|  |
| --- |
| Microsoft Corporation |
| RIN JS SDK |
| Developer Documentation v1.0 |
| Microsoft Research India |
|  |
|  |

Table of Contents:

[1 Introduction 2](#_Toc334521406)

[2 Getting Started 2](#_Toc334521407)

[2.1 Downloading RIN JS Library 2](#_Toc334521408)

[2.2 Adding RIN JS library to your HTML page 3](#_Toc334521409)

[2.3 Adding Player Control declaratively 3](#_Toc334521410)

[2.4 Programmatically controlling RIN Player 4](#_Toc334521411)

[3 Experience Provider development 4](#_Toc334521412)

[3.1 Sample 1: Hello World Experience Provider 4](#_Toc334521413)

[3.2 Sample 2: Discrete keyframe based Experience Provider sample 5](#_Toc334521414)

[3.3 Building your Experience Provider 5](#_Toc334521415)

[3.4 Communicating with RIN Core Host 6](#_Toc334521416)

[3.5 Replacing or adding Player Controller experience streams 7](#_Toc334521417)

[4 Using Interaction Controls 7](#_Toc334521418)

[4.1 Using the built-in interaction controls 7](#_Toc334521419)

[4.2 Creating custom interaction controls 7](#_Toc334521420)

[5 Enabling Authoring Experience in RIN Studio 7](#_Toc334521421)

[6 Deploying your Experience Stream 7](#_Toc334521422)

[7 Communication between experience streams 8](#_Toc334521423)

# Introduction

RIN JS is a small javascript library that plays Rich Interactive Narratives (RIN). The source of RIN can be external JSON files, or JSON content given at runtime. This documentation describes how to use RIN JS library to player RIN files and to extend RIN player, for example, with new experience providers.

For more information visit <http://digitalnarratives.net>

# Getting Started

The best way to learn about extending the RIN Player is to download the RIN Player SDK and experiment with it by following below steps.

## Downloading RIN JS Library

RIN JS consists of compressed “rin-core-1.0.js” file containing the core player, “rin-experiences-1.0.js” file containing all the experience streams, sample html pages and sample narratives. <download link etc will come here later>

## Adding RIN JS library to your HTML page

Insert the following lines in header of a new html file or existing html file:

<!-- styles for rin player controller -->

<link rel="stylesheet" type="text/css" href="css/default-player/controller.css"/>

<!—- optional third-party libraries -->

<script src="js/jquery-1.7.2.min.js" type="text/javascript"></script>

<script src="js/seadragon-0.8.9.js" type="text/javascript"></script>

<!-- rin player -->

<script src="js/rin-core-1.0.js" type="text/javascript"></script>

<!-- rin experience streams -->

<script src="js/rin-experiences-1.0.js" type="text/javascript"></script>

Note: When we go to production, we’d require websites to include only two scripts “rin-core-1.0.js”, “rin-experiences-1.0.js” and one css file “rin.css”. These three files will already contain all the required minified library dependencies in them, or these script files will appropriately load all other js and css files as needed. For now, until production, we’ll request users to load all above scripts. For details about run time script loaders, see [this link](http://msdn.microsoft.com/en-us/magazine/hh227261.aspx).

## Adding Player Control declaratively

Here is simple example of adding a player control in html:

<body onload="rin.processAll()">

<div class="rinPlayer" data-src="narratives/sample/narrative.js"></div>

</body>

In this example, there is a div in body with class name “rinPlayer”. The “rin.processAll” call in the onload event causes RIN to look at all contents of the document and embed RIN player when an element has class of “rinPlayer”. If “data-src” attribute points to a valid JSON url, then that JSON RIN is played automatically.

To add options to the player, add “data-options” attribute with options specified in query-string format. For example:

<div style="width:500px" class="rinPlayer" data-src="narratives/sample/narrative.js" **data-options="autoplay=true&loop=true"**></div>

This example sets autoplay to false and loop to true. Now, the player will not start automatically and once started manually, it’ll loop forever. List of options available:

|  |  |  |
| --- | --- | --- |
| Name | Default | Comment |
| autoplay | false | Indicates if RIN should autoplay contents if valid source is specified. |
| controls | false | Indicates if default controls for play/pause/seek etc should be shown. If false, no UI control is shown for users to interact. The player can be manipulated programmatically. |
| loop | false | Indicates if the contents should loop back to start on ending |
| muted | false | Indicates if audio should be muted on startup. |
|  |  |  |

## Creating Player Control programmatically

To create a rin player control programmatically, use the call “rin.createPlayerControl()”. See the methods under “rin” namespace below:

|  |  |  |
| --- | --- | --- |
| Name | Type | Comment |
| createPlayerControl(playerElement, options) | method | Creates a new player control and returns. playerElement is pointer to a div element where the player will draw its contents. Options param is list of options in query string or JSON format. |
| processAll(rootElement) | method | Looks for elements with class “rinPlayer” under rootElement (or Document if not specified) and creates new player controls. |
| getPlayerControl(playerElement) | method | Returns RIN player control associated with a given DOM element. |

Same set of options as specified in previous section can be specified in createPlayerControl options parameter in JSON form. An example:

var p = rin.createPlayerControl(playerElement, {autoplay: true, controls: true, loop: true });

## Programmatically controlling RIN Player

Look at below example:

<script type="text/javascript">

$(document).ready(function () {

rin.processAll(); // This checks for all elements in document with class='rinPlayer' and embeds player.

var myrin = rin.getPlayerControl(document.getElementById('myrin'));

setTimeout(function () { myrin.play(); }, 1000); // play after 1 seconds

setTimeout(function () { myrin.pause(); }, 3000); // pause after 5 seconds

});

</script>

This example calls rin.processAll to bind all RIN elements on load of html document. Then it gets a div element specified by an id and gets its rinPlayer. The rinPlayer object has below public members:

|  |  |  |
| --- | --- | --- |
| Name | Type | Comment |
| play(offset, screenplayId) | method | Causes RIN to play. offset: Optional parameter specifying offset of play in seconds. Default is current position, or 0 at start. screenplayId: optional parameter specifying screenplayId to play. Default is as specified in the xrin JSON file. |
| pause(offset, screenplayId) | method | Causes RIN to pause. The parameters are same as in play method. |
| load(narrativeUrl, onComplete) | method | Loads JSON file located in the given narrativeUrl. onComplete callback is fired after load, if provided. |
| loadData(rinData, onComplete) | method | Loads JSON object passed as rinData. onComplete callback is fired after load, if provided. |
| getPlayerState() | Method | Returns enumeration value of player state. The values under rin.contracts.PlayerState are : Playing, PausedForBuffering, PausedForExplore, Stopped, InTransition |
| getCurrentTimeOffset() | Method | Returns current time offset in seconds. |

# Experience Provider development

Experience providers can be built from scratch or using one of the sample projects as a starting point. It’s best to start by understanding how the sample experience streams work.

## Sample 1: Hello World Experience Provider

<TBD> This will talk about hello-world EP that comes with the SDK.

## Sample 2: Discrete keyframe based Experience Provider sample

<TBD> This section will talk about a sample EP that uses DiscreteExperienceStream as base class.

## Building your Experience Provider

To build your Experience Stream, you can use one of samples as starting point or build your own project. The steps are:

* Add a new js file. If keyframes in your EP simply jumps in state between keyframes, you can save time by inheriting from rin.contracts.DiscreteKeyframeESBase (as described in Sample 2)
* The experience providers should implement below public members.

|  |  |  |
| --- | --- | --- |
| Name | Type | Comments |
| load(keyframeSequenceId) | method | Load experience data during this call. Called only one. keyframeSequenceId param indicates the keyframeSequence that’ll be played first. |
| play(offset, keyframeSequenceId) | method | EP should play the contents at given time offset and keyframeSequenceId. |
| pause(offset, keyframeSequenceId) | method | EP should display the contents at given time offset and keyframeSequenceId and be in paused state. |
| unload() | method | Unload objects loaded in memory and release memory to prepare for removal from stage |
| getState() | method | Return one of valid states from rin.contracts.ExperienceStreamState enum. Values are: Closed, Buffering, Ready, Error. The initial default state should be Closed. The state moves to Buffering when loading is going on. Once the experience provider is ready to play, the state should change to Ready. If any error occurs during loading, the state should change to Error. |
| stateChangedEvent | rin.contracts.Event | Event to be triggered whenever experience provider state changes. The event args should be of type rin.contracts.ESStateChangedEventArgs |
| getUserInterfaceControl() | method | EP should return the HTML element where contents are displayed. If the EP has no user interface, return null. |
| onESEvent(sender, eventId, eventData) | method | This is an optional method to be implemented if communication across experience providers is needed. If implemented, this method is called by RIN core whenever an ES event is raised by other experience providers. |

* The experience provider needs to be registered by calling below method:

rin.Ext.registerESFactory("CompanyName.rin.UniqueProviderName", function (orchestrator, esData) { return new rin.MyExperienceName(orchestrator, esData); });

rin.Ext.registerESFactory takes a unique provider ID as first param. To make the provider ID unique, use this suggested format <CompanyName>.Rin.<ProviderName>. The second parameter is factory function.

The factory function has 2 parameters. First one is “orchestrator” – this is a proxy object to host. See next section for details. The second parameter “esData” is data specific to the EP as given in the xrin JSON file.

## Communicating with RIN Core Host

Experience Providers can communicate with host through “orchestrator” parameter provided in factory. Orchestrator is a proxy object to RIN core and has below members:

|  |  |  |
| --- | --- | --- |
| Name | Type | Comments |
| getCurrentLogicalTimeOffset() | Method | Returns current time offset of player in seconds |
| getRelativeLogicalTime(keyframeSequenceId, absoluteLogicalTime) | Method | Returns time relative current experience provider. Params: keyframeSequenceId: optional param to point to specific keyframeSequence. absoluteLogicalTime: optional absolute time, if not specified, assumed to be current player time. |
| play(offset, screenPlayId) | Method |  |
| pause(offset, screenPlayId) | Method |  |
| getIsMuted() | Method |  |
| setIsMuted(value) | Method |  |
| onESEvent(eventId, eventData) | Method | Call this method from experience provider to send a custom event notification to other experience providers. Standard set of event id are available in rin.contracts.ESEventIds enumeration. |
| getPlayerState() | Method | Returns current player state from rin.contracts.PlayerState enumeration. |
| getIsOnStage() | Method | Returns bool indicating if the current experience provider is playing on stage. |
| resolveReference(resourceItemId) | Method | Returns the resolved URL for the given resourceItemId based on current theme and language locale. |
| getCurrentStateSeekUrl() | Method | Returns a URL that indicates current seek state of player. This url can be later passed to seekUrl call to restore the state. |
| seekUrl(seekUrl) | Method | Navigates to the state as described by the seekUrl. |

## Replacing or adding Player Controller experience streams

<TBD>

# Using Interaction Controls

Interaction controls are the controls that are displayed in the player’s footer when the user interacts with any experience stream. Interaction controls can be custom-made for any experience stream, or the built-in interaction controls can be used for common scenarios.

<details TBD>

## Using the built-in interaction controls

<details TBD>

1. Pan and zoom controls:
2. Selection interaction controls:
3. Audio/video interaction controls:

## Creating custom interaction controls

To create custom interaction controls, you can use one of samples as starting point or build your own project. The steps are:

* Add a html file for adding the UI for the interaction controls
* Add a css file for styling the controls
* The interaction control needs to be registered by calling below method:

rin.ext.registerInteractionControlFactory("MicrosoftResearch.Rin.InteractionControls.RotateControl",function (resourcesResolver, loadedCallback) { $.get(resourcesResolver.resolveSystemResource("interactionControls/RotateControl.html"), null, function (visual) {

var wrap = document.createElement("div"),

systemRoot = resourcesResolver.getSystemRootUrl();

wrap.style["display"] = "inline-block";

wrap.innerHTML = visual.replace(/SYSTEM\_ROOT/g, systemRoot);

loadedCallback(wrap);

});

});

rin.Ext.registerInteractionControlFactory takes a unique interaction control ID as first param. To make the interaction control ID unique, use this suggested format <CompanyName>.Rin.InteractionControls.<InteractionControlName>. The second parameter is factory function.

The factory function has 2 parameters. First one is “resourcesResolver” – this is a proxy object to the resource resolving mechanism used by the player. This can be used to get full path to system resources by passing in relative paths. The second parameter “loadedCallback” is a callback to the experience stream that requested the interactionControl, signaling the completion of load and thus returning the html element.

* In the experience stream that has to display this interaction control, implement the following public method

this.getInteractionControls = function () {

var interactionControls = document.createElement("div");

self.\_orchestrator.getInteractionControls(

["MicrosoftResearch.Rin.InteractionControls.RotateControl", rin.contracts.interactionControlNames.panZoomControl],

function (wrappedInteractionControls) {

interactionControls.innerHTML = wrappedInteractionControls.innerHTML;

ko.applyBindings(self, interactionControls);

});

return interactionControls;

};

* In the getInteractionControls method of the Experiencestream, query the orchestrator’s getInteractionControls method and return the interaction controls ui element to be added to the player footer. The orchestrator’s getInteractionControls method takes two arguments; first argument is an array of the interaction control unique IDs, the same id with which the control was registered to the InteractionControlFactory in the step above, second argument is a callback function from the InteractionControl, after loading its html. Inside the callback function, you can bind or hookup to the UI events and implement the required functionality.

# Enabling Authoring Experience in RIN Studio

<TBD>

# Building theme support for your Experience

<TBD with folder structure to deploy theme files>

# Deploying your Experience Stream

<TBD>

# Communication between experience streams

<TBD>