otherwise false

Description

`fmw.isLoaded()` is a function of `fmw`.

This function returns the loaded state of the Framework, as summarized in the table below.

Return value	Description
`true`	The Framework is fully loaded. Framework functions can be called
`false`	The Framework is not fully loaded. Applications must not call Framework functions until `fmw.onload()` is fired

Basic usage

The example below prints the loaded state of the Framework to the console log.

```
console.log("Framework loaded state is: " + fmw.isLoaded());

// Framework loaded state is: true
```

onload()

The `onload()` event is fired when the Framework has completed loading.

Syntax

```
fmw.onload(callbackFn);
```

onfocus()

The `onfocus()` event is fired when focus within the Framework changes

Syntax

```
fmw.onfocus(callbackFn);
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following argument:

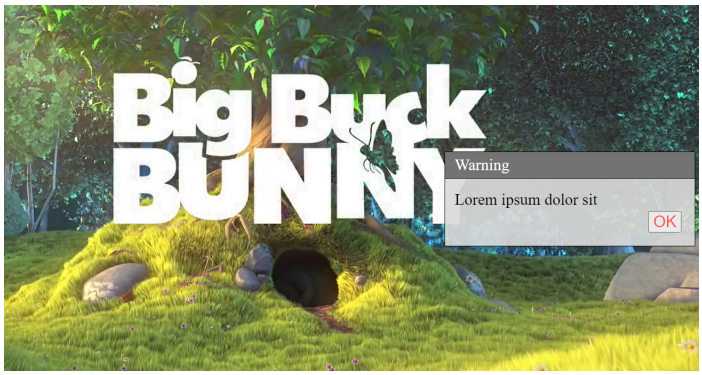
`newFocus <String>`

A string enum that reports the newly focused object:

newFocus value	Description
`console`	The Framework development console has focus
`app`	The application (iFrame) has focus
undefined	Another window has focus

notify.send()

The `notify.send()` method of `fmw` presents a developer-configurable popup message dialog box on screen. An example is shown below



The viewer can choose to close this message by pressing OK. Alternatively the message times out after a brief period of time.

Syntax

```
fmw.notify.send( {"priority": // 0|1|2,
                  "display":  // pop-up|console
                  "title":    // Pop-up title text
                  "text" :    // Pop-up message text
                  "footer" :  // Pop-up footer text
                 },
                  callbackFn()
                );
```

Parameters

`object`

A configuration object for the popup, containing the following parameters:

`priority <Number>`

A number defining the priority of the message. The effect of this field is dependent on the value of the display field, as summarized in the table below:

Priority value	display = pop-up	display = console
`0`	Messages with this value will be displayed over messages with priorities 1 and 2	Message is written to console error
1	Messages with this value will be displayed over messages with priority 2 but behind messages with priority 0	Message is written to console warn
2	Message with this value will be displayed behind messages with any other priority types	Message is written to console log

- `0` - High priority
- `1` - Medium priority
- `2` - Low priority

Any other value will be ignored and replaced with 1.

`display <String>`

The location to display the message. Permitted values are:

- `pop-up` (default)
- `console`

`title <String>`

The text to present as the title of the pop-up.

`text <String>`

The text to present within the pop-up.

`footer <String>`

The text to present at the bottom of the pop-up.

`callbackFn`

A function (with no parameters) that will be run when the popup dialog closes.

Description

`notify.send()` is a function of `fmw`.

This function displays a pop-up dialog box on screen.

If multiple notify.send() calls are made, the popups will be presented based on priority order, and are internally managed as a queue.

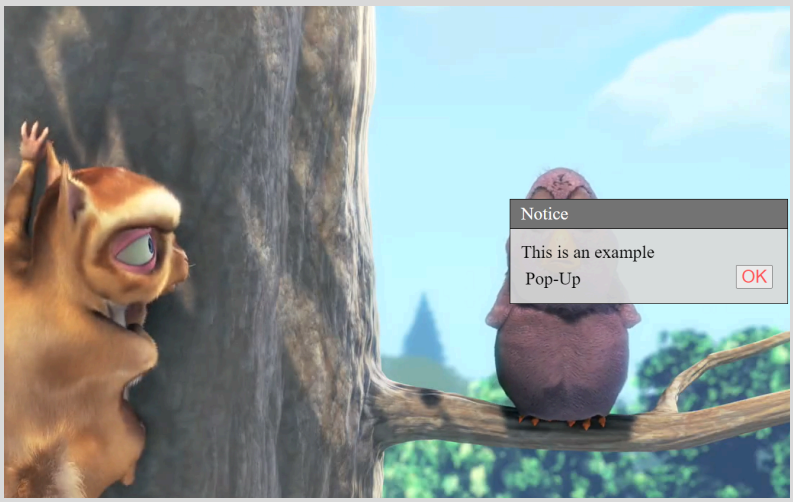
To adjust the design of the pop-up, modify the popUp object in **conf/fmw.json**.

Basic usage

This example presents a popup and presents text to the console log when the popup has closed.

```
fmw.notify.send(
    {"priority": 0,
      "display": "pop-up",
      "title": "Warning",
      "text" : "Lorem ipsum dolor sit",
      "footer" : ""},
      function(){
        console.log("The popup has closed.")
      }
    );

// Results in:
```



package()

The `package()` method of `fmw` returns details about the Framework.

Syntax

```
fmw.package(callbackFn(
                {
                  "name":    // Framework name
                  "version": // Framework version
                  "date":    // Framework release date (unix epoch)
                }
              )
            );
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

`Object`

A response object containing the following fields:

`name <String>`

The Framework release's name. At the time of writing, this will return "`a3fa-framework`"

`version <String>`

The version of the Framework, in the form:

`<major version>.<minor version>.<sub version>`

`date <Number>`

The release date of the Framework, in Unix epoch form.

Description

`fmw.package()` is a function of `fmw`.

This function returns details about the currently running Framework. Developers can use this information to verify version-specific functionality of the Framework.

Basic usage

This example prints the details of the Framework to the console log.

```
fmw.package(function(obj)
              {
              console.log("Framework Name: "  + obj.name +
              "\nFramework Version: "           + obj.version +
              "\nFramework Release Date: "      + obj.date);
              }
           )
// Results in:

// Framework Name: a3fa-framework
// Framework Version: 2.1.2
// Framework Release Date: 1698772141522
```

getTopLocation()

The `getTopLocation()` method of `fmw` gets the Top (parent) URL.

Syntax

```
fmw.getTopLocation(callbackFn(url));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

> **`url <String>`**
>
> The Top (parent) location URL

Description

`getTopLocation()` is a function of `fmw`.

This function returns the Top (parent) location url. Developers can use this information to determine the currently running environment.

Basic usage

This example prints the Top location to the console log.

```
fmw.getTopLocation(function(url)
                    {
                      console.log("Top is: " + url);
                    }
                  );

// Top is: http://localhost:5001/run3tv-common/?wsURL=ws%3A%2F%2Flocalhost%3A5001&rev=02212023
```

top()

The `top()` method of `fmw` sets the Top (parent) URL. This will cause the framework to be reloaded or replaced, depending upon the URL supplied.

Syntax

```
fmw.top(url);
```

Parameters

`url <String>`

The Top (parent) location url.

Description

`top()` is a function of `fmw`.

It sets the Top (parent) URL.

This function can only be used if the application has permission to change this value. This permission is granted by setting `allowTopNavigation : true` in the properties object of the application in **appsList.json** file (or equivalent file)

> ***Note*** Calling this function with the existing top url (found via getTopLocation()) will reload the Framework.
> Calling this function with a new url unload the current Framework.

Basic usage

This example sets the top location:

```
fmw.top("https://example.com/page.html");
```

8. Debug related

list()

The `list()` method of `fmw` returns a list of all methods and events available within the `fmw` object.

Syntax

```
fmw.list(callbackFn(arrayOfFeatures));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

> **`arrayOfFeatures <Array of Strings>`**
>
> An array of methods and events available within the `fmw` object.

Return value

None

Description

`list()` is a function of `fmw`.

This function returns a callback containing an array of all the methods and events available to use within the `fmw` object.

Basic usage

The example below prints the result of `list()` to the console log

```
fmw.list(console.log);

// (175) ['unload', 'uuidv4', 'dateRND', 'timeZone', 'mediaParse', … ]
```

log(), info(), warn(), error()

The `log()`, `info()`, `warn()` and `error()` methods of `fmw` write messages to the Framework log and the console log.

Syntax

```
fmw.log(...args);
fmw.info(...args);
fmw.warn(...args);
fmw.error(...args);
fmw.assert(assertion, ...args);
```

Parameters

…args ***<Any>***

Any number of arguments. Any non-string values will be coerced to strings. If multiple arguments are supplied they will be concatenated.

assertion <Boolean> (`fmw.assert()` only)

If false, the following `…args` will be written to the log, otherwise nothing will be written.

Return value

None

Description

`log()`, `info()`, `warn()`, `error()` and `assert()` are functions of `fmw`.

These functions write to the console log:

- `log()` writes informative logs (green in the Framework log)
- `info()` writes informative logs (blue in the Framework log)
- `warn()` writes warning logs (yellow in the Framework log)
- `error()` writes error logs (red in the Framework log)
- `assert()` writes error logs (red in the Framework log). If the `assertion` value is true, this will not be written to the console log.

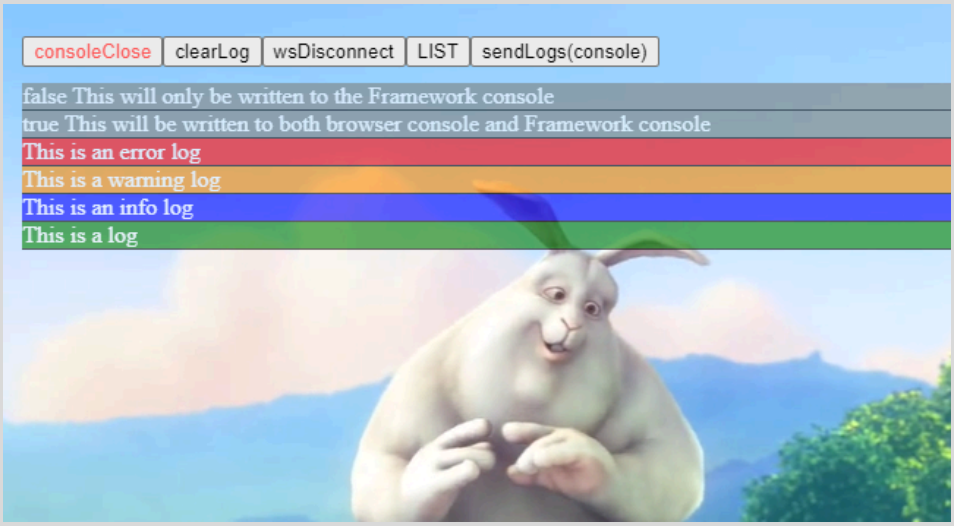
Basic usage

The example below write to the log at their various levels

```
fmw.log("This is a log");
fmw.info("This is an info log");
fmw.warn("This is a warning log");
fmw.error("This is an error log");
fmw.assert(1 == 1, "This will be written to both browser console and Framework console");
fmw.assert(1 != 1, "This will only be written to the Framework console");

// Browser console logs:

// This is a log
// This is an info log
// ⚠ This is a warning log
// ⊗ This is an error log
// ⊗ Assertion failed: This will only be written to the Framework console
```



navConsole()

The `navConsole()` method of `fmw` toggles the on-screen Framework console logs and buttons on and off.

Syntax

```
fmw.navConsole();
```

Parameters

None

Return value

None

Description

`navConsole()` is a function of `fmw`.

It toggles showing the on-screen debugging console logs and console buttons. This can be used to temporarily hide the console when navigating the application and to make it appear when actively debugging issues.

Basic usage

The example below can be run repeatedly to enable and disable the on-screen console logs and console buttons.

```
fmw.navConsole();
```

keyCodeSwitch()

The `keyCodeSwitch()` method of `fmw` toggles whether verbose keypresses are written to the on-screen Framework console logs.

Syntax

```
fmw.keyCodeSwitch();
```

Parameters

None

Return value

None

Description

`keyCodeSwitch()` is a function of `fmw`.

It toggles showing verbose keypress details on the on-screen debugging console log.

Basic usage

The example below can be run repeatedly to enable and disable keypress reporting to the on-screen console logs.

```
fmw.keyCodeSwitch();

// Enabled:
// {"keyCode":37,"key":"ArrowLeft","code":"ArrowLeft"}
// fmw key: 37

fmw.keyCodeSwitch();

// Disabled:
// fmw key: 37
```

clearLog()

The `clearLog()` method of `fmw` removes all on-screen console log entries from the screen.

Syntax

```
fmw.clearLog();
```

Parameters

None

Return value

None

Description

`clearLog()` is a function of `fmw`.

It removes any on-screen console log entries, but does not hide the console. This function is useful when debugging, to remove any previous and distracting log messages.

Basic usage

The example below clears the logs.

```
fmw.clearLog();
```

rpcLogs()

The `rpcLogs()` method of `fmw` enables and disables logging of ATSC 3.0 JSON-RPC calls to the Framework log.

Syntax

```
fmw.rpcLogs();
```

Parameters

None

Return value

`newRPCLoggingStatus <Boolean>`

If rpc logging has now been enabled, set to `true`, otherwise `false`.

Description

`rpcLogs()` is a function of `fmw`.

This function toggles the logging of ATSC 3.0 JSON-RPC calls to the Framework log.These calls can be numerous. This function enables the developer to hide and show them as required.

ATSC 3.0 JSON-RPC calls will always be written to the browser console log.

ATSC 3.0 JSON-RPC message logs begin with "`atscCmd ->`" for messages sent by the application or framework, and "`atscCmd <-`" for messages sent by the terminal.

An example pair of logs for the `org.atsc.query.rmpPlaybackState` method are shown below:

```
atscCmd -> {"jsonrpc":"2.0","method":"org.atsc.query.rmpPlaybackState","id":63}
atscCmd <- {"jsonrpc":"2.0","result":{"playbackState":0},"id":63}
```

Basic usage

The example below toggles the logging of ATSC 3.0 JSON-RPC calls.

```
fmw.rpcLogs();
```

dateRND()

The `dateRND()` method returns the dateRND parameter of the application's iFrame.

Syntax

```
fmw.dateRND(callbackFn(dateValue));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

> **`dateValue <Number>`**
>
> The unix epoch date value of the dateRND parameter of the application's iFrame.

Return value

None

Description

`dateRND()` is a function of `fmw`.

It returns the unix epoch date value of the dateRND parameter of the application's iFrame.

This dateRND parameter exists to ensure that the browser does not cache the application. It is included in the URL of the application's iFrame by the Framework, and is based on `Date.now()`.

This value may be of use to developers as it acts as a unique application instance identifier within the context of the browser.

This value should not be used as a globally unique identifier across the receiverbase, as any receiver that launches an application within the same second as any other will share the same `dateRND` value.

Basic usage

The example below gets the static date value.

```
fmw.dateRND(function(dateValue)
              {
                console.log("Date RND is : " + dateValue);
              }
           )

// Date RND is : 1700399534862
```

readyState()

The `readyState()` method returns the state of the WebSocket connection to the ATSC 3.0 JSON-RPC interface.

Syntax

```
fmw.readyState(callbackFn(state));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

`state <Number>`

The state of the ATSC 3.0 JSON-RPC interface. See https://developer.mozilla.org/en-US/docs/Web/API/WebSocket/readyState for permitted values

Return value

None

Description

`readyState()` is a function of `fmw`.

It returns the current state of the Framework's WebSocket connection to the ATSC 3.0 JSON-RPC WebSocket interface as detailed in [A/344:Various].

This function may be of use when debugging issues where the state of the WebSocket is in question.

Basic usage

The example below gets the state of the JSON-RPC interface before and after disconnecting and reconnecting to it.

```
fmw.readyState(function (status) {console.log("WebSocket status: " + status)});

// WebSocket status: 1


fmw.rpc.disConnect();
fmw.readyState(function (status) {console.log("WebSocket status: " + status)});

// WebSocket status: 3


fmw.rpc.connect();
fmw.readyState(function (status) {console.log("WebSocket status: " + status)});

// WebSocket status: 1
```

onopen()

The `onopen()` event is fired when the ATSC 3.0 JSON-RPC WebSocket interface is opened.

Syntax

```
fmw.onopen(callbackFn(readyState));
```

Please see `readyState()` for details of the various readyState values.

onclose()

The `onclose()` event is fired when the ATSC 3.0 JSON-RPC WebSocket interface is closed.

Syntax

```
fmw.onclose(callbackFn(readyState));
```

Please see `readyState()` for details of the various readyState values.

navClick(), navNumeric(), navNext(), navBack(), navUp(), navDown(), navEnter(), navExit(), navHome() and navQuit(), BAAppear()

The `nav*()` methods are used to simulate user navigation of the Framework.

Syntax

```
fmw.navClick(callbackfn(htmlElement));
fmw.navNumeric(digitString);
fmw.navNext();  // ArrowRight
fmw.navBack();  // ArrowLeft
fmw.navUp();    // ArrowUp
fmw.navDown();  // ArrowDown
fmw.navEnter(); // Select
fmw.navExit();  // BackUp
fmw.navHome();  // Home
fmw.navQuit();  // Exit
```

Parameters

.

`digitString <String>`

The digit to simulate. Permitted values are:

- `one`
- `two`
- `three`
- `four`
- `five`
- `six`
- `seven`
- `eight`
- `nine`
- `zero`

Return value

`htmlElement <DOM Object>`

The DOM object selected by the viewer. Only applicable to LG devices.

Description

The `nav*()` methods are functions of `fmw`.

They enable the developer to simulate user interaction with the Framework.

Basic usage

The example below simulates the “1” button on the remote control unit being pressed

```
fmw.navNumeric("one");
```

The example below registers for the onBAAppear() event and simulates the BAAppear button on the remote control unit being pressed:

```
fmw.onBAAppear(function(){ console.log("BAAppear pressed." );});
fmw.BAAppear(console.log);

//BAAppear pressed.
```

fmw.onNavClick(), fmw.onNavNumeric(), fmw.onNavNext(), fmw.onNavBack(), fmw.onNavUp(), fmw.onNavDown(), fmw.onNavEnter(), fmw.onNavExit(), fmw.onNavHome(), fmw.onNavQuit() and fmw.onBAAppear()

Keypresses on the remote control unit will cause the Framework events below to fire.

These events will also be fired when keypresses are simulated using the keypress simulation functions described earlier in this section.

Event	Cause of event	
	Keypress	Keypress simulation function
`fmw.onNavClick()`	MouseEvent	**`fmw.navClick()`**
`fmw.onNavNumeric()`	Numeric (1,2,3,4,5,6,7,8,9,0)	**`fmw.navNumeric()`**
`fmw.onNavNext()`	ArrowRight	**`fmw.navNext()`**
`fmw.onNavBack()`	ArrowLeft	**`fmw.navBack()`**
`fmw.onNavUp()`	ArrowUp	**`fmw.navUp()`**
`fmw.onNavDown()`	ArrowDown	**`fmw.navDown()`**
`fmw.onNavEnter()`	Select	**`fmw.navEnter()`**
`fmw.onNavExit()`	Back	**`fmw.navExit()`**
`fmw.onNavHome()`	Home	**`fmw.navHome()`**
`fmw.onNavQuit()`	Exit	**`fmw.navQuit()`**
`fmw.onBAAppear()`	BAAppear	**`fmw.BAAppear()`**

9. Device and viewer related

location.get()

The `location.get()` method of `fmw` returns the ZIP code of the device's location.

Syntax

```
fmw.location.get(callbackFn(location));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

> **`location <String>`**
>
> The location of the device.

Return value

None

Description

`location.get()` is a function of `fmw`.

It returns the stored location of the device. This location is either based on internet based geo-location service, or on the value set using location.set().

For production devices in the USA, the location value will be in the form of a five digit ZIP code or ZIP+4 code.

Geo-location services are not guaranteed to be 100% accurate.

This value may be updated by `fmw.location.set()`, in which case, the value may be either a five digit ZIP code or ZIP+4 code.

Note When running or debugging in the emulator outside of the USA, differently structured location values will be returned. For example, in the UK, the first half of the Postcode (the outward code) is provided.

Basic usage

The example below prints the location of the device to the browser's console log. The response matches that of a device in Noble, Indiana.

```
fmw.location.get(function(location)
                    {
                      console.log("Device location is: " + location)
                    }
                );

// Device location is: 47371
```

location.set()

The `location.set()` method of `fmw` sets the stored location of the device.

Syntax

```
fmw.location.set(zipCode, callbackFn);
```

Parameters

`zipCode <String>`

The ZIP code of the device. This may take the form of either a 5 digit ZIP code or a ZIP+4 code.

`callbackFn`

A function to execute once the location has been updated. The function is called with no arguments.

Return value

None

Description

`location.set()` is a function of `fmw`.

This function overwrites any existing location value with a new location. This is stored in the Framework's sessionStorage.

This function verifies that the value provided is a valid ZIP Code, using the following regex:

```
/^[0-9]{5}(?:-[0-9]{4})?$/
```

If the `zipCode` value provided does not match this regex, the location will not be updated.

Basic usage

The example below sets the location to a road in Noble, Indiana.

```
fmw.location.set("47371-1606", function()
                    {
                      console.log("Location successfully updated.")
                    }
                );

// Location successfully updated.
```

location.reset()

The `location.reset()` method of `fmw` resets the device's location to the default geo-location value.

Syntax

```
fmw.location.reset(callbackFn);
```

Parameters

`callbackFn`

A function to execute once complete. The function is called with no arguments.

Return value

None

Description

`location.reset()` is a function of `fmw`.

This function calls a geo-location service to determine the location of the device.

It may be useful to call this function in the following scenarios:

- When the location has previously been updated manually using `location.set()`, but an automatic detection is required.
- As a semi-regular check to determine if the device has moved address.

For devices in the USA, the geo-location service will likely reset the location to a 5-digit ZIP code.

> ***Note*** When running or debugging in the emulator outside of the USA, the location value will take a different form. For example, in the UK, the first half of the Postcode (the outward code) will be used.

Basic usage

The example below prints the result of `list()` to the console log

```
fmw.location.reset(function()
                    {
                      console.log("Location successfully updated.")
                    }
                );
```

location.hide()

The `location.hide()` method of `fmw` hides the location of the device

Syntax

```
fmw.location.hide(callbackFn);
```

Parameters

`callbackFn`

A function to execute once complete. The function is called with no arguments.

Return value

None

Description

`location.hide()` is a function of `fmw`.

This function hides the location by replacing it with the literal string “`hidden`”. As such it is functionally equivalent to calling `location.set(“hidden”, callbackFn)`.

This function might be used when a viewer requests that their location be made unavailable.

Basic usage

The example below hides the location.

```
fmw.location.hide(function()
                    {
                      console.log("Location successfully hidden.")
                    }
                );

// Location successfully hidden.
```

ccEnabled()

The `ccEnabled()` method of `fmw` returns the status of the viewer's preference for closed captions.

Syntax

```
fmw.ccEnabled(callbackFn(ccStatus));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

> **`ccStatus <Boolean>`**
>
> Set to `true` if closed captions are enabled by the viewer. Otherwise set to `false`.

Return value

None

Description

`ccEnabled()` is a function of `fmw`.

This function returns the viewer's preference to see closed captions.

This function might be used when deciding whether to enable subtitles on an AMP-presented video stream.

Basic usage

The example below prints to the console log the viewer's preference to see closed captions.

```
fmw.ccEnabled(function(ccStatus)
                {
                  console.log("Closed Captions state: " + ccStatus);
                }
              );

// Closed Captions state: true
```

onCaptionState()

The `onCaptionState()` event is fired whenever captioning is enabled or disabled by the viewer.

Syntax

```
fmw.onCaptionState(callbackFn(ccStatus));
```

Parameters

`ccStatus <Boolean>`

Set to `true` if closed captions are enabled by the viewer. Otherwise set to `false`.

Basic usage

The example below prints a message to the screen whenever captioning is enabled or disabled:

```
fmw.onCaptionState(
      function(ccStatus)
        {
          console.log("Captioning state is: " + ccStatus );
        }
);
```

getDevice()

The `getDevice()` method of `fmw` returns details about the device.

Syntax

```
fmw.getDevice(callbackFn(DeviceObject));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with a single object containing the following fields:

`DeviceObject`

A response object containing the following fields:

`deviceMake <String>`

The name of the manufacturer of the device.

If an error has occurred, “unknown” will be returned.

`deviceModel <String>`

The name of the model. The structure of this will be manufacturer-specific.

If an error occurs, this value will not be present.

`deviceInput: <Object{“<KeyName>” = data}>`

The list of keys available for use by the Framework, along with their keycode.

If an error occurs, this value will not be present.

Most entries in this object take the form:

```
“<keyName>” : keyCode
```

The BAAppear key includes additional information, of the form:

```
BAAppear :
{
  “label” : “<label text>”,
  “keycode” :  <keyCode>,
  “img” : <base 64 encoded image of the BA Appear button>
}
```

Return value

None

Description

`getDevice()` is a function of `fmw`.

This function provides details about the device, by querying `org.atsc.query.deviceInfo`.

This data is requested and cached when the Framework is first launched. When `fmw.getDevice()` is called, the cached data is returned.

The data provided by this function may be useful when determining how to best offer key-driven navigation functionality, as not all device's remote control units contain the same set of keys.

Additionally, the data provided by this function may be useful should manufacturer- or device-specific workarounds are required.

Note The `deviceInput` object can be requested separately by calling deviceInput().

Basic usage

The example below prints the response object of `getDevice()` to the console log.

The example output is based on that of the A3FA emulator. The "img" value has been truncated for brevity.

```
fmw.getDevice(function(deviceInfo)
                {
                  console.log("Manufacturer: " + deviceInfo.deviceMake +
                              "\nModel: " + deviceInfo.deviceModel +
                              "\nInput: " + JSON.stringify(deviceInfo, null, "  "));
                }
              );

// Manufacturer: Yotta
// Model: atsc3-rx-emulator-2023
// Input: {
//   "ArrowUp": 38,
//   "ArrowDown": 40,
//   "ArrowRight": 39,
//   "ArrowLeft": 37,
//   "Select": 13,
//   "Back": 8,
//   "Home": 36,
//   "Reload": 19,
//   "Console": 48,
//   "BAAppear": {
//     "label": "OK",
//     "keycode": 13,
//     "img": "..."
//   }
// }
```

deviceInput()

The `deviceInput()` method of `fmw` returns details about the keys available for use by the Framework.

Syntax

```
fmw.deviceInput(callbackFn(deviceInputObject));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with a single object containing the following fields:

`deviceInputObject`

The list of keys available for use by the Framework, along with their keycode. Most entries in this object take the form:

```
"<keyName>" : keyCode
```

The BAAppear key includes additional information, of the form:

```
BAAppear :
{
  "label" : "<label text>",
  "keycode" :  <keyCode>,
  "img" : <base 64 encoded image of the BA Appear button>
}
```

Return value

None

Description

`deviceInput()` is a function of `fmw`.

This function provides details about the keys associated with the device.

Basic usage

The example below prints the response object of `deviceInput()` to the console log. The “img” value has been truncated for brevity.

```
fmw.deviceInput(function(deviceInput)
                {
                  console.log("Input: " + JSON.stringify(deviceInput, null, "   "));
                }
              );

// Input: {
//   "ArrowUp": 38,
//   "ArrowDown": 40,
//   "ArrowRight": 39,
//   "ArrowLeft": 37,
//   "Select": 13,
//   "Back": 8,
//   "Home": 36,
//   "Reload": 19,
//   "Console": 48,
//   "BAAppear": {
//     "label": "OK",
//     "keycode": 13,
//     "img": "..."
//   }
```

languages()

The `languages()` method of `fmw` returns details about the viewer's preferred language settings.

Syntax

```
fmw.languages(callbackFn(languagesObject));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

`languagesObject`

A response object containing the following fields:

`preferredAudioLang : <String>`

The preferred audio language as set by the viewer.

`preferredCaptionSubtitleLang <String>`

The preferred closed captioning / subtitle language as set by the viewer.

`preferredUiLang <String>`

The preferred user interface language as either set by the viewer or by language.set().

Return value

None

Description

`languages()` is a function of `fmw`.

This function returns a callback containing details of the preferred language(s), as set by the viewer. All language identifiers are provided as two-character ISO 639-1 language codes.

Basic usage

The example below prints the result of `languages()` to the console log

```
fmw.languages(function(langDetails)
    {
      console.log("Preferred AUDIO language: "
                                + langDetails.preferredAudioLang +

                "\nPreferred CAPTIONS language: "
                                + langDetails.preferredCaptionSubtitleLang +

                "\nPreferred UI language: "
                                + langDetails.preferredUiLang);
    }
);

// Preferred AUDIO language: en
// Preferred CAPTIONS language: es
// Preferred UI language: en
```

onUiLang()

The `onUiLang()` event is fired whenever the preferred languages selection is changed by the viewer.

Parameters

`callbackFn`

A function that will be executed when the event is fired. The function is called with the same arguments as the callback function of fmw.languages().

Basic usage

The example below sets up `onUiLang()` to print a summary of the preferred languages to the console log whenever the language is changed. When `fmw.language.set()` is called, the preferred languages change and the summary is printed to the console log.

```
fmw.onUiLang(function(){
    fmw.languages(function(lang){
      console.log("Preferred languages have changed to:\n"
                  + JSON.stringify(lang, null, " "))
    } );
  }
);

fmw.language.set("es", null);

// Preferred languages have changed to:
// {
//   "preferredAudioLang": "en",
//   "preferredUiLang": "en",
//   "preferredCaptionSubtitleLang": "en"
//  }
```

language.set()

The `language.set()` function of `fmw` sets the UI language of the Framework and associated applications.

Syntax

```
fmw.language.set(uiLanguageCode,
                 callbackFn(langCode)
                );
```

Parameters

`uiLanguageCode <String>`

The new ISO 639-1 two-letter language code for the UI.

`callbackFn`

A function that will be executed once the UI language has been set. The function is called with the following arguments:

> **`langCode <String>`**
>
> The preferred audio language (as set by `uiLanguageCode`).

Return value

None

Description

`language.set()` is a function of `fmw`. It sets the UI language for the Framework and associated applications. This will be stored in the Framework's session storage.

This function does not modify the viewer's preferred UI language setting in the terminal.

When called, this function will fire an fmw.onUiLang() event.

Basic usage

The example below sets the preferred UI language to English ("en").

```
fmw.language.set("en", function(langCode)
                 {
                   console.log("UI language updated to: " +langCode);

                 }
);

// UI language updated to: en
```

language.get()

The `language.get()` function of `fmw` gets the UI language of the Framework and associated applications.

Syntax

```
fmw.language.get(callbackFn(langCode));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

> **`langCode <String>`**
>
> The preferred audio language

Return value

None

Description

`language.get()` is a function of `fmw`. It gets the preferred UI language for the Framework and associated applications.

The returned language identifier is provided as a two-character ISO 639-1 language code.

Basic usage

The example below gets the preferred language of the UI.

```
fmw.language.get(function(langCode)
                {
                  console.log("UI language is: " +langCode);

                }
);

// UI language is: en
```

timeZone()

The `timeZone()` function of `fmw` gets the time zone of the receiver, as determined by the receiver's browser.

Syntax

```
fmw.timeZone(callbackFn(timeZoneObject));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with a single object containing the following arguments:

> **`timeZoneObject`**
>
> A response object containing the following fields:
>
> **`code : <timezone abbreviation>`**
>
> The device's current time zone code.

Return value

None

Description

`timeZone()` is a function of `fmw`. It gets the device's current time zone.

The returned time zone abbreviation code is based on the receiver's time information (as provided by the broadcast ATSC 3.0 signal) and mapped using the **public/run3tv-common/conf/dtz.json** file.

Basic usage

The example below gets the device's current time zone.

```
fmw.timeZone(function(timeZone)
                {
                  console.log("Current time zone is: " +timeZone.code);
                }
            );

// Current time zone is: EST
```

10. Internal storage and Inter-app communication related

The Framework has a number of functions to store data persistently and semi-persistently in various areas of the browser, as summarized in the table below:

Storage area	Framework Functions	Description
LocalStorage	`fmw.setItem()`	Stores data to `Window.localStorage`
	`fmw.getItem()`	Gets data from `Window.localStorage`
	`fmw.removeItem()`	Removes data from `Window.localStorage`
SessionStorage	`fmw.set()`	Stores data to `Window.sessionStorage`
	`fmw.get()`	Gets data from `Window.sessionStorage`
	`fmw.del()`	Removes data from `Window.sessionStorage`
Framework storage *(For inter-app communications)*	`fmw.setData()`	Stores data to the Framework storage area
	`fmw.getData()`	Gets data from the Framework storage area
	`fmw.removeData()`	Removes data from the Framework storage area
All storage areas	`fmw.clear()`	Clears all persistent data and reloads the framework location

Framework storage is used when sharing data between applications. This data is written within the JS DOM of the Framework, so will be lost when the Framework terminates.

In order for applications to successfully communicate via Framework storage, the application attempting to read data must both:

- Know the `appName` of the application that wrote the data (as set in **appsList.json**)
- Have the `allowFrameworkStorage` flag set to true in the application's `properties` object of **appsList.json** file.

LocalStorage: setItem(), getItem() and removeItem()

Data can be persistently stored, retrieved and removed using `setItem()`, `getItem()` and `removeItem()` respectively. These functions make use of `Window.localStorage` storage.

Syntax

```
fmw.setItem(key,
            value,
            callbackFn(value) // optional
            );

fmw.getItem(key,
            callbackFn(value)
           );

fmw.removeItem(key);
```

Parameters

`key <String>`

The key to store or retrieve the data.

`value <String>`

The data associated with the `key`.

`callbackFn`

A function to execute containing the response. The function is called with a single object containing the following arguments:

> **`String`**
>
> The data stored with the `key` provided with the function call.
>
> If the `key` has no data associated with it, `null` is returned.

Description

`setItem()`, `getItem()` and `removeItem()` are functions of `fmw`.

They store, retrieve and remove data respectively from `Window.localStorage`.

Attempting to retrieve a key that does not exist will return `null`.

Attempting to remove a key that does not exist will cause no effect.

To use these functions, the application's `allowLocalStorage` value must be set to `true` in the **appsList.json** file (or equivalent).

Basic usage

The example below describes setting, getting, updating and removing data to `localStorage`.

```
// Store data to the "trialistFeaturesEnabled" key
fmw.setItem("trialistFeaturesEnabled","yes", console.log);

// yes

// Read the data back
fmw.getItem("trialistFeaturesEnabled", console.log);

// yes

// Update the data
fmw.setItem("trialistFeaturesEnabled","no", console.log);

// no

// Read the (updated) data back again
fmw.getItem("trialistFeaturesEnabled", console.log);

// no

// Remove the item
fmw.removeItem("trialistFeaturesEnabled")

// Attempt to read the item. It no longer exists.
fmw.getItem("trialistFeaturesEnabled", console.log);

// null
```

SessionStorage: set(), get() and del()

Data can be persistently stored, retrieved and removed using `set()`, `get()` and `remove()` respectively. These functions make use of `Window.SessionStorage` storage.

Syntax

```
fmw.setData(key,
            value,
            callbackFn(value) // optional
            );

fmw.getData(key,
            appName,
            callbackFn(value)
           );

fmw.removeData(key);
```

Parameters

`key <String>`

The key to store or retrieve the data.

`value <String>`

The data associated with the `key`.

`callbackFn`

A function to execute containing the response. The function is called with no arguments.

Description

`set()`, `get()` and `remove()` are functions of `fmw`.

They store, retrieve and remove data respectively from `Window.sessionStorage`.

Attempting to retrieve a key that does not exist will return `null`.

Attempting to remove a key that does not exist will cause no effect.

Basic usage

The example below describes setting, getting, updating and removing data to `localStorage`.

```
// Store data to the "A_B_Testing_Group" key
fmw.set("A_B_Testing_Group","A", console.log);

// null

// Read the data back
fmw.get("A_B_Testing_Group", console.log);

// null

// Update the data
fmw.set("A_B_Testing_Group","B", console.log);

// null

// Read the (updated) data back again
fmw.get("A_B_Testing_Group", console.log);

// null

// Remove the item
fmw.del("A_B_Testing_Group");

// Attempt to read the item. It no longer exists.
fmw.get("A_B_Testing_Group", console.log);

// null
```

Framework Storage: setData(), getData() removeData()

Data can be shared between Framework applications using `setData()`, `getData()` and `removeData()`.

Syntax

```
fmw.setData(key,
            value,
            callbackFn(value) // optional
            );

fmw.getData(key,
            appName,
            callbackFn(value)
            );

fmw.removeData(key);
```

Parameters

`key <String>`

The key to store or retrieve the data.

`value <String>`

The data associated with the `key`.

`appName <String>`

The name of the application (as set by `appName` in the **appsList.json** file) that wrote the data originally.

`callbackFn`

A function to execute containing the response. The function is called with a single object containing the following arguments:

> **`String`**
>
> The data stored with the `key` provided with the function call.
>
> If the `key` has no data associated with it, `null` is returned.

Description

`setData()`, `getData()` and `removeData()` are functions of `fmw`.

They store, retrieve and remove data respectively from the Framework's storage.

These functions enable inter-app communication. One application can write data to the Framework's storage and another can then read that data, so long as the reading application knows the `appName` of the application that wrote the data (as set in the **appsList.json** file).

Attempting to retrieve a key that does not exist will return `null`.

Attempting to remove a key that does not exist will cause no effect.

To make use of these functions, the `allowFrameworkStorage` field must be set to `true` in the applications `properties` object within **appsList.json**. If this field is not set, attempting to use these functions will fail silently.

Basic usage

The example below describes setting, getting, updating and removing data to Framework storage.

```
//
// --- Application "STATION-3" is running
//

// Store data to the "DeepLink" key
fmw.setData("DeepLink","weather", console.log);

// weather

//
//  --- New application loads
//


// Read the data back
fmw.getData("DeepLink", "STATION-3", console.log);

// weather
```

clear()

The `clear()` function of `fmw` clears all storage types and reloads the Framework.

Syntax

```
fmw.clear();
```

Parameters

None

Return value

None

Description

`clear()` is a function of `fmw`.

It first clears `Window.localStorage`, `Window.sessionStorage` and Framework storage, then it reloads the Framework.

Basic usage

The example below clears all storage and reloads the Framework.

```
fmw.clear();
```

11. Application launch and selection related

The functions in this section relate to application selection, launching and termination.

load()

The `load()` function of `fmw` launches an application into the Framework.

Syntax

```
fmw.load(srcURL,
         config, // Optional
         event,  // Optional
         force   // Optional
         );
```

Parameters

`srcURL <String>`

The URL of the application to launch.

`config <Optional Object>`

An optional configuration object. For more details see [R3TV-IG-0204].

`event <Optional String>`

An optional event name. This name is the equivalent of a name supplied in the **appSchedule.json** file. For more details, see [R3TV-IG-0204].

`force <Optional Boolean>`

If set to `true` will load the application even if the `event` value is the same as the current event name.

If not present, set to `false`.

Return value

None

Description

`load()` is a function of `fmw`. It loads the application at the URL defined in `srcURL`.

Basic usage

The example below shows the structure required to launch an application.

```
fmw.load( "http://example.com/pac-man.html",
          {"ga-id": "G-8JS5WSJ4JH"},
          "LocalNews",
          false
        );
```

loadApp()

The `loadApp()` function of `fmw` launches an application into the Framework.

Syntax

```
fmw.loadApp(appName,
            config, // Optional
            event,  // Optional
            force   // Optional
           );
```

Parameters

`appName <String>`

The name of the application to launch, as defined by the `appName` values in **appsList.json**. For more details see [R3TV-IG-0204].

`config <Optional Object>` *`Optional`*

An application configuration object that appends/replaces the original configuration of the previous application. For more details see [R3TV-IG-0204].

`event <Optional String>` *`Optional`*

An optional event name. This name is the equivalent of a name supplied in the **appSchedule.json** file. For more details, see [R3TV-IG-0204].

`force <Optional Boolean>` *`Optional`*

If set to `true` will load the application even if the `event` value is the same as the current event name.

If not present, set to `false`.

Return value

None

Description

`loadApp()` is a function of `fmw`. It loads the application with the same name as defined in **appsList.json**.

Basic usage

The example below launches the pac-man game.

```
fmw.loadApp( "pac-man",
          {"ga-id": "G-8JS5WSJ4JH"},
          null,
          "LocalNews"
        );
```

launchApp()

The `launchApp()` function of `fmw` launches an ATSC 3.0-native interactive application.

Syntax

```
fmw.launchApp(appId,
              parameters,
              callbackFn(error, result)  // Optional
             );
```

Parameters

`appId <String>`

The name of the application to launch, as defined by an `appID` value in the broadcast HELD (HTML Entry pages Location Description). For more details see [A/331:2023-10].

`parameters <String>`

An optional configuration string. For more details see [R3TV-IG-0204].

`callbackFn <Optional>`

A function to execute containing the response. The function is called with a single object containing the following fields:

> **`error <Object>`**
>
> An ATSC 3.0 JSON-RPC error object, as described in § 5.1 of [A/344:Various].
>
> **`result <Object>`**
>
> An ATSC 3.0 JSON-RPC result object, as described in § 9.7.6 of [A/344:Various].
>
> If the function is successful, an empty object will be returned here.

Return value

None

Description

`launchApp()` is a function of `fmw`. It attempts to launch an ATSC 3.0-native interactive application by calling the `org.atsc.launchApp` ATSC 3.0 JSON-RPC message. For more details see § 5.1 of [A/344:Various].

At the time of writing, the emulator does not support this function.

Launching an ATSC 3.0-native interactive application will terminate the Framework.

Basic usage

The example launches an application named example.com/news/1. This must be present in the broadcast HELD.

```
fmw.launchApp( "example.com/news/1",
          "",
          function(err, result)
          {
            console.log ("Error Object: " + JSON.stringify(err, null, "   "));
            console.log ("Result Object: "+ JSON.stringify(result, null, "   "));
           }
        );
```

appBack()

The `appBack()` function of `fmw` terminates the currently running Framework application and loads the previously running Framework application.

Syntax

```
fmw.appBack(configuration, // Optional
            eventName      // Optional
            );
```

Parameters

`configuration <Object>` *Optional*

An application configuration object that appends/replaces the original configuration of the previous application. For more details see [R3TV-IG-0204]

If not present, the original configuration of the previous application will be used.

`eventName <String>` *Optional*

An event name (equivalent to event names in the **appSchedule.json** file.

If not present, the original eventName of the previous application will be used (if present).

Return value

None

Description

`appBack()` is a function of `fmw`. It terminates the currently running application and loads the previously running application.

If no previously running application exists, it will perform no action.

Basic usage

Use the example provided in loadApp() to launch two new applications, then use the example below to close the most recent application and revert back to the previous application using its original configuration and event name.

```
fmw.appBack();
```

appPrev()

The `appPrev()` function of `fmw` returns the `appName` of the application that was previously running.

Syntax

```
fmw.appPrev(callbackFn(previousAppName));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

> **`previousAppName <String>`**
>
> The name of the previously running application on the service.

Return value

None

Description

`appPrev()` is a function of `fmw`. It returns the appName of the application that was previously running.

If no previously running application exists, the return value will be *Undefined*.

Basic usage

Use the example provided in loadApp() to launch two new applications, then use the example below to get the name of the previous application.

```
fmw.appPrev(function(previousNameApp)
        {
          console.log("Previous app is: " + previousNameApp);
        }
      );

// Previous app is: STATION-3
```

appEvent()

The `appEvent()` function of `fmw` returns the name of the currently active application event, as defined by the **appSchedule.json** file. For more details see [R3TV-IG-0204].

Syntax

```
fmw.appEvent(callbackFn(currentAppEventName));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following argument:

> **`currentAppEventName <String>`**
>
> The name of the currently running application event, as defined by the currently active `appSchedules/schedule` element of **appSchedule.json**.

Return value

None

Description

`appEvent()` is a function of `fmw`. It returns the name of the currently active application event.

If no event is active the return value will be *Undefined*.

Basic usage

The example below prints the current application event to the console log.

```
fmw.appEvent(function(currentAppEventName)
        {
          console.log("Current app event is: " + currentAppEventName);
        }
      );

// Previous app event is: event-1
```

12. Navigation related

acceptedKeys()

The `acceptedKeys()` function of `fmw` returns an array of all keys that the Framework is registered to accept.

Syntax

```
fmw.acceptedKeys(callbackFn(keyList));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following arguments:

> **`keyList <Array of Strings>`**
>
> An array where each element is the name of a registered key type.

Return value

None

Description

`acceptedKeys()` is a function of `fmw`. It returns a list of all keys that are currently registered to the Framework.

If no keys are registered, the return value will be *undefined*.

Basic usage

The example below prints the list of currently registered keys to screen.

```
fmw.acceptedKeys(function(keyList)
        {
          console.log("Currently registered keys are:\n    " + keyList);
        }
      );

// Currently registered keys are:
//   ArrowRight,ArrowLeft,Select,ArrowUp,ArrowDown
```

requestKeys()

The `requestKeys()` function of `fmw` adds key types to the list of keys registered by the Framework.

Syntax

```
fmw.requestKeys(keycodeArray,
                callbackFn()
               );
```

Parameters

`keycodeArray <Array of Strings>`

An array with at least one element. Each element must contain a name of a new key to register.

`callbackFn`

A function to execute once complete. The function is called with no arguments.

Return value

None

Description

`requestKeys()` is a function of `fmw`. It registers access to additional keys using the `org.atsc.request.keys` ATSC 3.0 JSON-RPC method.

The `acceptedKeys()` method should be used to determine which keys are available for use.

If this function is used to attempt to register already registered keys, that key will remain registered.

⚠ To ensure that applications do not inhibit normal use of the receiver, developers must not request keys other than `BAAppear` (and optionally `Back`) until the viewer has pressed the BAAppear key and entered the application.

When the viewer exits the application but stays within the Framework, all keys other than `BAAppear` and `Back` (to hide the trigger) must be deregistered.

For more details, please refer to [R3TV-IG-201].

If the Framework application is replaced with another Framework application (for example due to a schedule change or through the use of `load()` or `loadApp()`), access to any requested keys will be removed from the original application. In other words, the application does not need to relinquish keys if it detects that it has been made inactive.

If that application is later restored (for example, through the use of `appPrev()`), the Framework will restore access to the keys registered. In other words, the application does not need re-request keys if it detects that it has been restored.

Basic usage

The example below registers the ArrowRight and ArrowLeft keys.

```
fmw.requestKeys(["ArrowRight", "ArrowLeft"], function()
        {
          console.log("Keys registered.");
        }
      );

// Keys registered.
```

relinquishKeys()

The `relinquishKeys()` function of `fmw` deregisters keys that were previously registered by the Framework.

Syntax

```
fmw.relinquishKeys(keysToRelinquish,
                   callbackFn()
                  );
```

Parameters

`keysToRelinquish <Array of Strings>`

An array. Each element must contain the name of a key to deregister.

`callbackFn`

A function to execute once complete. The function is called with no arguments.

Return value

None

Description

`relinquishKeys()` is a function of `fmw`. It deregisters keys, using the `org.atsc.relinquish.keys` ATSC 3.0 JSON-RPC method.

If this function is used to attempt to register already registered keys, that key will remain registered.

Basic usage

The example below relinquishes the ArrowLeft key.

```
fmw.relinquishKeys(["ArrowLeft"], function()
        {
          console.log("Key relinquished.");
        }
      );

// Keys relinquished.
```

13. ESG related

programs()

The `programs()` method of `fmw` returns the list of broadcast events for the current service from the ESG (Electronic Service Guide).

Syntax

```
fmw.programs(callbackFn(programData));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with a single object containing the following fields:

`programData <Object>`

`service : <Global Service ID>`

The service ID of the current service.

`id : <String>`

The ESG's `ServiceReference` `idRef` attribute for the service.

`list : <Object of 0 or more ContentReference : EventObject>`

A list containing event details for all known events, keyed off of each event's ContentReference ID, as defined by the ESG's `ContentReference` `idRef` attribute.

The `EventObject` contains the following fields:

`startTime : <Number>`

The start time of the event, supplied as a Unix epoch value.

`endTime : <Number>`

The end time of the event, supplied as a Unix epoch value.

`duration : <Number>`

The length of the event in whole seconds.

`name : <String>`

The name of the event.

`description : <String>`

The description of the event.

Return value

None

Description

`programs()` is a function of `fmw`.

This function provides the list of events associated with the current service, as discovered using the ATSC 3 JSON-RPC method `org.atsc.query.serviceGuideUrl`.

Events in the returned list are sorted by `contentReference` IDs and may include events that have occurred in the past.

The schedule of events may be received via the ATSC 3.0 broadcast. If this is the case, the receiver downloads this data slowly over time to build up a schedule.
It may be appropriate to repeatedly poll the `programs()` method to ensure the most recent and complete set of data is returned.

Developers should ensure that their implementations are hardened to cope with gaps in the schedule, overlapping events, large amounts of data and other similar data states that may occur.

Basic usage

The example below prints the response object of `programs()` to the console log. The list of `EventObjects` has been truncated for brevity.

```
fmw.programs(function(programData)
      {
        console.log("Program Data:\n" + JSON.stringify(programData, null, "  "));
      }
    );


// // Program Data:
// {
//   "service": "tag:run3tv.org,2023:globalServiceID/2",
//   "id": "bcast://a3fa.com/Service-1",
//   "list": {
//     "bcast://a3fa.com/Content-1-100001": {
//       "startTime": 1700499909,
//       "endTime": 1700507109,
//       "duration": 7200,
//       "name": "Local News",
//       "description": "News, Sport and Weather for the local area"
//     },

//   ...

//     "bcast://a3fa.com/Content-1-100003": {
//       "startTime": 1700514309,
//       "endTime": 1700517909,
//       "duration": 3600,
//       "name": "World News",
//       "description": "Update on news around the world."
//     }
//   }
// }
```

serviceGuide()

The `serviceGuide()` method of `fmw` returns either the Now or Next event for the current service from the ESG (Electronic Service Guide).

Syntax

```
fmw.serviceGuide(nowOrNext, callbackFn(eventData));
```

Parameters

`nowOrNext <String>`

An enum. Permitted values are:

- `now` - Requests the current event on the current service
- `next` - Requests the following event on the current service

`callbackFn`

A function to execute containing the response. The function is called with a single object containing the following fields:

`eventData <Object>`

`service : <Global Service ID>`
The service ID of the current service.

`id : <String>`
The ESG `ServiceReference` `idRef` attribute.

`startTime : <Number>`
The start time of the event, supplied as a Unix epoch value.

`endTime : <Number>`
The end time of the event, supplied as a Unix epoch value.

`duration : <Number>`
The length of the event in whole seconds.

`idRef : <String>`
The ESG's `ContentReference` `idRef` attribute for the event.

`name : <String>`
The name of the event.

`duration : <Number>`
The length of the event in whole seconds.

Return value

None

Description

`serviceGuide()` is a function of `fmw`.

This function provides either the current (“now”) event or following (“next”) event on the current service, as defined by the ATSC 3.0 Electronic Service Guide (ESG).

Basic usage

The example below prints the current (“now”) event for the current service to the console log.

```
fmw.serviceGuide("now", function(eventData)
      {
        console.log("The current event is:\n" + JSON.stringify(eventData, null, "   "));
      }
    );

// The current event is:
// {
//   "service": "tag:run3tv.org,2023:globalServiceID/1",
//   "id": "bcast://a3fa.com/Service-1",
//   "startTime": 1700642846,
//   "endTime": 1700650046,
//   "duration": 7200,
//   "idRef": "bcast://a3fa.com/Content-1-100001",
//   "name": "Program title",
//   "description": "Current Program Description"
// }
```

14. Broadcast/IP service related

rmpAudioTracks()

The `rmpAudioTracks()` method of `fmw` returns the number of audio tracks per language available on the current ATSC 3.0 service.

Syntax

```
fmw.rmpAudioTracks(callbackFn(tracks));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with a single `tracks` object containing the following fields:

`<LanguageCode> : <Number>`

The number of tracks available for each audio language.

Return value

None

Description

`rmpAudioTracks()` is a function of `fmw`.

It returns a list of audio tracks currently available to be selected on the current ATSC 3.0 service.

Note This function is robust to the differing methods of track detection in ATSC 3.0.
The track list will be generated using the ATSC 3.0 `org.atsc.query.signaling` message if it is available on the receiver.

If it is not available, the function will manually parse the MPD to extract the audio track information.

Configuring the emulator

> When using this method with the emulator, the emulator must be configured with the appropriate `org.atsc.query.signaling` data in the `emulator/atscCmd-*.mc.json` file.

The `emulator/atscCmd-*.mc.json` file is used by the emulator to configure the channel line-up.

The key `org.atsc.query.signaling` key including an “MPD” field must be added to any service where an application will call `rmpAudioTracks()` or `audioTracks()`.

An example fragment of the file (describing a single service) is shown below.

emulator/atscCmd-*.mc.json

```
[
  {
    "com.run3tv.query.emulator": {
      "a3fa-appContextId": "tv:a3fa-apps.yottamedialabs.com",
      "a3fa-rmp": "http://example.com/file.mpd",
      "a3fa-held": "http://localhost:5001/run3tv-common/"
    },
    "org.atsc.query.service": {
      "service": "tag:run3tv.org,2023:globalServiceID/3"
    },
    "org.atsc.query.signaling": {
      "LLS": {
          "1": {"name": "1", "group": "1", "version": "1", "table": "<SLT bsid=\"9999\">...</SLT>"}
      },
      "RD": {
          "MPD": {"name": "MPD", "version": "1", "table": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><MPD>...</MPD>"}
      }
    }
  }
]
```

Attempting to run these methods without correctly configuring the emulator will result in a callback function response of *undefined* and the following error message sent to the logs:

```
Code: 1000-1004-1010 An event occurred during operation a3fa.fmwfunc.audioTracks
adaptationSet.length: 0
```

Basic usage

The example returns a summary of the audio tracks currently available in the RMP.

```
fmw.rmpAudioTracks("https://example.com/video.mpd" function(trackList)
      {
        console.log("The track list is:\n" + JSON.stringify(trackList, null, "  "));
      }
    );

// --------------------

// The track list is:
// {
//   "eng": 5
// }
```

audioTracks()

The `audioTracks()` method of `fmw` returns the list of audio tracks available in an MPD.

Syntax

```
fmw.rmpAudioTracks(mpdURL,
                   callbackFn(tracks)
                  );
```

Parameters

`mpdURL`

The url of an MPD

`callbackFn`

A function to execute containing the response. The function is called with a single `tracks` object containing the following fields:

> **`<LanguageCode> : <Number>`**
>
> The number of tracks available for each audio language.

Return value

None

Description

`audioTracks()` is a function of `fmw`.

This function returns a list of audio tracks currently available to be selected on the current ATSC 3.0 service

Basic usage

The example returns details about the audio tracks currently available in the RMP.

```
fmw.audioTracks("https://example.com/video.mpd" function(trackList)
      {
        console.log("The track list is:\n" + JSON.stringify(trackList, null, "  "));
      }
    );

// --------------------

// The track list is:
// {
//   "eng": 5
// }
```

trackSelect()

The `trackSelect()` method of `fmw` changes the audio track of the current service.

Syntax

```
fmw.trackSelect(trackSelectionId,
                callbackFn()
               );
```

Parameters

`trackSelectionId <Number>`

The number of the track to present, as defined by the ATSC 3.0 JSON-RPC method `org.atsc.track.selection`. For more information, see [A/344:Various].

`callbackFn`

A function to execute containing the response. The function is called with a single object. If successful, the result object of `org.atsc.track.selection` will be returned, otherwise the error object of this JSON-RPC method will be returned

> **`result / error <Object>`**
>
> If successful, an empty object.
>
> If the track cannot be found, an error object (with error code -10) will be returned.
>
> For more details, see [A/344:Various].

Return value

None

Description

`trackSelect()` is a function of `fmw`.

This function sets the audio track of the currently running service.

This function is not currently supported by the emulator.

New DASH Periods may change the audio track list. It is recommended to keep the time between determining the available audio tracks and selecting them to a minimum to avoid incorrect track selection.

Errors

This method will not run in the emulator. Attempting to run it will result in a callback function response of *undefined* and the following error message sent to the logs.

```
a3fa-framework.static.min.js:2 Code: 1000-1006-1018 An event occurred during
operation a3fa.fmwfunc.trackSelect
result: {}
```

Basic usage

The example below selects the second audio track and prints whether this was successful to the console log.

```
fmw.trackSelect( 2, function(result)
      {
        console.log("Result: "
                    + JSON.stringify(result, null, "   ")  );
      }
    );


// Result: {}
```

rmp.start()

The `rmp.start()` method of `fmw` starts the playback of a service using the RMP (Receiver Media Player).

Syntax

```
fmw.rmp.start(source,
              sync,
              callbackFn(error, result)
             );
```

Parameters

`source <String>`

The url of an MPD to present.

`sync <Number>`

The rmpSyncTime property of the `org.atsc.setRMPURL` JSON-RPC message, as detailed in [A/344:Various]. This is the number of seconds to wait before starting playback. If set to zero, playback will start immediately. If set to -1, playback will start when the end of the presentation currently being played by the RMP is reached.

`callbackFn`

A function to execute containing the response. The function is called with the following attributes

> **`error <Object>`**
>
> The error object of the `org.atsc.setRMPURL` JSON-RPC call, as defined by [A/344:Various].
> This will be *null* if no error has occurred.
>
> **`result <Object>`**
>
> The result object of the `org.atsc.setRMPURL` JSON-RPC call, as defined by [A/344:Various].

Return value

None

Description

`rmp.start()` is a function of `fmw`.

This function starts the playback of a service using the RMP (Receiver Media Player) by making use of the `org.atsc.setRMPURL` ATSC 3.0 JSON-RPC message.

Basic usage

The example below starts a new video playing back in the RMP. Please update the `source` url to be a valid MPD before attempting to run this example.

```
fmw.rmp.start("https://example.com/video.mpd",
              0,
              function()
              {
                console.log("Playback started.");
              }
);

// Playback started.
```

rmp.play()

The `rmp.play()` method of `fmw` starts the playback of a service using the RMP (Receiver Media Player) using the current MDP.

Syntax

```
fmw.rmp.play(position,
             callbackFn(error, result)
            );
```

Parameters

`position <Number>`

The position in the media timeline to start playback, measured in seconds from the start of the media.

`callbackFn`

A function to execute containing the response. The function is called with the following attributes:

> **`error <Object>`**
>
> The error object of the `org.atsc.setRMPURL` JSON-RPC call, as defined by [A/344:Various].
> In addition, if an invalid position value is supplied, the following error will be provided:
>
> ```
> {message: "invalid position value", position: position}
> ```
>
> This error object will be *null* if no error has occurred.
>
> **`result <Object>`**
>
> The result object of the `org.atsc.setRMPURL` JSON-RPC call, as defined by [A/344:Various].

Return value

None

Description

`rmp.play()` is a function of `fmw`.

This function starts the playback of a service using the RMP (Receiver Media Player) using the current MDP.

Basic usage

The example below starts playing back media in the RMP, 100 seconds into the media

```
fmw.rmp.play(100,
             function(result)
             {
               console.log("Result is: " + result);
             });

// undefined
```

rmp.pause(), rmp.stop() and rmp.resume()

The `rmp.pause()`, `rmp.stop()` and `rmp.resume()` methods of `fmw` are used for trickplay control of the RMP (Receiver Media Player).

Syntax

```
fmw.rmp.pause(callbackFn(error, result));
fmw.rmp.stop(callbackFn(error, result));
fmw.rmp.resume(callbackFn(error, result));
```

Parameters

`callbackFn`

A function to execute once the trickplay action has been performed. The function is called with the following attributes:ion to execute containing the response. The function is called with the following attributes:

`error <Object>`

The error object of the `org.atsc.setRMPURL` JSON-RPC call, as defined by [A/344:Various].
This error object will be *null* if no error has occurred.

`result <Object>`

The result object of the `org.atsc.setRMPURL` JSON-RPC call, as defined by [A/344:Various].

Return value

None

Description

`rmp.pause()`, `rmp.stop()` and `rmp.resume()` are functions of `fmw`.

These functions perform trickplay actions on media playing on the RMP (Receiver Media Player).

`rmp.pause()` will immediately pause the current playback. If the content includes video, the last presented frame will remain on screen. Running `rmp.pause()` whilst the content is paused will result in no action.

`rmp.stop()` will immediately stop the current playback and hide the media.

`rmp.resume()` will immediately resume playback from either a stopped or paused state.

Basic usage

The example below pauses, resumes and then stops RMP playback.

```
fmw.rmp.pause( ()=>{ console.log("Video paused"); });

// Video paused

fmw.rmp.resume( ()=>{ console.log("Video resumed"); });

// Video resumed

fmw.rmp.stop( ()=>{ console.log("Video stopped"); });

// Video stopped
```

rmp.scale()

The `rmp.scale()` method of `fmw` is used to adjust the on-screen size and position of the RMP (Receiver Media Player) content.

Syntax

```
fmw.rmp.scale(scaleObject,
              callbackFn(error, result)
             );
```

Parameters

`scaleObject <Object>`

An object that describes the scale and positioning of the RMP, the required fields are summarized below:

> **`scaleFactor : <Number>`**
>
> The percentage to scale the video from 10.0% to 100.0%.
>
> **`xPos`**
>
> The x-position of the video.
>
> **`yPos`**
>
> The y-position of the video.

`callbackFn`

A function to execute containing the response. The function is called with the `error` and `response` objects of `org.atsc.scale-position` as detailed in [A/344:Various] and summarized below:

> **`error <Object>`**
>
> This will be *null* if no error has occurred.
>
> **`result <Object>`**
>
> This will be an empty object if no error has occured

Return value

None

Description

`rmp.scale()` is a function of `fmw`.

This function sets the on-screen scale and position of the RMP, making use of the `org.atsc.scale-position` JSON-RPC message.

Full screen video can be restored by calling this method with a `scaleFactor` of 100, an `xPos` of 0 and a `yPos` of 0.

Basic usage

The example below sets the RMP to quarter screen in roughly the center of the screen.

```
fmw.rmp.scale(
                {"scaleFactor" : 50, xPos: 300, yPos: 300},
                  function(err)
                   {
                      console.log("Error: " + JSON.stringify(err, null, "   "))
                   }
              );

// Error: null
```

rmp.time()

The `rmp.time()` method of `fmw` returns the current playback time information of media in the RMP (Receiver Media Player)

Syntax

```
fmw.rmp.time(callbackFn(error, result));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the `error` and `response` objects of `org.atsc.query.rmpMediaTime` as detailed in [A/344:Various] and summarized below:

`error <Object>`

An empty object is returned on success.

`result <Object>`

Containing the following fields:

`currentTime : <Number>`

The number of seconds since the current event started. This value may be a non-integer value.

`startDate : <String>` *`Optional`*

The start date and time of the current event. This value is an ISO-8601 timestamp.

Return value

None

Description

`rmp.time()` is a function of `fmw`.

This function returns the result of the ATSC 3.0 JSON-RPC WebSocket call `org.atsc.query.rmpMediaTime`.

Basic usage

The example below gets the current time information for media in the RMP.

```
fmw.rmp.time( function(err, result)
  {
    console.log("Time: " + JSON.stringify(result, null, "   "));
  }
);

// Time: {
//   "startDate": "2023-11-25T18:21:16.647Z",
//   "currentTime": 1462.879
// }
```

rmp.state()

The `rmp.state()` method of `fmw` returns the current playback state of the RMP (Receiver Media Player).

Syntax

```
fmw.rmp.state(callbackFn(error, result));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the `error` and `result` objects of `org.atsc.query.rmpPlaybackState` as detailed in [A/344:Various] and summarized below:

`error <Object>`

An empty object is returned on success.

`result <Object>`

Containing the following field:

`playbackState: <Number>`

The current RMP playback state. A summary of the valid states is provided below. Please refer to [A/344:Various] for more details.

State	Description
-1	Initializing, or connecting, or unknown
0	Playing
1	Paused (for any reason other than ending)
2	Playback has ended
3	Content is encrypted and unplayable

Return value

None

Description

`rmp.state()` is a function of `fmw`.

This function returns the current playback state of the RMP (Receiver Media Player).

Basic usage

The example below gets the playback state, then pauses the content and gets the playback state again.

```
fmw.rmp.state((err, result) => {console.log(JSON.stringify(result, null, "  "));});

// {
//   "playbackState": 0
// }

fmw.rmp.pause(()=>{ console.log("Video paused"); });

// Video paused

fmw.rmp.state((err, result) => {console.log(JSON.stringify(result, null, "  "));});

// {
//   "playbackState": 1
// }
```

rmpPlaybackRateChange()

The `rmpPlaybackRateChange()` event is fired when the playback rate of the RMP (Receiver Media Player) changes.

To make use of this event, you must first subscribe to it, using the `org.atsc.subscribe` JSON-RPC message (as detailed in [A/344:Various]), with the `msgType` value of `rmpPlaybackRateChange`.

Syntax

```
fmw.rmpPlaybackRateChange(callbackFn(result));
```

The callback function includes the data as defined by rmpPlaybackStateChange in [A/344:Various].

rmpPlaybackStateChange()

The `rmpPlaybackStateChange()` event is fired when the playback state of the RMP (Receiver Media Player) changes.

To make use of this event, you must first subscribe to it, using the `org.atsc.subscribe` JSON-RPC message (as detailed in [A/344:Various]), with the `msgType` value of `rmpPlaybackStateChange`.

Syntax

```
fmw.rmpPlaybackStateChange(callbackFn(error, result));
```

The callback function includes the same data as the callback in the `fmw.rmp.state()` function.

rmpMediaTimeChange()

The `rmpMediaTimeChange()` event is fired at regular intervals during RMP media playback.

In order to receive these events, the application must first subscribe to it, using the `org.atsc.subscribe` JSON-RPC message (as detailed in [A/344:Various]), with `msgType` value of `rmpPlaybackTimeChange`.

Some devices will provide this event without the developer subscribing to it.

Syntax

```
fmw.rmpMediaTimeChange(callbackFn(error, result));
```

The callback function includes the same data as the callback in the `fmw.rmp.time()` function.

queryService()

The `queryService()` method of `fmw` returns information about the current ATSC 3.0 service.

Syntax

```
fmw.queryService(callbackFn(serviceObject));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with a single object containing the following fields:

`serviceObject <Object>`

`service : <Global Service ID>`

The service ID of the current service.

`shortServiceName : <String>`

The short service name of the service, as defined by the ATSC 3.0 broadcast configuration.

`majorChannelNo: <Number>`

The major channel number of the service, as defined by the ATSC 3.0 broadcast configuration.

`minorChannelNo: <Number>`

The minor channel number of the service, as defined by the ATSC 3.0 broadcast configuration.

`ccEnabled : <Boolean>`

True if the service includes closed captions. False if not.

This value does not confirm that closed captions are available for the current event, just that the service has the ability to offer closed captions.

Return value

None

Description

`queryService()` is a function of `fmw`.

This function uses the ATSC 3.0 `org.atsc.query.service` JSON-RPC method to query the current ATSC 3.0 service.

Basic usage

The example below prints details of the current service to the console log.

```
fmw.queryService( function(serviceData)
  {
    console.log("The current service is:\n" + JSON.stringify(serviceData, null, "   "));
  }
);

// The current service is:
// {
//   "service": "tag:run3tv.org,2023:globalServiceID/3",
//   "shortServiceName": "RYML-3",
//   "majorChannelNo": 1,
//   "minorChannelNo": 3,
//   "ccEnabled": true
// }
```

acquireService()

The `acquireService()` method of `fmw` attempts to tune to a different ATSC 3.0 service in the receiver's channel list.

Syntax

```
fmw.acquireService(gSID,
                   callbackFn(result, error)
                  );
```

Parameters

`globalServiceID <String>`

The Global Service Identifier (GSID) of the service to tune to.

`callbackFn`

A function to execute containing the response. The function is called with two objects as summarized below:

> **`result <Object>`**
>
> The ATSC 3.0 response object as defined by § 9.7.1 of [A/344:Various].
>
> **`error <Object>`**
>
> An ATSC 3.0 JSON-RPC result object, as described in § 9.7.6 of [A/344:Various].

Return value

None

Description

`acquireService()` is a function of `fmw`.

This function uses the ATSC 3.0 "`org.atsc.acquire.service`" JSON-RPC method to tune to a ATSC 3.0 service.

Basic usage

The example below tunes to a service. Please update the GSID to be a value already present in the receiver's service list. Be sure to choose a GSID that is different to the currently running service.

```
fmw.acquireService("tag:run3tv.org,2023:globalServiceID/3", function(result, err)
  {
    console.log("The result:\n"
            + JSON.stringify(result, null, "  ")
            + "\nThe error:\n"
            + JSON.stringify(err, null, "  ") );
  }
);

// The result:
// undefined
// The error:
// {}

// ------ Running the same command a second time will result in the following response

// The result:
// {
//   "jsonrpc": "2.0",
//   "error": {
//     "code": -6,
//     "message": "Can not change to current service, please use a different service id"
//   },
//   "id": 40
// }
// The error:
// undefined
```

serviceConf()

The `serviceConf()` method of `fmw` returns the properties object of the current application.

Syntax

```
fmw.serviceConf(callbackFn(appProperties));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with one object, as summarized below.

> **`appProperties <Object>`**
>
> The properties object of the currently running application. This is defined by either **appsList.json**, **bridge.json** or **appSchedule.json**.

Return value

None

Description

`serviceConf()` is a function of `fmw`.

This function returns the properties object of the currently running application. Please refer to [R3TV-IG-0204] for more information.

Basic usage

The example below gets the properties of the currently running application.

```
fmw.serviceConf( function(properties)
  {
    console.log("Properties:\n" + JSON.stringify(properties, null, "  "));
  }
);

// Properties:
// {
//   "ga-id": "G-LG9QZ89357",
//   "tracking": true
// }
```

15. Event stream related

events.subscribe()

The `event.subscribe()` function of `fmw` subscribes to DASH EventStreams.

Syntax

```
fmw.events.subscribe(schemeIdURI,
                     onReceive, // Optional
                     callbackFn(error, result)
                    );
```

Parameters

`schemeIdURI <String>`

The scheme ID URI of the EventStream to subscribe to.

`onReceive <Boolean>` *`Optional`*

If set to `true`, events will be announced using `onStreamEvent()` at the point at which they are received.

Otherwise, events will be announced using `onStreamEvent()` at the point at which they are scheduled.

If not provided, this value will default to `false`.

`callbackFn`

A function to execute containing the response. The function is called with a two objects as detailed below:

> **`errorObject`**
>
> The `error` object of `org.atsc.eventStream.subscribe`, as defined by [A/344:Various].
>
> If no error is generated, this value shall be `null`.
>
> **`resultObject`**
>
> The `result` object of `org.atsc.eventStream.subscribe`, as defined by [A/344:Various]. A successful subscription will result in an empty object here.

Return value

None

Description

`event.subscribe()` is a function of `fmw`. It registers for events in the RMP MPD that contain the schemeIdURI as supplied.

Basic usage

The example below registers for all events that have a scheme ID URI of “urn:example:broadcastEvent123”. Events here will be fired as soon as they are received.

```
fmw.events.subscribe("urn:example:broadcastEvent123",
                     true,
                     console.log);

// {}
```

The example below registers for all events that have a scheme ID URI of “urn:example:broadcastEvent567”. Events here will be fired at the point in time when they are scheduled to occur.

```
fmw.events.subscribe("urn:example:broadcastEvent567",
                     false,
                     console.log);

// {}
```

events.unsubscribe()

The `event.unsubscribe()` function of `fmw` unsubscribes to DASH EventStreams.

Syntax

```
fmw.events.unsubscribe(schemeIdURI,
                       callbackFn(error, result)
                      );
```

Parameters

`schemeIdURI <String>`

The scheme ID URI of the EventStream to unsubscribe from.

`callbackFn`

A function to execute containing the response. The function is called with a two objects as detailed below:

> **`error <Object>`**
>
> The `error` object of `org.atsc.eventStream.unsubscribe`, as defined by [A/344:Various].
>
> If no error is generated, this value shall be `null`.
>
> If the attempt to unsubscribe is unsuccessful, this value shall be `undefined`.
>
> **`result <Object>`**
>
> The `result` object of `org.atsc.eventStream.unsubscribe`, as defined by [A/344:Various].
>
> A successful subscription will result in an empty object here.
>
> If the attempt to unsubscribe is unsuccessful, this value shall be `undefined`.

Return value

None

Description

`event.unsubscribe()` is a function of `fmw`. It unsubscribes for events in the RMP MPD that contain the schemeIdURI as supplied.

Basic usage

The example below unsubscribes for all events that have a scheme ID URI of “urn:example:broadcastEvent123”:

```
fmw.events.unsubscribe("urn:example:broadcastEvent123",

function(err, result){
  console.log("Result is: " + result);
});
```


event.list()

The `event.list()` function of `fmw` returns a list of known events for a given event scheme ID.

Syntax

```
fmw.events.list(schemeIdURI,
                active,  // Optional
                callbackFn(error, result)
               );
```

Parameters

`schemeIdURI <String>`

The scheme ID URI of the EventStream to unsubscribe from.

`active <Boolean>` *`Optional`*

If true, only currently active events will be returned. Otherwise, all known events will be returned.

By default, this value is `false`.

`callbackFn`

A function to execute containing the response. The function is called with a two objects as detailed below:

`error <Object>`

`Null` if no error has occurred.

`result <Array of MPDEvent objects>`

The array of `MPDEvent` objects, each with the following structure:

`schemeIdUri : <String>`

The scheme ID URI of the MPD event.

`value : <String>`

The value field of the MPD event.

`eventTime : <Number>`

The MPD event's time, as based on the RMP's timeline.

`duration : <Number>`

The duration (in seconds) of the MPD event.

`id : <Number>`

The identifier of the MPD event.

`data : <String>`

The custom data associated with the MPD event. This value is URI encoded.

Return value

None

Description

`event.list()` is a function of `fmw`. It lists all known events for the current MPD in the RMP. The `events.subscribe()` function must be used first to register the `schemeIdURI`.

Basic usage

The example below lists all events that have a scheme ID URI of “urn:example:broadcastEvent123”:

```
// First, register for the events
fmw.events.subscribe("urn:example:broadcastEvent123", true, console.log);

// Then, get the list of known events
fmw.events.list("urn:example:broadcastEvent123", false,
  function(err, result){
    console.log("Result is:\n" + result);
  }
);

// Result is:
// [
//   {
//     "schemeIdUri" : "urn:example:broadcastEvent123",
//     "value" : "ABC",
//     "eventTime" : 1697293431,
//     "duration" : 5,
//     "id" : 123,
//     "data" : "Data1234"
//   }
// ]
```

onStreamEvent()

The `onStreamEvent()` event is fired for any RMP MPD event that has been subscribed to using `events.subscribe()`.

Syntax

```
fmw.onStreamEvent(callbackFn(MPDeventData));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with a two objects as detailed below:

> **`MPDEvent <Object>`**
>
> For details of the `MPDEvent` object, see `event.list()`.

Timing of this event is based on the value of the `events.subscribe()`'s `onReceive` parameter.

If `onReceive` is set to `true`, `onStreamEvent()` will fire as soon as a new event with the same event scheme ID is received. Otherwise, `onStreamEvent()` will fire at the point that the event is scheduled to occur.

For more details, please refer to [R3TV-IG-0234].

onAppEvent()

The `onAppEvent()` event is fired whenever a Framework Application event for the current application is supplied. For more details, please refer to [R3TV-IG-0234].

Syntax

```
fmw.onAppEvent(callbackFn(data));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with a two objects as detailed below:

`data : <String>`

The data associated with the event. For more details, please refer to [R3TV-IG-0234].

Application properties data in the `propertiesBase58` field is base58 encoded.

16. Streaming content (AMP) related

amp.start()

The `amp.start()` method of `fmw` stops playback of RMP and then informs the AMP (Application Media Player) to prepare to present a piece of content.

Syntax

```
fmw.amp.start(source, callbackFn(error, result));
```

Parameters

`source <String>`

The url of an MPD to present.

`callbackFn`

A function to execute containing the response. The function is called with the following attributes:

`error <Object>`

If successful, this will be *null*.

> Note This function does not verify that the source URL is valid. If it is invalid, no error will be provided in this response.

`result <Object>`

An object with the following fields:

`operation : <String>`

An enum. Permitted values are:

- `startAmp`

`ampurl : <String>`

The source URL, as set by the source parameter of `amp.start()`.

Return value

None

Description

`amp.start()` is a function of `fmw`.

This function informs the AMP (Application Media Player) to prepare to present a piece of content. Calling this function will present the “loading” poster image on screen but not start playback. This functionality differs from that of `rmp.start()`.

The “loading” poster image can be updated by calling `amp.setposter()` in advance of calling `amp.start()`.

Basic usage

The example below prepares the AMP to start playback of a video asset.

```
fmw.amp.start("https://example.com/video.mp4", console.log)
```

The example below prepares the AMP to start playback of a video asset and then, when ready, uses the callback function to start playback.

```
fmw.amp.start("https://example.com/video.mp4", fmw.amp.play)
```

amp.play()

The `amp.play()` method of `fmw` starts the playback of a service using the AMP (Receiver Media Player) using the media provided by `amp.start()`.

Syntax

```
fmw.amp.play(position, callbackFn(result, error));
```

Parameters

`position <Number>` *Optional*

The position in the media timeline to start playback, measured in seconds from the start of the media.

`callbackFn`

A function to execute containing the response. The function is called with the following attributes:

`result <Object>`

An object taking the form of either *null*, or:

`<Numeric Identifier> : <Object>`
This object includes the following fields:

`operation : <String>`
An enum. Permitted values are:

- `playAmp`

`position : <Number>`
The current playback position. This is the previous playback position, not the newly requested `position` value.

`error <Object>`

Set to *null* if no error has occurred.

Return value

None

Description

`amp.play()` is a function of `fmw`.

This function starts the playback of a service using the AMP (Application Media Player) using the media provided by `amp.start()`.

Basic usage

The example below starts playing back media in the AMP, 100 seconds into the media

```
fmw.amp.start("https://example.com/video.mp4", console.log)

fmw.amp.play(100,
  function(result)
  {
    console.log("Result is: " + JSON.stringify(result, null, "  "))
  });

//  {
//   "0": null,
//   "1": {
//     "operation": "playAmp",
//     "position": 1701210720
//   }
```

amp.stop()

The `amp.stop()` method of `fmw` stops playback of content in the AMP (Application Media Player) and resets the player.

Syntax

```
fmw.amp.stop(callbackFn());
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with no attributes.

Return value

None

Description

`amp.stop()` is a function of `fmw`.

This function stops playback, resets the AMP (Application Media Player) and restarts RMP (Receiver Media Player) playback.

The following actions take place:

- The stream is stopped
- The video element is reset
- The source is reset
- Any subtitle (caption) tracks are deleted
- rmp.resume() is called.

If you wish to pause playback of AMP content, do not use `amp.stop()`. Rather, use `amp.pause()`, as this will not reset AMP playback.

Basic usage

The example below demonstrates the use of `amp.stop()`.

```
fmw.amp.stop(console.log)
```

amp.state()

The `amp.state()` method of `fmw` returns the current playback state of the AMP (Application Media Player).

Syntax

```
fmw.amp.state(callbackFn(error, result));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with two objects.

`error <Object>`

Set to *null* if no error has occurred.

`result <Object>`

Provides details of the state of the AMP, with the following field:

`playbackState: <Number>`

The current AMP playback state:

State	Description
-1	Initializing, or connecting, or unknown
0	Playing
1	Paused (for any reason other than ending)
2	Playback has ended
3	Content is encrypted and unplayable

Return value

None

Description

`Amp.state()` is a function of `fmw`.

This function returns the current playback state of the AMP (Application Media Player).

Basic usage

The example below gets the playback state, then pauses the content and gets the playback state again.

```
fmw.amp.state((err, result) => {console.log(JSON.stringify(result, null, "  "));});

// {
//   "playbackState": 0
// }

fmw.amp.pause(()=>{ console.log("Video paused"); });

// Video paused

fmw.amp.state((err, result) => {console.log(JSON.stringify(result, null, "  "));});

// {
//   "playbackState": 1
// }
```

amp.getCurrentTime()

The `amp.getCurrentTime()` method of `fmw` returns the current playback position of the AMP (Application Media Player).

Syntax

```
fmw.amp.getCurrentTime(callbackFn(currentPlaybackTime));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following attribute:

`currentPlaybackTime <Number>`

The position (in seconds) of playback of the AMP.
If the AMP is in a reset state, this value will be zero.

Return value

None

Description

`amp.getCurrentTime()` is a function of `fmw`.

This function returns the current playback position of the AMP (Application Media Player).

Basic usage

The example returns the current playback position of media in the AMP..

```
fmw.amp.getCurrentTime(console.log);

// 23.424
```

amp.setCurrentTime()

The `amp.setCurrentTime()` method of `fmw` sets the current playback position of the AMP (Application Media Player).

Syntax

```
fmw.amp.setCurrentTime(time,
                       callbackFn(currentPlaybackTime)
                      );
```

Parameters

`time <Number>`

The new playback time (in seconds)

`callbackFn`

A function to execute containing the response. The function is called with the following attribute:

> **`currentPlaybackTime <Number>`**
>
> The position (in seconds) of playback of the AMP.
> This will match the `time` parameter.

Return value

None

Description

`amp.setCurrentTime()` is a function of `fmw`.

This function sets the current playback position of the AMP (Application Media Player).

Basic usage

The example returns the current playback position of media in the AMP.

```
fmw.amp.setCurrentTime(2, console.log);

// 2
```

amp.setposter()

The `amp.setposter()` method of `fmw` sets the “Loading” screen poster image of the AMP (Application Media Player).

Syntax

```
fmw.amp.setposter(imageSrc,
                  callbackFn(imgSrc)
                 );
```

Parameters

`imageSrc <String>`

The location of the poster image.

`callbackFn`

A function to execute containing the response. The function is called with the following attribute:

> **`imageSrc <String>`**
>
> The `imageSrc` value, as supplied as a parameter to `amp.setposter()`.

Return value

None

Description

`amp.setposter()` is a function of `fmw`.

This function sets the “Loading” screen Poster image of the AMP (Application Media Player). Any profiled image type can be used. It is recommended to ensure that the images are an appropriate resolution and file size.

Basic usage

The example sets the “Loading” screen poster image of the AMP. Please replace the example URL with a valid URL before running this example.

```
fmw.amp.setposter("http://example.com/image.jpg", console.log);

// http://example.com/image.jpg
```

amp.getposter()

The `amp.getposter()` method of `fmw` gets the “Loading” screen poster image of the AMP (Application Media Player).

Syntax

```
fmw.amp.getposter(callbackFn(imageSrc));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following attribute:

`imageSrc <String>`

The URL of the “Loading” screen poster image.
If this has not been set by `amp.setposter()` this value will be *undefined*.

Return value

None

Description

`amp.getposter()` is a function of `fmw`.

This function gets the “Loading” screen Poster image of the AMP (Application Media Player).

Basic usage

The example gets the “Loading” screen poster image of the AMP.

```
fmw.amp.getposter(console.log);

// http://example.com/image.jpg
```

amp.loop()

The `amp.loop()` method of `fmw` sets the looping state of the AMP (Application Media Player).

Syntax

```
fmw.amp.loop(loopingState,
             callbackFn(loopingState)
            );
```

Parameters

`loopingState <Boolean>`

The looping state of the content in the AMP.

If set to `true`, the content in the AMP will repeatedly loop. Otherwise the content in the AMP will play to the end, and then stop.

`callbackFn`

A function to execute containing the response. The function is called with the following attribute:

> **`loopingState <Boolean>`**
>
> The `looping` value, as supplied as a parameter to `amp.loop()`.

Return value

None

Description

`amp.loop()` is a function of `fmw`.

This function sets the looping state of the AMP (Application Media Player).

Basic usage

The example sets the AMP to loop its content.

```
fmw.amp.loop(true, console.log);

// true
```

amp.isLoop()

The `amp.isLoop()` method of `fmw` returns the looping state of the AMP (Application Media Player).

Syntax

```
fmw.amp.isLoop(callbackFn(loopingState));
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the following attribute:

`loopingState <Boolean>`

True if the AMP has been configured to loop its current content. False if not.

Return value

None

Description

`amp.isLoop()` is a function of `fmw`.

This function gets the looping state of the AMP (Application Media Player).

Basic usage

The example gets the AMP to loop its content.

```
fmw.amp.isLoop(console.log);

// false
```

ampMediaTimeChange()

The `ampMediaTimeChange()` event is fired at regular intervals during AMP media playback.

The current position of playback can be found using the `fmw.amp.getCurrentTime()` function.

ampPlaybackStateChange()

The `ampPlaybackStateChange()` event is fired when the playback state of the AMP changes.

The current state of the AMP can be found using the `fmw.amp.state()` function.

17. Alerting related

aeat()

The `aeat()` function of `fmw` gets the current AEAT (Advanced Emergency Alert Table) XML data.

Syntax

```
fmw.aeat(callbackFn);
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with a single object containing the following arguments:

> **`String`**
>
> The current AEAT XML, as defined by § 6.5.1 of [A/331:2023-10].

Return value

None

Description

`aeat()` is a function of `fmw`. It gets the current AEAT XML data.

Basic usage

The example below gets the AEAT XML data. In this case, no alerts are signaled.

```
fmw.aeat(function(aeatXML)
  {
    console.log("AEAT XML is: " + aeatXML);
  }
);

// AEAT XML is: <?xml version="1.0" encoding="UTF-8"?><AEAT/>
```

onAlert()

The `onAlert()` event is fired when a new alert (that the viewer has subscribed to using `fmw.input.preference()` or equivalent) is received.

Syntax

```
fmw.onAlert(callbackFn);
```

Parameters

`callbackFn`

A function to execute containing the response. The function is called with the result of `alertList` from the `org.atsc.notify` `alertingChange` message type. This is summarized below:

> **`alertList <Array of Alert objects>`**
>
> An array of the alerts, with the following structure:
>
> **`alertingType : <String>`**
>
> Either `AEAT` or `OSN`, as defined by [A/344:Various].
>
> **`alertingFragment : <String>`**
>
> The XML fragment describing the alert, as defined by [A/344:Various].

Return value

None

Description

`fmw.onAlert()` is an event of `fmw`.

This event fires when a new alert (or alerts) are received.

The alerting information itself is returned in XML format. The structure of this data is defined in [AEA-IT-024r31].

Basic usage

The example below is fired when new alerts are received.

```
fmw.onAlert(function(alertList){
      console.log("Alert List: " + JSON.stringify(alertList, null, "  ") )
    }
);

// Alert List:
// [
//   { "alertingType": "AEAT",
//   "alertingFragment": "<AEAT>…</AEAT>" },
//   { "alertingType": "OSN",
//   "alertingFragment": "<OSN>…</OSN>",
//   "receiveTime": "2023-19-01T10:00:00.000Z" }
// ]
```

alerting()

The `alerting()` method of `fmw` returns the various alerting metadata from the current ATSC 3.0 broadcast.

Syntax

```
fmw.amp.alerting(typeArray,  //Optional
                 callbackFn(error, result)
                );
```

Parameters

`typeArray`

An array of alerting types. Permitted values are:

- AEAT
- OSN

If not present, all values listed above will be used.

`callbackFn`

A function to execute containing the response. The function is called with two objects as summarized below:

> **`error <Object>`**
>
> Set to *null* if no error has occurred.
>
> **`result <Object>`**
>
> Provides the various alerting metadata in the current broadcast. It takes the form of a `result` object from the ATSC 3.0 JSON-RPC WebSocket call `org.atsc.query.alerting`.
>
> For more details see [A/344:Various].

Return value

None

Description

`alerting()` is a function of `fmw`.

This function returns the various alerting metadata from the current ATSC 3.0 broadcast.

The alerting information itself is returned in XML format. The structure of this data is defined in [AEA-IT-024r31].

Emulator configuration

To make use of this function in the emulator, the appropriate **emulator/atscCmd-*.mc.json** file must include AEAT configuration information.

Basic usage

The example below gets the alerting metadata from the current broadcast.

```
fmw.alerting(function(err, result)
  {
    console.log("Alerting metadata is:\n"
                + JSON.stringify(result, null, "   "));
  }
);

// Alerting metadata is:
// {
//   "alertList": [
//     { "alertingType": "AEAT",
//       "alertingFragment": "<AEAT>…</AEAT>"
//     },
//     {
//       "alertingType": "OSN",
//       "alertingFragment": "<OSN>…</OSN>",
//       "receiveTime": "2023-11-20T10:00:21.000Z"
//     }
//   ]
// }
```

18. Data processing related

ajax()

The `ajax()` function of `fmw` requests XML data from a URL and responds with an AJAX (`XMLHttpRequest`) object.

Syntax

```
fmw.ajax(url || pathObject,
         callbackFn(response)
        );
```

Parameters

Either

`url`

The URL of an XML file. The default timeout value of 30 seconds will be used.

Or

`pathObject`

An object that includes a URL and optional timeout length, as summarized below:

> **`url : <String>`**
>
> The URL of an XML file.
>
> **`timeout : <Number>`** ***`Optional`***
>
> The amount of time (in seconds) to wait for a response before stopping.
> If set to zero, no timeout will be used.
> If not present, the default timeout of 30 seconds will be used.

`callbackFn`

A function to execute containing the response. This function is called with a single object, as summarized below:

> **`response <XMLHttpRequest>`**
>
> An `XMLHttpRequest` object.

Return value

None

Description

`ajax()` is a function of `fmw`. It requests XML data from a URL and provides an `XMLHttpRequest` object in response.

Basic usage

The example below gets XML data from one of the example ticker files in the Starter Kit and responds with an XMLHttpRequest object. For brevity this object is not shown.

```
fmw.ajax("http://localhost:5001/tv%3Aa3fa-apps.yottamedialabs.com/london/Station-3/q-bar/dynamic/ticker-en.atom", console.log)

// <XMLHttpRequest object>
```

mediaParse()

The `mediaParse()` function of `fmw` requests RSS feeds (as profiled by [R3TV-IG-0231]) from a URL and responds with a JSON Feed object.

Syntax

```
fmw.mediaParse(url || pathObject,
               callbackFn(response)
              );
```

Parameters

Either

`url`

The URL of a [R3TV-IG-0231] profiled RSS (Atom) feed. The default timeout value of 30 seconds will be used.

Or

`pathObject`

An object that includes a URL and optional timeout length, as summarized below:

`url : <String>`

The URL of a [R3TV-IG-0231] profiled RSS (Atom) feed. A timeout of 30 seconds will be used.

`timeout : <Number>` ***`Optional`***

The amount of time (in seconds) to wait for a response before stopping.
If set to zero, no timeout will be used.
If not present, the default timeout of 30 seconds will be used.

`callbackFn`

A function to execute containing the response. This function is called with a single object, as summarized below:

`response <JSON RSS object>`

A JSON Feed object, as profiled by [R3TV-IG-0231].

If this cannot be generated, an empty object is returned.

Return value

None

Description

`mediaParse()` is a function of `fmw`. It requests RSS (Atom) feeds as profiled by [R3TV-IG-0231] from a URL and responds with a JSON Feed object.

This function makes use of fmw.`ajax()` to download the data.

Basic usage

The example below gets XML data from one of the example ticker files in the Starter Kit and responds with the resultant JSON Feed object. For brevity this object is not shown in full.

```
fmw.mediaParse("http://localhost:5001/tv%3Aa3fa-apps.yottamedialabs.com/london/Station-3/q-bar/dynamic/ticker-en.atom",
          function(jsonFeed)
            {
              console.log("JSON feed:\n" + JSON.stringify(jsonFeed, null, "  "));
            }
)

// JSON feed:
// {
//   "items": [
//     {
//       "media:group": {
//         "media:content": [
//           ...
```

legals()

The `legals()` function of `fmw` requests the terms of service (or privacy policy) documents and performs the tag replacement as defined in the **appsList.json** file.

Syntax

```
fmw.legals(filename,
           LegalID, // Optional
           callbackFn(generatedLegals)
           );
```

Parameters

`filename <String>`

The filename of the legal file to parse. This must not include any filename extension.

`LegalID <String>` *`Optional`*

The LegalID value as defined in **appsList.json**. For more details, see [R3TV-IG-0204]. If this is not supplied, the "*default*" `legalID` will be used.

`callbackFn`

A function to execute containing the response. This function is called with a single object, as summarized below.

> **`generatedLegals <String>`**
>
> The legal data with associated string substitutions.
>
> If this cannot be generated, *undefined* is returned.

Return value

None

Description

`legals()` is a function of `fmw`. It takes as input the filename of a legal document and searches and replaces the tokens as defined by the `match-keys` data within the **appsList.json** file for the service.

In other words, it replaces the {name}, {address}, {email} and {date} tags with the values supplied within "`name`", "`address`", "`email`" and "`date`" of the legals object in appsList.json respectively.

Basic usage

The example below generates the terms of service text using the “custom” `legalID`. Before running this example, please ensure that this custom `legalID` exists in appsList.json. For brevity this object is not shown in full.

```
fmw.legals("terms-of-service", "custom",
          function(jsonFeed)
            {
              console.log("Generated Legals:\n" +
                JSON.stringify(jsonFeed, null, "  "));
            }
)

// Generated Legals:
// "<div class=\"head\">\r\n<p>VIEWER TERMS OF USE</p>\r\n</div>\r\n\r\n
// <div class=\"text\">\r\n\r\n<p>This end-viewer \"Terms Of Use\" for using
// the common application emitted by the Broadcaster shall
```

tagReplace()

The `tagReplace()` function of `fmw` takes a string and substitutes any Replaceable Field Tags (as detailed in [R3TV-IG-0204]) with their replacement values. Additional replacements can also be defined if required.

Syntax

```
fmw.tagReplace(sourceString,
               additionalReplacements, // Optional
               callbackFn(outputString)
              );
```

Parameters

`sourceString <String>`

The string on which to perform the replacement.

`additionalReplacements <Array of replacementObject>` *`Optional`*

An optional array listing any additional string replacements to perform. The structure of replacementObject is shown below:

> **`find : <String>`**
>
> The search tag. Tags must be surrounded by angle brackets (<>) in the `sourceString`, however these brackets are not required in this `find` field.
>
> **`replace: <String>`**
>
> The string that will replace any instances of the `find` search tag found.

`callbackFn`

A function to execute containing the response. This function is called with a single object, as summarized below.

> **`outputString <String>`**
>
> The updated sourceString value with any text replacements made.

Return value

None

Description

`tagReplace()` is a function of `fmw`.

It takes a string and substitutes any Replaceable Field Tags (as detailed in [R3TV-IG-0204]) with their replacement values.

It is possible to include additional replacement searches by including the `additionalReplacements` parameter.

This function can be used with a URL string to generate the most appropriate URL for the viewer or receiver.

Basic usage

The example below takes a URL with Replaceable Field Tags and expands them out based on the type and state of the receiver. The responses below match that of a device in Noble, Indiana.

```
// Replacing standard tags
fmw.tagReplace("http://example.com/feed-<location>.json",
          function(result)
            {
              console.log("Generated URL: " + result);
            }
)

// Generated URL: http://example.com/feed-47371.json
```

The example below takes a URL with Replaceable Field Tags and a custom additional tag “<TAG>”. It expands these out based on the type and state of the receiver

```
// Replacing standard tags and an additional "<TAG>" tag.
fmw.tagReplace("http://example.com/feed-<location>-<TAG>.json",
          [{"find" : "TAG", "replace" : "enhanced"}],
          function(result)
            {
              console.log("Generated URL: " + result);
            }
)

//  Generated URL: http://example.com/feed-47371-enhanced.json
```

19. Deprecated functions

The following functions are deprecated and may be removed in future versions of the Framework.

Deprecated function	Description
`fmw.tos()`	Previously used to verify if a now-deprecated form of onboarding was partially complete - In other words, that the Terms on Service were agreed to, but not the Privacy Policy
`fmw.pp()`	Previously used to verify if a now-deprecated form of onboarding was complete - In other words, that both the Terms on Service and the Privacy Policy were agreed to
`fmw.guideSwitch()`	Previously used to switch between selecting ESG data from ATSC 3.0 or from a web URL for development use
`fmw.rmp.clock()`	Previously used to return the wallclock time of RMP media
`fmw.dai()`	Previously used to access details about dynamic ad insertion data