Trinity Audio Player Integration Guide

This document describes how to integrate the Trinity Audio Player into a page as well as how to configure and control it

> Updated: Mar 27, 2023
> Document version: 2.6

Integration

In order to integrate the player into a page, you need to insert a `script` tag into your HTML page.
The player tag should looks like this:

```
<script src="https://trinitymedia.ai/player/trinity/XXXXXXX/?pageURL=YOUR_ENCODED_PAGE_URL_HERE"
        charset="UTF-8"></script>
```

There are a few ways to do that.

HEAD

Include the player `<script>` tag into the **HTML** `<head>` tag.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Trinity Page Example</title>
    <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1" />
    <script src="https://trinitymedia.ai/player/trinity/XXXXXXX/?pageURL=YOUR_ENCODED_PAGE_URL_HERE"
            charset="UTF-8"></script>
  </head>
  <body>
    <div class="content">Hello!</div>
  </body>
</html>
```

In that case, the player will be appended right into the `body` tag. In order to control where exactly it should get rendered, you can create a `div` tag with `trinityAudioPlaceholder` class in the desired place, e.g.

```
<header></header>
<div class="trinityAudioPlaceholder"></div>
<article></article>
<footer></footer>
```

BODY (Dynamic)

Put the player tag wherever you want inside the `body` tag, and it will get rendered there, e.g.

```
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Trinity Page Example</title>
```

```
    <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1" />
  </head>
  <body>
    <div class="content">Hello!</div>
    <script src="https://trinitymedia.ai/player/trinity/XXXXXXX/?pageURL=YOUR_ENCODED_PAGE_URL_HERE"
            charset="UTF-8"></script>
    <article></article>
    <footer></footer>
  </body>
</html>
```

You can use the following script to inject player with correct `pageURL` for you. Just place the script where you expect player to render:

```
<script>
  const scriptEl = document.createElement('script');
  scriptEl.setAttribute('fetchpriority', 'high');
  scriptEl.setAttribute('charset', 'UTF-8');
  scriptEl.src = 'https://trinitymedia.ai/player/trinity/XXXXXXX/?pageURL='
    + encodeURIComponent(window.location.href);
  document.currentScript.parentNode.insertBefore(scriptEl, document.currentScript);
</script>
```

Script Tag Parameters

Trinity Player accepts various query parameters, so you can change its configuration on the fly without changing those settings on the unit level for all pages.

Name	Description	Rewriting unit settings?	Values
XXXXXXX	**Required**. Your unit ID		
pageURL	**Required**. Encoded URL of your page		e.g. https%3A%2F%2F example.com%2Fpage-1%2Farticle-1
language	Text language	+	en, es, it, fr, de, pt
partner	Partner name		
FAB	FAB view. Player becomes small FAB when user scrolls and player disappears from view. Can be only enabled	+	1
abtest	A/B testing for player view. For more information please contact us.	+	
voiceGender	Gender of voice	+	m, f. NOTE: Rewrites gender if set
voiceId	Desired voice ID	+	To get the full list of supported voice ID's visit the dashboard and choose at the Player configuration screen
playbackSpeed	Reading speed	+	0.5 - 2
textSelector	Custom text selector	+	

Name	Description	Rewriting unit settings?	Values
readContentType	URL to read text from instead of the original one, located by provider selector	+	
readContentConfig	JSON config	+	
documentLoadType	Type of document load scenario		DOMContentLoaded, onload, onSelectorVisible, onSelectorExists
documentLoadTypeSelector	CSS selector when `documentLoadType` is `onSelectorVisible` or `onSelectorExists` type		
multipleArticlesAlg	Enable playlist of popular articles from the same domain	+	byContentStarted
adMaxDurationAllowed	Advertisement length limitation in seconds	+	e.g. 30
subscriber	Disable Ads	+	1
publisherUserId	Identifier of your visitor that will be stored in the Trinity player		Any custom identifier, for example a UUID
g_cust_params	additional parameters to pass to Google IMA. Should be URL encoded		param1%3Dabc%26param2%3D123
cms	cms attribute to automatically index the content created into the CMS		cms=%7B%22channelHash%22%3A%22a5gTS... success%22%5D%7D
themeId	Player theme id	+	
publisherSections	Use this field to pass the different section of the content page. This will be used for indexing and reporting.	-	
shareEnabled	Enable/disable share functionality	+	1/0

Just pass the appropriate parameter to Player tag as a query parameter, e.g.

```
<!-- testing FAB functionality -->
<script src="https://trinitymedia.ai/player/trinity/XXXXXXX/?pageURL=YOUR_ENCODED_PAGE_URL_HERE&FAB=1"
        charset="UTF-8"></script>
```

Page Parameters

The Trinity Player supports some functionality enablement by passing certain parameters to the page URL. That comes in handy for testing features without any code modifications. For example, if Trinity Player is disabled for the unit by default, it is possible to pass `?TRINITY_LOAD_PLAYER=1` in the page URL in order to test it. This allows production environment testing, while other users can't see it until you are ready.

Name	Description	Values
TRINITY_FAB	Enable `FAB` functionality	1
TRINITY_MULTIPLE_ARTICLES_ALG	Enable multiple articles	byContentStarted
TRINITY_FAB_ONLY	Enable/disable FAB-only mode for Trinity Player	1/0

Just pass those options as query parameters to your page URL, e.g.

`http://example.com/some-article?TRINITY_LOAD_PLAYER=1`

Will load Trinity Player even if it's disabled.

Document load scenario

> This section should be considered for publishers using dynamic content loading

Some pages have dynamic content loading (AJAX). Since the player loads as soon as the Document DOM is ready, it can turn out that the textual content hasn't rendered yet, and the player will show `00:00` as it has no content to read.
That could be controlled by adjusting the `documentLoadType` parameter.

The default one is `DOMContentLoaded`. As this is the default method, nothing should be passed to the URL parameter. When the DOM is ready the player will read the content using the selector set for the unit.

The `onload` type will wait until the document is fully loaded, e.g. all images, all AJAX requests and more. It could take a while, but could be useful for simple fixing of loading issues, especially if you don't want to use more advanced techniques such as `onSelectorVisible` or `onSelectorExists`.

The `onSelectorVisible` type requires passing the `documentLoadTypeSelector` parameter as well. The `documentLoadTypeSelector` should be passed with a CSS selector, and the player will check if that selector exists and is visible on the page. When the selector is ready, the player will render and read the content using the default selector set for the unit.
That technique is useful if you have dynamic content loading to the page, and using it will make the player display only once its ready.

The `onSelectorExists` type uses the same approach as `onSelectorVisible` but instead checks if the element at the provided CSS selector exists, not if it is visible.

```
<!-- example for checking .content > div (encodeURIComponent(btoa('.content > div'))) -->
<script src="https://trinitymedia.ai/player/trinity/XXXXXXX/?pageURL=YOUR_ENCODED_PAGE_URL_HERE
&documentLoadType=onSelectorVisible&documentLoadTypeSelector=LmNvbnRlbnQgPiBkaXY%3D" charset="UTF-8">
</script>
```

Read Content Hook

In some cases, you can't provide all of the content at once on one page, due to lazy-loading text or other types of pagination. This also pertains to cases where getting the full text from the page could be difficult or impossible due to missed semantic HTML. In cases like these, you can use one of the custom ways we offer to pass the text.

Type	Description
URL	Read the content using the provided URL. All text on the page will be read
URL_BY_PAGE_SELECTOR	Read the content from a provided URL and use a certain CSS selector to filter out the right text
WP_JSON_BY_PAGE_SELECTOR	Read the content from WP-JSON API, prefix URL provided on a page using a certain selector

Examples

Encode pageURL

pageURL query param must be encoded

```
encodeURIComponent('https://example.com/page-1/article-1'); // https%3A%2F%2Fexample.com%2Fpage-1%2Farticle-1
```

Read content from remote URL

In this method, we would like to read the text from a provided URL, for example https://example.com/amp/article-123.
First, prepare the query string passed to the player.

```
encodeURIComponent(JSON.stringify({
    url: 'https://example.com/amp/article-123',
    dataType: 'html' // since our content is HTML,
    /*
      In case using CORS and having page under paywall/login that requires cookies
      in order entire content to be rendered.
      Be sure that your cookies use subdomain (.mydomain.com) if needed and you don't have
      `Access-Control-Allow-Origin: *` in response, since such response is not allowed when cookies is sent.
    */
    // sendCookies: true
})); // %7B%22url%22%3A%22https%3A%2F%2Fexample.com%2Famp%2Farticle-123%22%2C%22dataType%22%3A%22html%22%7D
```

Now, pass two additional parameters to the player:

1. Add the `&readContentType=URL` parameter to our tag, in order to select this method.
2. Pass the hash you received above using the `readContentConfig` parameter:
   `&readContentConfig=%7B%22url%22%3A%22http...`.

Read content from a remote HTML, e.g. AMP page

This method is very similar to what is mentioned above, while the main difference is that we won't read all the text from the URL but we will apply the CSS selector from the unit on it.
This is usually used with AMP integration, due to AMP limitations, as we're using the regular HTML version of the page to get acess to the text.

In the following example, you can see that we have no access to the page content:

```
<html>
    <head>
        <link rel="amphtml" href="https://example.com/amp/article-123">
    </head>
</html>
```

We want to read the text from the link above, e.g. https://example.com/amp/article-123.
In order to do that, let's first prepare the query string that will be passed to the player.

```
encodeURIComponent(JSON.stringify({
    selector: 'link[rel=amphtml]',
    dataType: 'html' // since our content is HTML
})); // %7B%22selector%22%3A%22link%5Brel%3Damphtml%5D%22%2C%22dataType%22%3A%22html%22%7D
```

Now, pass two additional parameters to the player:

1. Add the `&readContentType=URL_BY_PAGE_SELECTOR` parameter to our tag, in order to select this method.
2. Pass the hash you received above using the `readContentConfig` parameter: `&readContentConfig=%7B%22selector%22%3A%22link%5Brel...`.

Read the content from WP-JSON API

In this case, your site is using the Wordpress CMS and we can leverage the WP API to read the text directly.

Assuming that our part of the HTML code is:

```
<html>
    <head>
        <link rel="https://api.w.org/" href="https://example.com/wp-json/">
        <!-- Pay attention, by default WP doesn't provide such functionality.
             You have to expose it by yourself -->
        <meta name="slug" content="my-article-123">
    </head>
</html>
```

The resulting URL in the example will be `https://example.com/wp-json/wp/v2/posts?slug=my-article-123` and we'll be using the WP-JSON API to get the text for the provided slug.
Next, let's prepare the query string for the player.

```
encodeURIComponent(JSON.stringify({
    selector: 'link[rel=https://api.w.org]',
    slugSelector: 'meta[name=slug]'
})); // %7B%22selector%22%3A%22link%5Brel%3Dhttps%3A%2F%2Fapi...
```

Now, pass two additional parameters to the player:

1. Add `&readContentType=WP_JSON_BY_PAGE_SELECTOR` parameter to our tag, in order to select this method.
2. Pass the hash you received above using the `readContentConfig` parameter: `&readContentConfig=%7B%22selector%22%3A%22link...`.

API

The Trinity Player provides a simple API for reading its status and controlling it. It's exposed via the global variable `window.TRINITY_PLAYER`.

Property	Type	Description
constants	Object	Constants used in the API, like event names
api	Object	API methods
api.isMultiplePlayers	Boolean	indicates if multiple players are going to be used on the page
api.pause(playerId)	Function	Pause a specific player
api.play(playerId)	Function	Play a specific player. **Please note, it will work only if the user has already clicked play at least once during this session on the page. Otherwise, the**

Property	Type	Description
		browser will throw an error.
api.pauseAll()	Function	Pause all players
api.getFirstPlayer()	Function	Returns (first) player id, e.g. 945a52a1149c91eee4b167958b51e406. Useful when 1 instance of the player is used on one page. In case of multiple players, please access via the `players` object
api.createPlayer(playerId)	Function	(Re)create the player for a certain playerId. Please note that the `playerId` should exist in the `TRINITY_PLAYER.players` object. `playerId` is optional. See Reinit player
api.removePlayer(playerId)	Function	Remove a player with a specific playerId. `playerId` is optional. See Reinit player
api.getDuration(playerId)	Function	Get audio duration for specific playerId
api.setVolume(volume, playerId)	Function	Set audio volume for player
api.getVolume(playerId)	Function	Get audio volume for player
api.setMuted(isMuted, playerId)	Function	Mute/unmute player
api.setCurrentTime(time, playerId)	Function	Seek audio to specific position. Time in seconds
api.getCurrentTime(playerId)	Function	Get current audio progress time
api.getMetadata()	Function	Get currently playing article's metadata
api.setUserId(userId)	Function	Set your custom identifier for the current visitor dynamically
options	Object	Unit configuration
players	Object	Players configuration
players[playerId]	Object	Configuration of a player under a certain Id
players[playerId].resultReadingText	String	Text that will be read by the player
players[playerId].textSelector	String	CSS text selector
players[playerId].state	String	Player current state. `not-started`, `play`, `pause`
players[playerId].enteredView	Boolean	Tells if the player has been shown during this user session
players[playerId].translateTo	String	Selected translation language
isLoaded	Boolean	indicates if the Trinity initial script has been loaded
resultReadingText	String	Text that will be read from the page. Will be overridden with last player in case of multiple players. Check the `players[playerId].resultReadingText` property instead.

api.getMetadata

Get currently playing article's metadata:

```
window.TRINITY_PLAYER.api.getMetadata();
```

Example result:

```
{
    articleSection: "Test Section",
    author: "Article Author",
    contentURL: 'https://example.com/news/article/1',
    dateModified: "2020-02-20T00:00PM-01:00",
    datePublished: "2020-02-02T02:02AM-01:00",
    description: "Test Description",
    faviconURL: "https://www.example.com/favicon.ico",
    imageURL: "https://www.example.com/image.png",
    language: "en",
    publisher: "Test Publisher",
    title: "Test Title"
}
```

Reinit player

In case of an SPA (single page application) where content changes dynamically and the player gets automatically removed from the DOM, there's an option to re-create the player once new content is available.
Possible steps are:

1. Remove the player on route change using `TRINITY_PLAYER.api.removePlayer()`. Although it's an optional step as player is deleted when the old content is removed from the DOM, it will stop the player automatically for you.
2. Re-create the player using `TRINITY_PLAYER.api.createPlayer()`. The player will be rendered either in the provided placeholder (`trinityAudioPlaceholder`) or where the player tag is presented.

Events

All events from Trinity Player fire with `type` `TRINITY_TTS`.
In order to listen for events, you have to wait until injector script is loaded, and use `TRINITY_PLAYER.message.*` to check what event was fired.

Listen for injector script ready

Fires when the Trinity Player initial script loaded, and the `TRINITY_PLAYER` object is in the scope and ready to use. Can be used if you need access to API and need to wait until the player is ready.

```
window.addEventListener('message', (event) => {
  if (event.data?.type !== 'TRINITY_TTS') return;
  if (event.data.value.action === 'injectorImp') {
    console.info('Trinity Audio player injector script is loaded!');
    // now API is ready via window.TRINITY_PLAYER.api
  }
});
```

Listen for events

Format of events being fired is:

```
{
  type: 'TRINITY_TTS',
  value: {
    action: String,
    message: Object|String|undefined,
    playerId: String
  }
}
```

In order to listen to them, filter out all postMessage events by `window.TRINITY_PLAYER.constants.postMessageType` type.

```
window.addEventListener('message', (event) => {
  if (event.data?.type !== window.TRINITY_PLAYER.constants.postMessageType) return;
  console.log(event);
});
```

Events

NOTE: Always use constants provided below, since names of events can be changed. That's the only was to guarantee you persistent of names.

NOTE: In case abtest is enabled every event will contain abtest name in the message body

Event name (action)	Description
injectorImp	when injector script is loaded, and `TRINITY_PLAYER` is ready
TRINITY_PLAYER.message.playerReady	Player is ready for use
TRINITY_PLAYER.message.playClicked	Play is clicked
TRINITY_PLAYER.message.pauseClicked	Pause is clicked
TRINITY_PLAYER.message.resumed	Player resumed
TRINITY_PLAYER.message.contentStarted	Audio content started
TRINITY_PLAYER.message.onFirstQuartile	Audio content is 25% complete
TRINITY_PLAYER.message.onMidPoint	Audio content is 50% complete
TRINITY_PLAYER.message.onThirdQuartile	Audio content is 75% complete
TRINITY_PLAYER.message.onComplete	Audio content is completed
TRINITY_PLAYER.message.adOpp	Ad being requested
TRINITY_PLAYER.message.onAdStarted	Ad is started
TRINITY_PLAYER.message.onAdComplete	Ad is completed
TRINITY_PLAYER.message.enteredView	Player being in view
TRINITY_PLAYER.message.exitedView	Player being out of view
TRINITY_PLAYER.message.FABShow	Player in FAB mode being in view
TRINITY_PLAYER.message.FABHide	Player in FAB mode being out of view
TRINITY_PLAYER.message.touchStart	Touch started (mobile only)
TRINITY_PLAYER.message.touchEnd	Touch ended (mobile only)
TRINITY_PLAYER.message.scrubbing	Audio scrubbed
TRINITY_PLAYER.message.translateTo	Translation was selected

Multiple Players

There are two ways to do that:

1. set `window.__trinityMultiplePlayers__` to `true`
2. set `window.TRINITY_PLAYER.api.isMultiplePlayers` to `true`

In the first case you don't need to wait until the Trinity Player initial script is loaded, and can set the value right away. In the second case you have to wait until the Trinity Player initial script is loaded, and listen to a certain event:

```
window.addEventListener('message', (event) => {
  if (event.data?.type !== window.TRINITY_PLAYER.constants.postMessageType) return;
  if (event.data.value.action === 'injectorImp') {
    console.info('Trinity Audio player injector script is loaded!');
    // need to set it since we have multiple players
    window.TRINITY_PLAYER.api.isMultiplePlayers = true;
  }
});
```

The first case is much simpler, but pollutes the global scope.

Setting Trinity Player to multiple player mode disables `FAB` functionality and automatically enables support for **one playing player at a time**. So if a page has, say, 3 players, once the user clicks on the first one, it will start playing. When the user clicks on the second one - the first one will stop and second one will start playing, and so on...

If you are planning to control each player manually via the **API**, you have to set a unique `data-player-id` attribute to each Trinity tag.

```
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Trinity Page Example</title>
    <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1" />
  </head>
  <body>
    <div class="content">Hello!</div>
    <script data-player-id="player-1"
      src="https://trinitymedia.ai/player/trinity/XXXXXXX/" charset="UTF-8"></script>
    <article class="article-1"></article>

    <script data-player-id="player-2"
      src="https://trinitymedia.ai/player/trinity/XXXXXXX/" charset="UTF-8"></script>
    <article class="article-2"></article>

    <script data-player-id="player-3"
      src="https://trinitymedia.ai/player/trinity/XXXXXXX/" charset="UTF-8"></script>
    <article class="article-3"></article>
  </body>
</html>
```

Setting that, you are able to control a player via the **API**, calling **pause()**, **play()** methods passing a corresponding player ID, e.g.

```
window.TRINITY_PLAYER.api.pause('player-3');
window.TRINITY_PLAYER.api.play('player-3');
```

Also note, **pageURL** is a required field when using multiple players as well and is expected to pass a unique value per content.

Custom text selector

By default, the text selector is controlled by the unit configuration. But if you want to take control over it, or use multiple players on one page, say, in an SPA, you can pass you own CSS text selector. This technique is useful when you have multiple articles on a page and want each player to read its own article.

In order to do that, just pass a `textSelector` parameter to Trinity tag. Selector value should be encoded into base64 and the resulting string should be URI encoded.

See passing CSS selector

Now just pass it as a query parameter, and the player will read text from appropriate selector.

```
<script data-player-id="player-123456789"
src="https://trinitymedia.ai/player/trinity/XXXXXXX/?textSelector=I2FydGljbGUtMTIz" charset="UTF-8"></script>
```

In order to test it on the fly, execute the following code in the browser's JavaScript console

```
var textSelector = '#article-123';

var js = document.createElement('script');
js.type = 'text/javascript';
js.setAttribute('fetchpriority', 'high');
js.setAttribute('charset', 'UTF-8');
js.setAttribute('data-player-id', 'player-123456789');
js.src = 'https://trinitymedia.ai/player/trinity/XXXXXXX/?textSelector=' +
  encodeURIComponent(btoa(textSelector));

document.body.appendChild(js);
```

The player will get loaded and should appear on the page. In order to check if it will read the correct text, you can execute

```
TRINITY_PLAYER.players['player-123456789'].resultReadingText
```

> Please note, that you can't change `textSelector` on the fly. When the player is loaded with an appropriate selector - text selector can't be changed, and the player will play text from the initially provided text selector.

Text Filtering

In case you would like to mark some of the text for our player **not** to read, just create an element with a class name of `trinity-skip-it` and we'll do the rest.

```
<p class="trinity-skip-it">...</p>
```

Helpers

Passing CSS selector

```
const textSelector = encodeURIComponent(btoa('#article-123'));
```

Result of that operation will be `I2FydGljbGUtMTIz` string.

Passing config

```
const config = encodeURIComponent(JSON.stringify({
  // some options goes here, e.g.
  x: 1,
  y: 2
}));
```

Result of that operation will be `%7B%22x%22%3A1%2C%22y%22%3A2%7D` string.

Passing CMS Attribute

```
const cms = encodeURIComponent(JSON.stringify({
    channelHash: 'a5gTS2', // The relevant channel hash as shown in the CMS
```

```
    tags: ['money', 'business', 'success'] // The relevant tags for the content
}));
```

Result of that operation will be
`%7B%22channelHash%22%3A%22a5gTS2%22%2C%22tags%22%3A%5B%22money%22%2C%22business%22%2C%22success%22%5D%7D` string.

Improve player loading time

```
<link href="https://trinitymedia.ai" rel="preconnect" crossorigin="anonymous"/>
<link href="https://vd.trinitymedia.ai" rel="preconnect" crossorigin="anonymous"/>
```

Code provided above helps to improve player loading time due to pre-connect to defined domains in advance.
Add HTML provided above inside head or body tag. Note, that code should be added before the Player tag, e.g.

```
<link href="https://trinitymedia.ai" rel="preconnect" crossorigin="anonymous"/>
<link href="https://vd.trinitymedia.ai" rel="preconnect" crossorigin="anonymous"/>
<script src="https://trinitymedia.ai/player/trinity/XXXXXXX/" charset="UTF-8"></script>
```