iab. TECH LAB

Secure HTML Ad Rich-media Container (SHARC)

First Draft

Released for public comment April 28, 2022

Please email support@iabtechlab.com with feedback or questions by **May 27, 2022**.
This document is available online at https://iabtechlab.com/sharc

Overview

Secure HTML Ad Richmedia Container (SHARC) is a secure container API for managed communication between an app or webpage and a served ad creative.

Change Log

Version	Summary
1st draft	Initial release to public comment

Audience

Developers on the sell side for content platforms will need the details in this document for implementing SHARC on their systems. On the buy side, creative developers will need this document to develop display ads that make use of the SHARC APIs.

About IAB Tech Lab

The IAB Technology Laboratory (Tech Lab) is a non-profit consortium that engages a member community globally to develop foundational technology and standards that enable growth and trust in the digital media ecosystem. Comprised of digital publishers, ad technology firms, agencies, marketers, and other member companies, IAB Tech Lab focuses on improving the digital advertising supply chain, measurement, and consumer experiences, while promoting responsible use of data. Its work includes the OpenRTB real-time bidding protocol, ads.txt anti-fraud specification, Open Measurement SDK for viewability and verification, VAST video specification, and DigiTrust identity service. Board members include ExtremeReach, Facebook, Google, GroupM, Hearst Digital Media, Index Exchange, Integral Ad Science, LinkedIn, LiveRamp, MediaMath, Microsoft, Oracle Data Cloud, Pandora, PubMatic, Quantcast, Rakuten Marketing, Telaria, The Trade Desk, Verizon Media Group, Xandr, and Yahoo! Japan. Established in 2014, the IAB Tech Lab is headquartered in New York City with staff in San Francisco, Seattle, and London. Learn more at https://www.iabtechlab.com.

This document has been developed by the Secure Ad Container Working Group.

IAB Tech Lab Lead:
Katie Stroud, Sr. Product Manager, Ad Experiences

Special Thanks to:

Co-chairs on this project

- Jeffrey Carlson, Chartboost
- Aron Schatz, DoubleVerify

Other key contributors

- Kyle Grymonprez, Twitter
- Marian Rusnak, Verizon
- Bichen Wang, Chartboost
- Laura Evans, Flashtalking by Media Ocean
- Sarah Kirtcheff, Flashtalking by Media Ocean

TABLE OF CONTENTS

Introduction

Secure HTML Ad Richmedia Container (SHARC) is a secure container API for managed communication between an app or webpage and a served ad creative.

SHARC is built on the same premise as two of IAB Tech Lab's ad container standards: SafeFrame and Mobile Rich Ad Interface Definition (MRAID). SafeFrame was designed to run in-web and MRAID was designed to run in a webview in mobile in-app devices. The trouble with these two standards is that they're both very similar and yet different enough that you would have to build two different ad creatives to run a campaign across both web and mobile.

Several ad platforms have tried to build a bridge between the two APIs so that an MRAID ad could also run in a SafeFrame container and a SafeFrame ad could run in an MRAID container. Unfortunately, the differences are stark enough that these attempts at cross-compatibility never really worked out.

The Safe Ad Container working group for ad experiences at IAB Tech Lab have started from the ground up to build a standard for managing rich interactive display ads. Our motto for SHARC is:

Build one ad; serve it everywhere.

With SHARC, a creative developer can build one ad with all the available API functions and serve it to any connected display platform that has implemented SHARC. This is not just limited to web or mobile in-app, it includes a variety of platforms (such as CTV) that are available today and future platforms.

Guiding principles

- Performance
- Industry standards interoperability
- Consumer protection
- Publisher safety and security
- Low barrier of entry (simplicity and ubiquity)
- Minimize impact on key stakeholders in the supply chain (example: OMID included JS libraries to reduce customization that impacted efficiency)
- No ambiguity (detailed specifics in both spec and implementation guide) - while also not delaying release for the sake of clarifying, sub-groups to focus on blocking issues and defying a process to get things done
- Extensibility (helps enable testing of new features before implementing)
- Graceful degradation

Scope

SHARC is intended for managing rich media ad interactions in display placements. While video can be included in the final creative, SHARC provides no playback controls or tracking. SHARC ads can also be served into video players that have implemented SHARC and may be a great way to handle non-linear and companion ads in video ad placements, but this spec does not yet cover that use case.

Out of Scope

The following ad tech operations are out of scope in SHARC:

- Ad request
- Ad delivery
- Measurement
- Ad tracking and reports

While the above operations are out of scope for SHARC, they play a role in the success of SHARC and certain SHARC functions either use or support these operations.

For example IAB Tech Lab's Advertising Common Object Model (AdCOM) is a standardized data structure for relaying details about the placement, the creative, the context, and any other information that all parties in the supply chain need for placing, tracking, and reporting on the ad exchanges in the campaigns they run. It is a dataspec used in the ad request and response, and SHARC requires data from the same dataspec to communicate some of these unchanging details as part of the initiation cycle. AdCOM is the default and preferred dataspec to use, but SHARC itself doesn't supply any of this data; it only provides additional data expected to change during runtime, such as the current state of the container, size changes, or volume details.

The example above explains how other ad operations beyond loading and managing interactions are left to other standards, thereby simplifying SHARC as much as possible. This separation helps SHARC meet some of its guiding principles such as performance and interoperability.

Goals

Write one ad; serve it anywhere.

This is the key goal of SHARC. In order to achieve this goal, we must achieve certain supporting goals to integrate SHARC into systems and operations that make up the digital advertising supply chain.

Adoption is dependent on the following:

- Producing a clear and unambiguous spec for SHARC implementers (this document)
- Providing guidance for different operational audiences targeted to their specific needs for making use of SHARC
- Developing reference code, tools, and examples that simplify implementation
- Creating an awareness of the challenges that SHARC solves in the marketplace for display advertising, especially where no solution currently exists
- Educating different audiences on the benefits and use of SHARC
- Regular updates to support growing market needs

If you would like to get involved, please reach out to support@iabtechlab.com and we'll set you up. You can also visit our GitHub repository at https://github.com/IABTechLab/SHARC.

How it works

SHARC is a protocol for managing ad interactions in a secure container that prevents an ad from accessing data on the platform where the ad displays. In the most simplistic overview of how it works, the steps are as follows:

- [pre-SHARC] an ad is matched and delivered to the SHARC placement
- SHARC initiates. In this step, the following occurs:
    - The SHARC-enabled platform creates the secure (IE: an iframe on web, webview on mobile) container
    - The SHARC container inserts the creative markup and the creative prepares its resources
    - Once in a state to receive SHARC information, the creative informs the container that it is ready to receive initialization information .
    - The SHARC container initializes and provides the creative with data about the container.
    - Data about the environment (placement) and the creative is pulled from the dataspec (default is AdCOM) along with any runtime details such as current size and state and volume settings
    - Once the creative and the container are ready, SHARC asks the creative to start and waits for the creative to respond
- Creative responds with "resolve" indicating that it is ready
- Creative executes, using SHARC functions to resize, navigate away from platform, close, etc.
- Upon completion of the ad experience, SHARC signals a close function and unloads the ad.

A diagram and more detailed descriptions of different use cases are provided in the section on Common Workflows.

The relationship between SIMID and SHARC

To develop SHARC, we looked to the structure of SIMID as a model. SIMID is IAB Tech Lab's Secure Interactive Media Interface Definition. Like SHARC, it uses a secure container to manage an ad experience, except that SIMID functions in the context of a media player. Discussion on whether SIMID should be extended to also handle display ads across platforms or to develop a new standard (SHARC) to handle display ads separately from the SIMID video standard was explored. The decision to create a separate standard emerged as part of the following logic.

As API specs and standards at IAB Tech Lab evolve, there is an opportunity to develop new APIs with shared design principles of existing APIs. Doing so creates a firm foundation upon which new APIs can be built more rapidly while lowering the learning curve for the industry to adopt new specs.

SHARC has opted to share the same messaging protocol and API structure that SIMID developed in order to take full advantage of this opportunity. Sharing this core messaging structure has enabled SHARC to more rapidly prototype a spec tasked with being the cross-platform rich media successor to the Safe Frame and MRAID specs.

In adopting a lot of the API design, features and functionality, a common question asked is why shouldn't SIMID solve all use cases?

The three main reasons are specialization, flexibility and simplicity.

1. For specialization, SIMID was created exclusively to provide rich interactivity for streaming audio and video ads. Expanding its scope beyond its intended use case is exactly how its predecessor VPAID got into trouble. SHARC being separated as a rich media container resolves any issues with SIMID becoming overloaded.
2. For flexibility, it allows both SIMID and SHARC to develop separately as market innovations, needs and problems to be solved potentially fork former shared priorities.
3. For simplicity, SIMID and SHARC can keep a focused API spec relevant to each solution. SHARC has no parallel use case for certain video functions, for example. These improvements can be updated independently without forcing unnecessary version updates on technology stacks.

It is important to have different tools for different use cases. Use SIMID when working with VAST audio or video creative that need interactivity. Use SHARC when working with next-gen rich media display HTML5 creative in web and other platforms. A use case for both specs could be a VAST creative with interactivity and a companion ad as an end card. SIMID would be used to overlay the video and SHARC would be used to display the companion end card.

Secure By Default

One of the main tenets of SHARC is the focus on providing a robust and secure communication and security framework for rich media ad experiences. The end result is that the container performs almost all the functions needed for interacting with the greater publisher content (a web page or an application). The creative must request for actions to be done on the container and the container will either resolve or reject those requests. This puts the container in control and allows for publishers to enable their expected consumer experience without an ad taking over their content. There are common uses cases covered to allow for the wide range of ad experience, but this standard ensures that an ad cannot present a poor consumer experience without the consent of the container.

API Reference

SHARC is a set of messages and data structures that ad-rendering parties exchange using a messaging protocol.

Reference Table: Container

API	resolve	reject
SHARC:Container:init	resolve	reject
SHARC:Container:startCreative	resolve	reject
SHARC:Container:stateChange	n/a	n/a
SHARC:Container:placementChange	n/a	n/a
SHARC:Container:log	n/a	n/a
SHARC:Container:fatalError	resolve	n/a
SHARC:Container:close	resolve	n/a

Reference Table: Creative

API	resolve	reject

SHARC:Creative:fatalError	n/a	n/a
SHARC:Creative:getContainerState	resolve	n/a
SHARC:Creative:getPlacementOptions	resolve	n/a
SHARC:Creative:log	n/a	n/a
SHARC:Creative:reportInteraction	resolve	n/a
SHARC:Creative:requestNavigation	n/a	n/a
SHARC:Creative:requestPlacementChange	resolve	n/a
SHARC:Creative:requestClose	resolve	n/a

Messages from the Container

SHARC specifies a group of messages that enables the container to transmit data, instructions, or state changes to the creative. The container prepends such message types with the `SHARC:Container` namespace.

`SHARC:Container` messages do not communicate ad creative states; SHARC dedicates Messages Triggered by Creative Events to report creative status. A private message bus can be created to inform internal systems on messages sent by the creative, or a public message bus can be created to share these messages to other systems, such as measurement providers.

While some `SHARC:Container` messages expect `resolve` and/or `reject` creative responses, other messages do not require replies.

SHARC:Container:init

The purpose of the `SHARC:Container:init` message is to relay information to the creative and prepare for the creative to start the SHARC ad experience. See Typical Initialization WorkFlow.

The creative must respond to `Container:init` with either resolve or reject.

```
dictionary MessageArgs {
  required EnvironmentData environmentData;
  Supports supportedFeatures;
  Extensions supportedExtensions;
```

```
};
```

environmentData,
Information about publisher's environment and container capacities upon initialization.

supportedFeatures,
Information about SHARC features supported that are beyond basic functionality.

supportedExtensions,
Information about any extensions supported.

```
dictionary EnvironmentData {
  required Placement currentPlacement;
  required Dataspec dataspec;
  required Data data;
  Required enum currentState;
  required string version;
  boolean muted;
  float volume;
};
```

currentPlacement,
Information about the container's current placement properties such as dimensions, location, inline or over content, etc.

dataspec,
The name and version of the dataspec that provides placement and creative information. Default dataspec is AdCOM.

data,
The data provided by the dataspec identified.

currentState,

The current state of the container: ready, active, passive, hidden, frozen, closing, unloaded. See table for descriptions under SHARC:Container:stateChange.

version,
The full version number of the SHARC implementation.

muted,
True if known and device is muted.

volume,
If known, the volume level of the device, expressed as a number between 0 and 1.0.

```
dictionary Placement {
  required Dimensions defaultDimensions;
  boolean inline;
  enum standardSize;
  enum extendDirection;
  boolean push;
  boolean sticky;
};
```

defaultDimensions,
The standard dimensions and coordinates of the container.

inline,
True if the container is anchored within the content of the platform. False if the container is placed over the content.

standardSize,
Indicates whether the current dimensions are one of a standard size: default, max, min.

- default: the initial size of the container
- max: the standard maximum size the container allows. Maximum size may or may not be the full view available to the container but is the max size allowed.

- min: the minimum standard size the container offers.

extendDirection,
Indicates the direction in which the container can extend when resized to larger dimensions: all, up, down, left, right.

push,
True if container resize pushes content in direction of resize.

sticky,
True if container is “sticky,” meaning that it stays in place while content is scrolling up until a given threshold.

```
dictionary Dimensions {
  required long x;
  required long y;
  required long width;
  required long height;
  enum anchor;
 };
```

x,
The x coordinate of the container anchor point.

y,
The x coordinate of the container anchor point.

width,
The width of the container in pixels.

height,
The height of the container in pixels.

anchor,

The anchor corner of the container: top-left, top-right, bottom-left, bottom-right. Default is top-left.

```
dictionary Dataspec {
  required string model;
  required string ver;
  };
```

model,
The data structure model used to provide data. Default is adCOM.

ver,
The version of the data model identified above. Default is “1.0”.

```
dictionary Data {
  // Defined by the dataspec
  };
```

AdCOM example

```
dictionary Data {
  required AdcomAd ad;
  required AdcomPlacement placement
  required AdcomContext context
  };
```

data,
The data provided by the dataspec identified. Recommended AdCOM nodes:

- ad (see AdCOM Ad Object)

- placement (see AdCOM Placement Object)
- context (see AdCOM Context Object)

```
dictionary Supports {
  Sizes supportedSizing;
  Navigation containerNavigation;
  boolean closeSequence;
  };
```

supportedSizing,
Information about container sizes supported.

containerNavigation,
Information about how the container handles navigation. The container always handles navigation, except in situations where it's not technically possible, and the creative must always request navigation so that the container can log the instance.

closeSequence,
True if the container allows the creative to run a 2-second close sequence before the container unloads.

```
dictionary Sizes {
  required Dimensions defaultSize;
  Dimensions maxSize;
  Dimensions minSize;
  };
```

defaultSize,

The container's default dimensions and coordinates.

maxSize,
The maximum dimensions the container allows. If not supported, leave blank.

maxSize,
The minimum dimensions the container offers. If not supported, leave blank.

```
dictionary Navigation {
  boolean navigationPossible;
  boolean navigationAllowed;
  };
```

navigationPossible,
True if the platform in which the container operates supports navigation away from the ad experience and can be handled by the container. If false, navigation away from the ad experience must be handled by the creative (if possible); however, the creative must still always request navigation so that the container can log the request.

navigationAllowed,
True if navigationPossible=true and container allows navigation away from the ad experience.

```
dictionary Extensions {
  array supportedExtensions;
  };
```

`supportedExtensions`,
An array of extensions that the container supports. Each extension must include parameters for `labelName` and `version`.

resolve

The creative acknowledges the initialization parameters.

If the creative delays calling resolve, see Creative Delays Resolving Init

reject

The creative may respond with a reject based on its internal logic.

```
dictionary MessageArgs
{
  required unsigned short errorCode;
  DOMString reason;
};
```

`errorCode`,
See Error Codes.

`reason`,
Optional information about cause of rejection.

The container then will follow the rejection workflow. See Creative Rejects Init.

SHARC:Container:startCreative

See Typical Initialization Workflow

The container posts `SHARC:Container:startCreative` message when it is ready to make the

iframe visible. The container waits for a resolve response to display itself. The interactive creative should be ready to reply to `Container:startCreative` immediately.

SHARC:Container:init section describes the flow that precedes the instant the container emits a `Container:startCreative` message.

resolve

By posting `resolve`, the interactive creative acknowledges that it is ready for display. The creative should be ready to respond immediately. The container makes itself visible upon a resolve receipt

Refer to Typical Initialization WorkFlow.

reject

When the creative responds with a reject, the container may unload the ad. The player reports error tracker with the `errorCode` the creative supplied.

```
dictionary MessageArgs{
  required unsigned short errorCode;
  DOMString reason;
};
```

errorCode,
See Error Codes.

reason,
Additional information.

SHARC:Container:stateChange

The container posts a SHARC:Container:stateChange message whenever the container state is changed. Certain container or environment events can trigger a state change. For example, Container:init triggers the “ready” state. Or a change in focus, such as when a user switches tabs in a browser, can change the state from “active” or “passive” to “hidden.” The new container state is reported with the message.

```
dictionary MessageArgs{
  DOMString containerState;
  };
```

containerState,

The current (new) container state, which is one of: ready, active, passive, hidden, frozen, closing, unloaded. See reference chart below for definitions of states.

Table of possible container states

State	Description
ready	The container has successfully completed initialization (Container:init) and is ready for the creative to start. **Possible previous states:** (none) **Possible next states:** active
active	Container is currently in a space that is visible and in use (has focus and input) **Possible previous states:** Ready (Container:init) Passive **Possible next states:** Passive Closing
passive	Container is currently in a space that is visible but no longer in use (has focus but no input). **Possible previous states:** active **Possible next states:** active hidden

hidden	Container is no longer visible or in use, but certain tasks may still be running and the container has not yet been closed or unloaded **Possible previous states:** passive **Possible next states:** passive frozen unloaded
frozen	Container function is unavailable while container waits for something such as a response to a certain function **Possible previous states:** hidden passive **Possible next states:** active closing
closing	The close sequence has been initiated and the container is in a state of closing. **Possible previous states:** Frozen Active **Possible next states:** Unloaded
unloaded	The container has unloaded and can no longer function. **Possible previous states:** Closing **Possible next states:** (none)

The creative can request the current state of the container any time using Creative:getContainerState.

SHARC:Container:placementChange

When the container changes its properties, such as dimensions and location (usually in

response to a request by the creative), it posts the `SHARC:Container:placementChange` message. The message describes the container dimensions and coordinates.

```
dictionary MessageArgs{
  required Placement placementUpdate;
  };
```

placementUpdate,
Information about changes in the container properties, such as dimensions and location.

```
dictionary Placement {
  Dimensions containerDimensions;
  boolean inline;
  enum standardSize;
  enum extendDirection;
  boolean push;
  boolean sticky;
};
```

containerDimensions,
The standard dimensions and coordinates of the container.

inline,
True if the container is anchored within the content of the platform. False if the container is placed over the content.

standardSize,
Indicates whether the current dimensions are one of a standard size: default, max, min.

- default: the initial size of the container
- max: the standard maximum size the container allows. Maximum size may or may not be the full view available to the container but is the max size allowed.

- min: the minimum standard size the container offers.

extendDirection,
Indicates the direction in which the container can extend when resized to larger dimensions: all, up, down, left, right.

push,
True if container resize pushes content in direction of resize.

sticky,
True if container is “sticky,” meaning that it stays anchored in place while content is scrolling up until a given threshold.

```
dictionary Dimensions {
  required long x;
  required long y;
  required long width;
  required long height;
  enum anchor;
 };
```

x,
The x coordinate of the container anchor point.

y,
The x coordinate of the container anchor point.

width,
The width of the container in density-independent pixels.

height,
The height of the container in density-independent pixels.

anchor,

The anchor corner of the container: top-left, top-right, bottom-left, bottom-right. Default is top-left.

See SHARC:Creative:requestPlacementChange

SHARC:Container:log

The purpose of the Container:log message is to convey optional, primarily debugging, information to the creative.

Note: In SHARC prefixing log messages with “WARNING:” has a specific meaning. The container is communicating performance inefficiencies or specification deviations aimed at creative developers. For example, if the creative sends the requestPlacementChange message but does not use the correct parameters (dimensions and coordinates), a “WARNING:” message is appropriate.

```
dictionary MessageArgs{
  required DOMString message;
};
```

message,
Logging information.

SHARC:Container:fatalError

The container posts a `SHARC:Container:fatalError` message when it encounters exceptions that disable any further function. If feasible, the container waits for `resolve` response from creative before unloading.

See Container errors out

```
dictionary MessageArgs{
  required unsigned short errorCode;
  DOMString errorMessage;
};
```

errorCode,
See Error Codes

errorMessage,
Additional information

resolve

The creative must respond to `Container:fatalError` with `resolve`. After `resolve` arrives, the container unloads.

See Creative Errors Out

SHARC:Container:close

The container provides a close control and handles the `Container:close` and subsequent `Container:unload` events. If supported, the container may allow the creative to run a close sequence that is no more than 2 seconds long.

The container issues Container:close when:

- The user activates the close control
- The creative requests close with Creative:requestClose

- Something in the content platform requires the container to close

resolve

The creative responds with resolve to acknowledge that the container is going to close. The container may proceed to unload with or without creative response. If supported, the container may wait for up to 2 seconds to allow the creative to run a close sequence.

Messages from the Creative to the Container

The creative posts messages to the container to request container state changes, obtain data, and to send notifications. The creative prefixes its messages with the namespace `SHARC:Creative`.

`SHARC:Creative` messages may require the container to accept and process arguments. With some messages, the creative expects the container to respond with resolutions.

SHARC:Creative:fatalError

The creative posts `SHARC:Creative:fatalError` in cases when its internal exceptions prevent the interactive component from further execution. In response to the `Creative:fatalError` message, the container unloads the SHARC iframe and reports the `errorCode` specified by the creative.

```
dictionary MessageArgs{
  required unsigned short errorCode;
  DOMString errorMessage;
};
```

errorCode,
See Error Codes.

errorMessage,
Additional information.

SHARC:Creative:getContainerState

The creative posts a `SHARC:Creative:getContainerState` message to request the current container state.

resolve
The container should always respond with `resolve`.

```
dictionary MessageArgs{
  enum currentState;
  };
```

currentState,
The current container state, which is one of: ready, active, passive, hidden, frozen, closing, unloaded. See Table of Possible Container States for definitions of states.

SHARC:Creative:getPlacementOptions

The creative posts a `SHARC:Creative:getPlacementOptions` message to request information about placement options.

resolve
The container should always respond with `resolve`, including in situations when the container is unable to provide all expected values.

```
dictionary MessageArgs{
  required Placement currentPlacementOptions;
  };
```

currentPlacementOptions,
Information about current container properties, such as dimensions and location.

```
dictionary Placement {
  Dimensions containerDimensions;
  boolean inline;
  enum standardSize;
  enum extendDirection;
  boolean push;
  boolean sticky;
};
```

containerDimensions,
The standard dimensions and coordinates of the container.

inline,
True if the container is anchored within the content of the platform. False if the container is placed over the content.

standardSize,
Indicates whether the current dimensions are one of a standard size: default, max, min.

- default: the initial size of the container
- max: the standard maximum size the container allows. Maximum size may or may not be the full view available to the container but is the max size allowed.
- min: the minimum standard size the container offers.

extendDirection,
Indicates the direction in which the container can extend when resized to larger dimensions: all, up, down, left, right.

push,
True if container resize pushes content in direction of resize.

sticky,
True if container is "sticky," meaning that it stays anchored in place while content is scrolling up until a given threshold.

SHARC:Creative:log

The message `SHARC:Creative:log` enables the creative to communicate arbitrary information to the player.

> **Note:** If the `log` message purpose is to notify the container about the container's non-standard behavior, the creative prepends `Message.args.message` with "WARNING:" in the `string`. Warning messages are used to inform container developers about occurences of non-fatal issues.

```
dictionary MessageArgs{
  required DOMString message;
};
```

`message`,
Logging information.

SHARC:Creative:reportInteraction

The `SHARC:Creative:reportInteraction` message enables a creative to delegate arbitrary interaction metrics to the container.

These interaction metrics are URIs into which the creative may inject macros.

In response to the `reportInteraction` message, the container must:

- Send the trackers specified by the message as soon as possible.
- Replace any macros in the dataspec with the corresponding values.
- Accept and send the trackers with custom macros – leave non-standard macros intact unless the publisher-ad integration involves custom macros processing.

```
dictionary MessageArgs{
  required Array trackingUris;
};
```

trackingUris,
Array of URIs.

resolve
The player posts a `resolve` after it sends the trackers.

SHARC:Creative:requestNavigation

The creative posts the `SHARC:Creative:requestNavigation` message when an interaction or some other event has triggered navigating to the creative's clickthrough URI.

The container handles all navigation in situations where the function is available to the container. In some situations, such as in web, navigation is handled by the browser. However, even when the container cannot handle navigation to the creative's link, the creative must always request navigation so that the container is aware.

Navigation capabilities are provided upon initiation. See SHARC:Container:init.supports.navigation for details.

SHARC:Creative:requestPlacementChange

The creative posts the SHARC:Creative:requestPlacementChange message when the creative would like the container to modify its properties, such as size.

Requesting a placement change is a robust way to request a resize. Along with resize, the creative can ask the container to expand in a specified direction or change the "stickiness" of the container.

resolve
The container should always respond with `resolve`, including in situations when the container is unable to provide all expected values. The container should also post the

SHARC:Container:placementChange message with updates to any changes in properties to the container.

```
dictionary MessageArgs{
  required Placement changePlacement;
};
```

changePlacement,
Information about what container properties the creative would like to change.

```
dictionary Placement {
  Dimensions containerDimensions;
  boolean inline;
  enum standardSize;
  enum extendDirection;
  boolean push;
  boolean sticky;
};
```

containerDimensions,
The standard dimensions and coordinates of the container.

inline,
True if the container is anchored within the content of the platform. False if the container is placed over the content.

standardSize,
Indicates whether the current dimensions are one of a standard size: default, max, min.

- default: the initial size of the container
- max: the standard maximum size the container allows. Maximum size may or may not be the full view available to the container but is the max size allowed.
- min: the minimum standard size the container offers.

extendDirection,

Indicates the direction in which the container can extend when resized to larger dimensions: all, up, down, left, right.

`push,`
True if container resize pushes content in direction of resize.

`sticky,`
True if container is "sticky," meaning that it stays anchored in place while content is scrolling up until a given threshold.

SHARC:Creative:requestClose

The container ALWAYS handles closing the container, including providing the close function. However, if the creative has a reason to close the container before the container's close control is activated, the creative can post the ShARC:Creative:requestClose message to ask the container to close.

resolve
If the container can close, it responds with a `resolve`.

reject
If the container cannot close, it responds with a `reject`.

With the requestClose rejection:

- The container maintains its current state.
- The container continues posting messages as appropriate.
- The creative may unload and send a Creative:log message to report that it has unloaded.

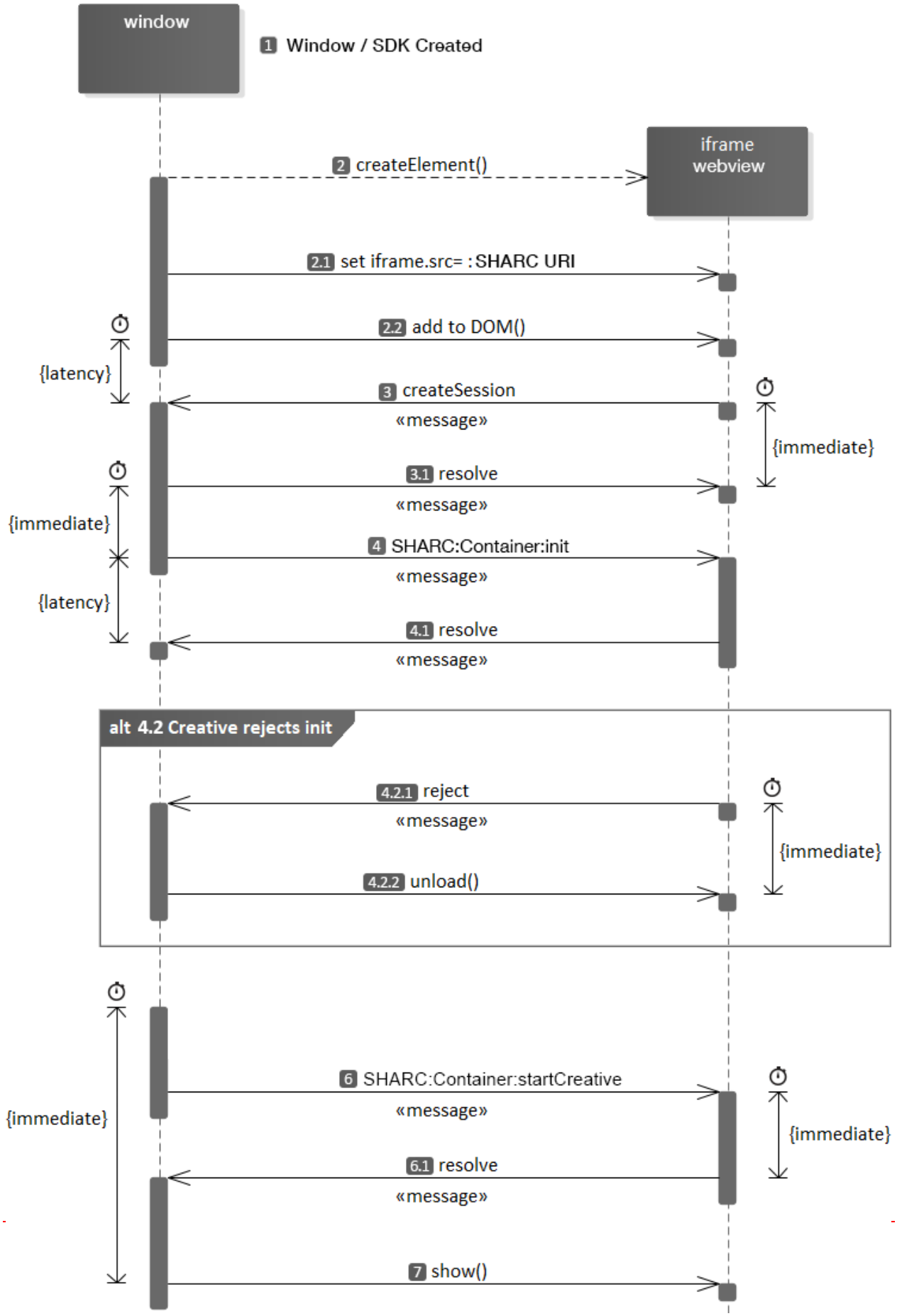
Extensions

SHARC cannot account for all possible use cases. In these circumstances, SHARC implementers may include one or more extensions. Any extensions can be provided in the Extensions node in init.

Common Workflows

Loading (Ad Lifecycle)

(creating container, preparing to execute ad)
Define the end-to-end lifecycle and break down by states


window
1 Window / SDK Created
iframe
webview
2 createElement()
2.1 set iframe.src= : SHARC URI
2.2 add to DOM()
{latency}
3 createSession
«message»
{immediate}
3.1 resolve
«message»
{immediate}
4 SHARC:Container:init
«message»
{latency}
4.1 resolve
«message»
alt 4.2 Creative rejects init
4.2.1 reject
«message»
{immediate}
4.2.2 unload()
{immediate}
6 SHARC:Container:startCreative
«message»
{immediate}
6.1 resolve
«message»
7 show()

Typical Initialization Workflow

- Create Session happens
    - Creative asks for session
    - Resolve: Container creates session and sends info about what's next (container set up)
    - Reject: If session not created, reject includes details about why
- Container set up
    - Container sets up container
        - Sends details about environment, settings, etc. to creative
        - Use case: mute button settings defined, creative sets creative with defined settings, tells container that it's ready
    - Creative receives container details and responds (resolve/reject) with ready details
        - Resolve: includes setting details
        - Reject: can't get into ready state; go ahead and try to load something else (include some specific use cases for when this happens, like connection cut. Can't send reject just because you want to; only in case where it's impossible. Rejecting just because you want to causes publisher to lose opportunity)
- Start Creative (time to play the ad)
    - Getting to this point means that both container/creative have said they're ready to go and everything is in place.

Non-SHARC Creatives

SHARC as a container standard expects SHARC enabled creatives. However, as part of this standard, a SHARC enabled container must be able to render a standard HTML type of creative within its secure container, if possible, based on the container runtime environment. While the ad experience will not be as robust as a SHARC enabled experience, the publisher content will be protected by wrapping the creative in a secure container. However, SHARC dictates an initialization and start workflow for creatives. In cases where the creative won't respond to the container, the container may assume that the creative is malfunctioning unless the creative gives a hint to the container that it isn't SHARC enabled.

How to Handle Close Sequence

The container always handles close, but may allow for the creative to run a brief close (max 2 seconds) sequence upon initiating close. Upon close, the container may return to a default size and position, minimize the container, or unload the creative and container.

- Upon init, container establishes whether close sequence is allowed and whether close unloads the ad or returns to a default or minimum size.
- User initiates close using container-provided close feature.
- Container reports that close has been initiated.
- If container allows close sequence, creative runs close sequence and reports SHARC:Creative:closeSequenceCompleted.
- Container executes close. If container doesn't report closeSequenceCompleted within 2 seconds, container proceeds with close.
- If container returns to default or minimum size, container reports result.
- If container unloads in response to close, there is nothing to report.

Note: Each SHARC instance only ever contains one ad. If container wants to replace closed ad with new ad, it must unload existing instance and replace with new instance and new ad. If reloading the same ad, it's still a new instance.

How to Handle Navigation Event

To-do

How to Handle Interactions

To-do

How to Handle Ad End and Unload

The following events trigger an ad end:

- User activates close control
- The environment in which the container is deployed (app, browser, etc.) has been shut down
- The container encounters a fatal error

Creative Delays Resolving Init

To-do

Creative Rejects Init

To-do

Error Handling and Timeouts

If the creative cannot be executed the container should terminate the ad and fire an error.

If either the creative or container wants to terminate with an error the player should fire a 902 error. The creative or container should pass a specific error code to indicate why it errored out. The creative can also hand back a string with extra details about the error.

Error Codes

Code	Error	Description
2100	Unspecified creative error	Catchall error when no existing code matches the error. Creative errors should be as specific as possible.
2101	Resources could not be loaded	The SHARC creative tried to load resources but failed.
2102	Container dimensions not suited to creative	The container dimensions provided were unmatched to the dimensions the creative specified.
2103	Wrong SHARC version	The creative could not support the container's version of SHARC.
2104	Creative could not be executed	For an unspecified technical reason, the creative could not be executed.
2105	Resize request not honored	The container rejected the creative's resize request.
2106	Pause?	
2107	Player controls?	
2108	Ad internal error	The creative had an error not related to any external dependencies.
2109	Device not supported	The creative could not render or execute on the device.
2110	Container not sending	

	messages as specified	
2111	Container not responding adequately to messages	
2200	Unspecified container error	Catchall error when no existing code matches the error. Container errors should be as specific as possible.
2201	Wrong SHARC version	The container could not support the creative's version of SHARC.
2202	Too much time requested?	
2203	SHARC creative requesting more functionality than container willing to support	
2204	SHARC creative executing actions not supported	
2205	SHARC creative is overloading the postmessage channel	
2206	Media could not be loaded?	
2207	Media timeout?	
2208	SHARC creative taking too long to resolve or reject message(s)	
2209	SHARC creative provided is not supported on this device	
2210		

Error CodeError TypeDescription1100Unspecified error.Catchall error if the creative could not find a matching error code. The creative should be more specific in the error message.1101Resources could not be loaded.The SIMID creative tried to load resources but failed.1102Playback area not usable by creative.The dimensions the creative needed were not

what it received.1103Wrong SIMID version.The creative could not support the players version.1104Creative not playable for a technical reason on this site.1105Request for expand not honored.The creative requested to expand but the player did not allow it.1106Request for pause not honored.The creative requested pause but the player did not pause.1107Play mode not adequate for creative.The creative requires playback control but the player is not giving control. This error should only fire if the VAST for the ad specified that it needs playback control.1108Ad internal error.The creative had an error not related to any external dependencies.1109Device not supported.The creative could not play or render on the device.1110The player is not following the spec in the way it sends messages.1111The player is not responding adequately to messages.1200Unspecified error.Catchall error if the player could not find a matching error code. The player should be more specific in the error message.1201Wrong SIMID version.The player could not support the creatives version.1202SIMID creative requesting more time than the player is willing to support.1203SIMID creative requesting more functionality than the player is willing to support.1204SIMID creative is doing actions not supported on this site.1205SIMID creative is overloading the postmessage channel.1206The SIMID media could not be loaded.1207Media Timeout.The ad media creative buffered for too long and timed out.1208The SIMID creative is taking too long to resolve or reject messages.1209The SIMID creatives media from the VAST response is not supported on this device.1210The SIMID creative is not following the spec when initializing.1211The SIMID creative is not following the spec in the way it sends messages.1212The SIMID creative did not reply to the initialization message.1213The SIMID creative did not reply to the start message.

Container Times Out

Creative Times Out

Messaging Protocol

In SHARC, the media container and the creative overlay communicate by exchanging asynchronous signals that maintain a custom messaging protocol.

This protocol governs:

- Data Layer
- Transport Layer
- Session Layer

Data Layer

SHARC messages transport data. In HTML environments, the data is the `message` argument of the `Window.postMessage()` function.

Data Structure

The `message` data implements the following data structure for an HTML environment:

```
dictionary Message{
  required DOMString sessionId;
  required unsigned long messageId;
  required unsigned long timestamp;
  required DOMString type;
  any args;
};
```

`sessionId`,

A string that uniquely identifies the session to which Message belongs. See Session Layer.

`messageId`,

A message sequence number in the sender's system. Each participant establishes its own independent sequence counter for the session. The first message `messageId` value is `0`. The sender increments each subsequent messageId value by `1`. In practice, this means that the creative and the container `messageId` values will be different based on the number of sent messages.

`timestamp`,

A number of milliseconds since January 1, 1970, 00:00:00 UTC (Epoch time). The message sender must set the `timestamp` value as close as possible to the moment the underlying process occurs. However, the receiver should not assume that the `timestamp` value reflects the exact instant the message-triggering event occurred, not necessarily the time of the event.

`type`,

A string that describes the message-underlying event and informs the receiver how to interpret the `args` parameter.

`args`,

Additional information associated with the message `type`.

Example of message data:

```
{
    sessionId: "173378a4-b2e1-11e9-a2a3-2a2ae2dbcce4",
    messageId: 10,
    timestamp: 1564501643047,
    type: "SHARC:Container:adClosed",
    args: {
        code: 0
    }
}
```

Messages Categories

The protocol defines two message classes:

- ***Primary*** messages - the signals triggered by the sender's internal logic.
- ***Response*** messages - the signals the receiver transmits as acknowledgments of the primary message receipt and processing. There are two response Message types: resolve Messages and reject Messages.

Both primary and response messages implement the same data structure (see Data Structure).

`resolve` Messages

The receiver confirms successful message processing by replying with a resolution message.

`Message.type` must be `resolve`.

`Message.args` must be a `ResolveMessageArgs` object:

```
dictionary ResolveMessageArgs{
      required unsigned long messageId;
      any value;
};
```

`messageId`,

The value of the messageId attribute of the message to which the receiver responds.

`value`,

Additional data associated with this `resolve` message.

Example of `resolve` message:

```
{
    sessionId: "173378a4-b2e1-11e9-a2a3-2a2ae2dbcce4",
    messageId: 10,
    timestamp: 1564501643047,
    type: "resolve",
    args: {
        messageId: 5,
        value: {
            id: 45
        }
    }
}
```

`reject` Messages

When the receiver is unable to process the message (or refuses it), it responds with rejection.

`Message.type` must be `reject`.

`Message.args.value` must be a `RejectMessageArgsValue` object:

```
dictionary RejectMessageArgsValue{
   required unsigned long errorCode;
   DOMString message;
};
```

`errorCode`,

The error code associated with the reason the receiver `rejects` the message.

`message`,

Additional information.

Example of `reject` message:

```
{
    sessionId: "173378a4-b2e1-11e9-a2a3-2a2ae2dbcce4",
    messageId: 10,
    timestamp: 1564501643047,
    type: "resolve",
    args: {
        messageId: 5,
        value: {
            errorCode: 902,
            message: "The feature is not available."
        }
    }
}
```

Transport Layer

Transport is a communication mechanism that can send serialized messages between two parties.

`postMessage` Transport

In HTML environments, where the container loads creative overlay in a cross-origin iframe, the parties utilize the standard `Window.postMessage()` API as the message transport mechanism.

Message Serialization

The message sender serializes data into a `JSON` string. The deserialized `JSON` must result in a clone of the original Message data object.

In JavaScript, `JSON.stringify()` performs serialization; `JSON.parse()` - deserialization.

Session Layer

The media container may manage several ads that are in different phases of their lifespans; multiple concurrent sessions may be active. For example, while the container is rendering ad-A,

it preloads and engages ad-B. Simultaneous two-way communication between the container and both ads persists.

Each session has a unique identifier. All messages that belong to a specific session must reference the same session id.

Establishing a New Session

The `createSession` message is the signal from the creative to the SHARC container that the underlying rich media is ready to proceed in the ad lifecycle and ready to send and receive further messages.

SHARC delegates the session initialization to the creative overlay. The creative generates a unique session id and posts the first session message with the `Message.type createSession`. By posting the `createSession`message, the creative acknowledges its readiness to receive messages from the container.

Note: There is no expectation for the interactive component to be entirely able to participate in ad rendering at the time the creative signals `createSession` message. Full creative initialization may occur at later stages when the container provides complete data - see § 4.3.7 SHARC:container:init.

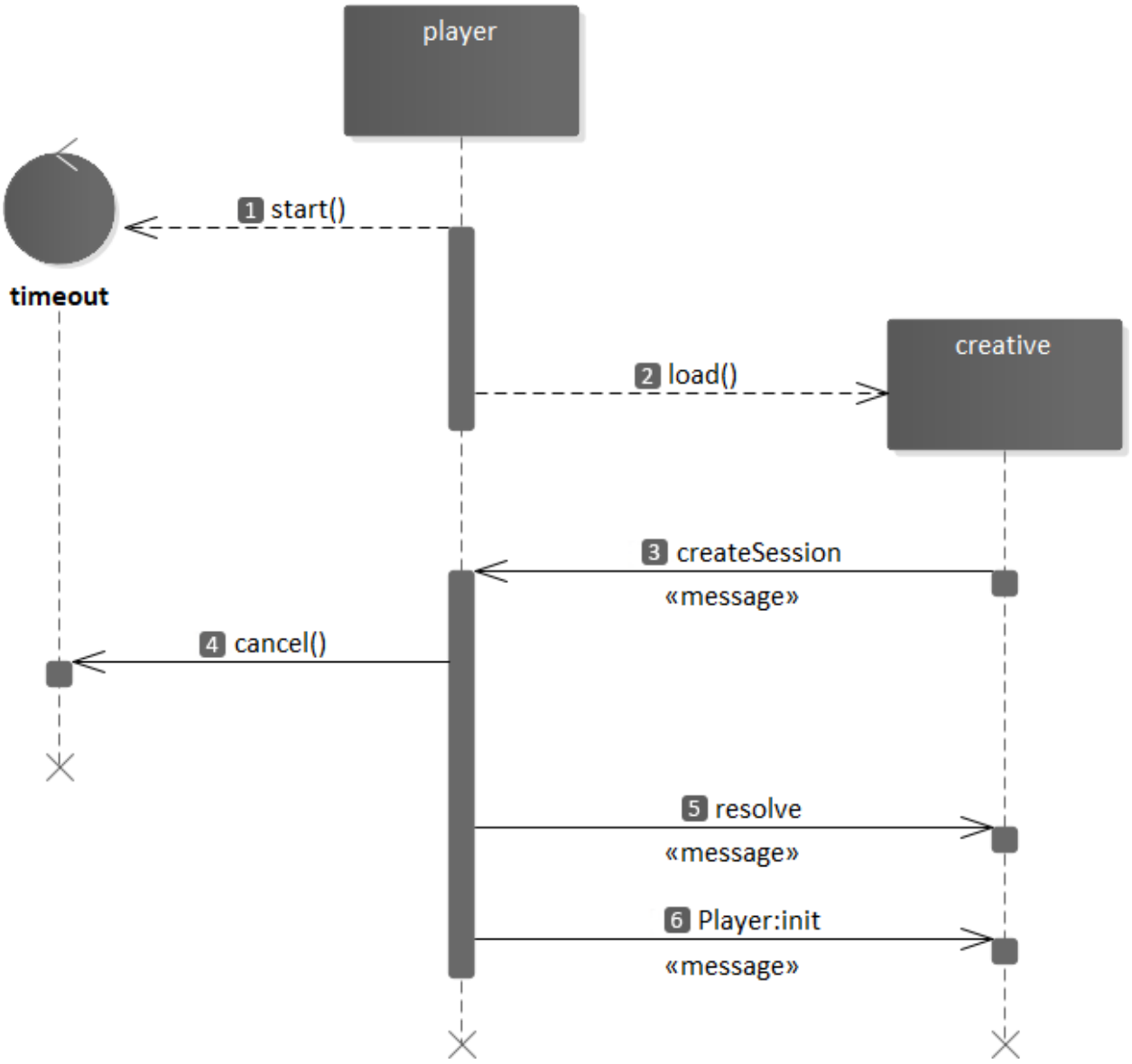
Example of `createSession` Message data:

```
{
    sessionId: "173378a4-b2e1-11e9-a2a3-2a2ae2dbcce4",
    messageId: 0,
    timestamp: 1564501643047,
    type: "createSession",
    args: { }
}
```

Creative should initialize the session as soon as possible. The container should establish a reasonable timeout for the session initialization message receipt.

The container responds to `createSession` with a `resolve` message.

Typical Session Initialization Sequence



1. The container starts a `createSession` message timeout.
2. The container loads creative.
3. Creative posts `createSession` message.
4. The container cancels the timeout.
5. The container responds with a `resolve` message.
6. The container initializes creative. See § 4.3.7 SHARC:container:init.

Session Establishing Delays and Failures

Typically, the container should wait for the creative to post a `createSession` message before proceeding to the simultaneous rendering of both ad media and the interactive component. However, SHARC recognizes scenarios when:

- The creative fails to establish a session within the allotted time.
- The container's environment restricts timeout usage (effectively, the timeout is zero). Specifically, SSAI and live broadcasts force zero-timeout use cases.

The creative's failure to establish a session does not prevent the container from rendering the ad media. If the creative does not post a `createSession` message on time, the container may proceed with the ad media rendering. However, the container allows the creative to recover in the middle of the ad media playback. The container:

- Does not unload the creative.
- Does not post messages to the creative.
- Maintains the `creativeSession` message handler.

If the creative has not established a session before the media playback is complete, the container will report a VAST Error tracker with the proper error code. Examples of situations when this may occur are listed below.

Sequence for a failed session initialization

1. The timeout expires.
2. The `createSession` message does not arrive.
3. The container starts ad media.
4. The container reports the impression.
5. The ad media playback completes.
6. The container reports the VAST error tracker.
7. The container unloads the creative iframe.

Creative posts a `createSession` message after the timeout occurs

1. The timeout expires.
2. The container retains the interactive component.
3. The container initiates ad media playback.
4. The container reports the impression.
5. The container does not post messages to the creative.
6. The creative posts `createSession` message.
7. The container proceeds with the creative initialization.

Compatibility Modes

SHARC does NOT support MRAID or SafeFrame, but for adoption SHARC is working on bridge layers to work with MRAID or SafeFrame.

Compatibility Mode with MRAID

The SHARC working group is working on a compatibility bridge to enable transitioning from MRAID to SHARC.

Compatibility Mode with SafeFrame

The SHARC working group is working on a compatibility bridge to enable transitioning from MRAID to SHARC.