info@readspeaker.com
www.readspeaker.com

ReadSpeaker docReader JavaScript API

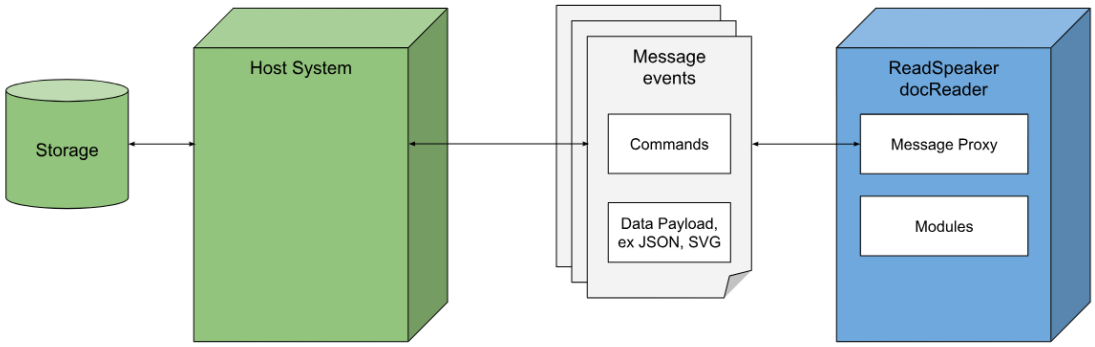

Illustration: Communication flows between the host system and docReader.

Introduction

The ReadSpeaker docReader JavaScript API allows you to enhance the functionality of ReadSpeaker docReader. Through the API your web application can integrate directly with certain features in docReader, such as the Annotation tool, the Highlighter tool, as well as some general features like audio playback, document meta data and page navigation.

This document describes the technical details of this API.

A select number of tools in docReader can be controlled by an external host system by sending postMessage events. These events are read by the docReader *message proxy* that listens to incoming messages from certain predefined URIs.

The message proxy will expect messages to be structured in a certain way. Generally, this means wrapping the API calls in literal JavaScript objects in the `message.data` property of the message object.

The docReader message proxy will handle two different categories of messages: “dcevent” and “dcdata”.

Event Messages

Events are dispatched from docReader in response to certain user-interactions (e.g. page changed, reading started, tools selected, annotations changed, and so on) or when specific criteria are met. Through the message proxy, you can register event-triggers for events in docReader. Each event can send parameters to the handler function.

Event Registration

In order for the host system to catch an event, it has to be registered in docReader. This is the expected format for registering an event (stored in messageEvent.data):

```
{
  type: 'dcevent',
  eventType: 'eventName',
  eventId: 'eventId'
}
```

`type`	Tells the message proxy that this is a dcevent registration.
`eventType`	Which kind of event we want docReader to react to. There is a list of possible events below.
`eventId`	A unique ID that will be sent as a parameter when the event is dispatched. We explain this in more detail below.

Example

```
window.postMessage({
      type: 'dcevent',
      eventType: 'modcommandsent',
      eventId: 'myrandomeventid1234'
}, '*');
```

Event Response

When a registered event is dispatched in docReader, the message proxy will broadcast a response to all frames that listen to message events.

This is what the event response message typically looks like:

```
{
  type: 'dcevent',
  params: {
    eventId: eventId,
    args: [arguments]
  },
  eventType: 'pagechanged'
}
```

`type`	Tells the receiver that this message is in response to an event.
`params`	The event parameters.
`eventId`	The event ID that was previously sent when registering the event.
`args`	An array with any arguments that were sent back from the event. These vary depending on which event type was dispatched. See list below.
`eventType`	The type of event that was dispatched. E.g. `pagechanged`, `modcommandsent`. Added in docReader 4.9.0.

Events

These are the events that can be listened for. Note that some events send arguments that the receiving handler can make use of (see args above).

Event name	Description	Arguments
`documentloadingerror`	This event will be fired if an error occurs while loading a document. Example of errors could be: document does not exist/can't be reached, backend fails to parse, frontend fails to get data from backend.	None
`beforemodinitialized`	Fires right before a module is initialized.	{String} `modname` - The name of the module that is to be initialized.
`beforepagechanged`	Fires when a user has navigated to a different page, but before the page is actually changed. This event is triggered when docReader is first loaded, since it fetches the pages via an AJAX call.	{Number} `page` - The page number we are changing to. {Bool} `firstLoad` - Whether or not this is the first page that was loaded.
`modcommandsent`	Fires when a module sends a command of some sort. Its use varies depending on the module. For the annotations module (svgtools), this event is mostly used when the integrated toolbar is used.	*See description below.*

Event name	Description	Arguments
`modechanged`	This event happens when the user switches between layout mode and text mode in docReader. It can be relevant to the Annotation tools, because you probably want to hide the annotations while in text mode.	{String} `mode` - The name of the new mode. `layoutmode`|`textmode`
`modinitialized`	Fires after a module has been initialized.	{String} modname - The name of the module that was initialized.
`pagechanged`	Fires after docReader changed to a different page. This is where you want to fetch and apply any annotations that were saved for the new page.	{Integer} `pagenum` - The current page number. {Integer} `pagecount` - The total number of pages in this document.
`pageinitialized`	This event fires after a page has been loaded and then also initialized, so in other words after `pagechanged`/`pageloaded`.	None
`pageloaded`	Fires when a page has been loaded. The difference between this event and `pagechanged` is that this event only fires once, when docReader has been loaded, whereas `pagechanged` fires on each subsequent page load. `pagechanged` also does not fire on the first page load.	{Integer} `pagenum` - The current page number. {Integer} `pagecount` - The total number of pages in this document.
`pageresized`	Happens when the user resizes his or her browser window.	{Float} `xratio` - Is the ratio of the rendered document width relative to its original width.

Event name	Description	Arguments
`readpaused`	Fires when the audio playback was paused.	None
`readstarted`	The audio was started or resumed after having been paused.	None
`readstopped`	The audio was stopped, either because the user clicked the stop button, or because the audio had played all the way to the end.	None

modcommandsent

As noted above, the modcommandsent event fires when a docReader module (such as highlighter or annotations) issues a command of some sort. The returned argument for modcommandsent events is an object with the following structure:

```
{
      mod: 'svgtools',
      commandType: 'command',
      commandParams: ['save']
}
```

- {String} `mod` - The name of the module that sent the command.
- {String} `commandType` - Types can be used to categorize commands, in order to more easily filter them.
- {Array} `commandParams` - Any parameters sent along with the command. These vary depending on the command. Some examples are `'save'` and `'print'`. The best way to determine which parameters are sent back is to log the `commandParams` property to the browser console.

modcommandsent Events From the Highlighter Tool

The highlighter tool in docReader sends the predefined “modcommandsent” events outlined below.

modcommandsent Name	Description	Data
`highlighter.change`	One or more highlights were changed (this includes deleting highlights).	JSON representation of affected highlights
`highlighter.collectdialog`	Dialog for collecting highlight data was called.	
`highlighter.clearpagehighlights`	The highlighting module requested that a confirmation dialog for “Clear page highlights” should be shown.	
`highlighter.cleardochighlights`	The highlighting module requested that a confirmation dialog for “Clear page highlights” should be shown.	

Example of an event response with modcommandsent content:

```
{
  type: 'dcevent',
  eventId: '432',
  args: [
    'highlighter.change',
    '[{"start":13,"stop":13, ... }]'
  ]
}
```

Data Messages

Data messages are used when (near) immediate feedback is expected, for instance for fetching values, or sending commands to control specific tools.

Data Message

This is the structure of a data message object:

```
{
  type: 'dcdata',
  dataId: 'dataId',
  dataObj: {
    'apiCommand': [parameters],
    'anotherApiCommand': null
  }
}
```

`type`	Tells the message proxy to treat this as a data message.
`dataId`	A unique id that will be sent back as a reference in the message response.
`dataObj`	Contains an object where each API command is a property. This way, multiple API commands can be executed at once. They will be executed in the order they appear in the object.

Example

```
window.postMessage({
      type: 'dcdata',
      dataId: 'myrandomdataid1234',
      dataObj: {
            'highlighter.displayModeSet': ['edit']
      }
}, '*');
```

Data Message Response

Data messages will be processed immediately by the docReader message proxy. It will try to execute the requested commands in the order they appear, and then return the response from each command in an object:

```
{
  type: 'dcdata',
  params: {
    dataId: dataId,
    returnObj: {
      apiCommand: returnValue,
      anotherApiCommand: returnValue
    }
  }
}
```

`type`	Informs us that this is a response to a data message.
`params`	The response parameters.
`dataId`	The same dataId that was provided when the data message was first issued. This is returned in order for the receiver to distinguish different data message calls.
`returnObj`	This is where the response from the different API commands are stored. The return values differ depending on which command was called.

Data Message Commands

Data message commands are divided into categories that are specified in the command string in the form of `'category.command'`. Currently, the available categories are:

- `annotations` – for controlling the docReader annotation tools
- `highlighter` – for controlling docReader color highlights
- `general` – misc docReader commands

If no category is specified (if only `'command'` is given), the command is first searched for in the *annotations* category and, if not found there, in the *general* category, for back compatibility with older versions of the API.

Categories and commands are listed below.

Annotations Commands

The following commands are within the 'annotations' category, for example `'annotations.contentClear': []`.

contentClear

Parameters	None
Returns	Void

Clears all annotation data, and inserts the default, empty SVG template.

Note that executing this command does not change the saved data for this page.

contentGet

Parameters	None
Returns	{String} The outer SVG data. This includes the <svg> element itself.

This command is used to retrieve the current state of the annotations. Some auxiliary elements, such as drag handles and icons are removed before returning the data.

contentGetForPrint

Parameters	None
Returns	{String} The outer SVG data. This includes the `<svg>` element itself.

Prepares the SVG for print and returns it. The difference between `contentGet` and `contentGetForPrint` is that more auxiliary elements are removed when preparing for print.

contentSet

Parameters	{String} `svgdata` - SVG data that will be injected into the page. This should be data that was previously fetched using `contentGet`.
Returns	Void

This command clears the current SVG data and replaces it with the data provided in the svgdata parameter. The event `contentSet` is fired immediately after the content has been updated.

dirtyGet

Parameters	None
Returns	{Boolean} True if the content has changed, false otherwise.

Fetches the current dirty state, i.e. if the content has changed since it was loaded or last reset.

dirtyReset

Parameters	None
Returns	{Void}

Resets the dirty state. This is useful if we have saved the content and want to be notified when the content changes next. If not reset, the dirty state will remain the same even if the content is changed.

displayModeGet

Parameters	None
Returns	{String}

Fetches the current display mode as a string.

displayModeSet

Parameters	{String} `mode` - The annotation display mode. `hidden`|`edit`|`view`|`get`
Returns	{Void|String}

Switches the display mode of the docReader annotations:

- Hidden - The annotations are not displayed at all.
- Edit - Annotations are displayed and can be edited by the user.
- View - Annotations are displayed but cannot be edited.
- Get - Returns the current display mode.

Hidden, Edit, and View cause the `displayModeChanged` event to fire, with the name of the display mode as a parameter.

displayModeToggle

Parameters	None
Returns	{Boolean} True if display mode was changed, false otherwise. It could be false if the current view mode is set to `textmode`, for instance.

Toggles the display mode between **`edit`** and **`view`**, depending on the current mode.

phrasesSet

Parameters	{Object} phrases - containing strings with names.
Returns	{Boolean} Returns true if successful.

Sets the phrases, i.e. labels, of the annotation tools, shapes, colors, etc.
To see an example of a phrases object, use the command phrasesGet (see below). Example phrases object:

```
{
  colorpicker: "Select color",
  colors: {black: "Black", red: "Red", white: "White", blue: "Blue", ...},
  editandread: "Read text",
  ellipsetool: "Ellipse tool",
  emoji: "Emoji",
  entertext: "Enter text",
  freehandtool: "Freehand tool",
  ignore: "Ignore",
  ignoreall: "Ignore All",
  linetool: "Line tool",
  markertool: "Marker tool",
  nospellingerrors: "No spelling errors found.",
  print: "Print",
  recttool: "Rectangle tool",
  save: "Save",
  stickynotetool: "Sticky Note tool",
  texttool: "Text tool",
}
```

toolSet

Parameters	{String} `tool` - The machine-readable name of the tool to activate.
Returns	{String} Returns the tool parameter.

Sets the annotations tool to use. These are the available tools:

- text - The text tool. Inserts plain text.
- line - The line tool.
- ellipse - Ellipse tool.
- rect - Creates outlined rectangles.
- marker - Creates filled, semi-transparent rectangles.
- selection - Also known as Read Text. Not a tool per se, but is a hybrid display mode, where the toolbar and the annotations are visible while it is possible to select text for docReader to read.
- freehand - Freehand drawing tool.
- stickynote - Sticky note tool.
- emoji - Emoji tool (adds emoji symbols as objects).
- marquee - Marquee selection tool to select existing objects.

toolGet

Parameters	None
Returns	{String} The currently selected tool.

Returns the name of the currently selected tool. (The description for *toolSet* has a list of possible values.)

toolColorSet

Parameters	{String} `color` - The machine-readable color name to use.
Returns	{String} Returns the color name.

Sets the currently active color, which all tools will use. You can list all available colors by using the `toolColorList` command. See below for more information.

toolColorGet

Parameters	None
Returns	{String} Returns the name of the currently active color.

Returns the name of the currently active color. You can map color names to color codes, by fetching a list of all available colors, using the `toolColorList` command.

toolColorList

Parameters	None
Returns	{Object} An object where the name is the key and the color code is the value.

This command can be used to get a list of all available colors. It returns an object where the color names are the keys and their respective colors are the values.

Example of returned data from toolColorList:

```
{
      black: "rgb(0, 0, 0)",
      white: "rgb(255, 255, 255)",
      red: "rgb(255, 0, 0)",
      green: "rgb(0, 200, 0)",
      blue: "rgb(0, 0, 255)",
      yellow: "rgb(255, 255, 0)"
}
```

toolStrokeSet

Parameters	{Integer} `stroke` - The stroke width to use, such as 2, 4, 8, etc.
Returns	{Integer} The stroke width.

Sets the currently active stroke width, which all tools will use. You can list all available stroke widths by using the `toolStrokeList` command.

toolStrokeGet

Parameters	None
Returns	{Integer} The current stroke width value.

Returns the value of the currently active stroke width, such as *2* or *10*.

Highlighter Commands

The following commands are within the 'highlighter' category, for example `'highlighter.displayModeSet': ['edit']`.

displayModeSet

Parameters	{String} mode - the highlighter tool viewing/editing mode `edit|view|off`
Returns	{String} - the mode that was set

Selects the viewing/editing mode of the highlighter tool. Possible modes are:

- `edit` - Highlighter toolbox is open and it's possible to highlight text portions by selecting a color and using click and drag with the mouse.
- `view` - Highlighter toolbox is closed but existing highlights are on (visible)
- `off` - Highlighter toolbox and existing highlights are turned off (not visible)

displayModeGet

Parameters	None
Returns	{String} Returns the current mode (edit, view, or off)

displayModeGet returns the current display mode as set by displayModeSet.

displayModeToggle

Parameters	None
Returns	{Void}

Toggles the display mode between **`edit`** and **`view`**.

contentClear

Parameters	{Boolean} `skipUpdate` - If set to `true`, will prevent the `highlighter.change` event to be triggered.
Returns	{Boolean} Returns true if successful

Removes all highlights from the page and the corresponding JavaScript object in the browser frontend.

contentShow

Parameters	None
Returns	Void

Makes all highlights on the current page visible.

contentHide

Parameters	None
Returns	Void

Makes all highlights on the current page hidden.

contentGet

Parameters	None
Returns	{String} Returns all highlights on page as JSON data

Gets all highlights on the current page as stringified JSON data. The actual JSON object consists of an array in which each element is an object representing a highlight on the page, with information on position, color, etc.

Example of a returned returnObj with JSON string:

```
{'highlighter.contentGet':
'[{"start":1,"stop":1,"type":"green","text":"Study","version":2},
{"start":13,"stop":13,"type":"green","text":"operating","version":2
}]'}
```

contentSet

Parameters	{String} Page highlights as string with JSON data
Returns	None

Sets highlights on the current page from JSON data. The actual JSON object consists of an array in which each element is an object representing a highlight on the page, with information on position, color, etc.

Example dataObj:

```
{'highlighter.contentSet':
['[{"start":1,"stop":1,"type":"green","text":"Study","version":2},
{"start":13,"stop":13,"type":"green","text":"operating","version":2
}]']}
```

General Commands

The following commands are within the 'general' category, for example `'general.audioStop': [true]` (where true refers to HTML5 mode in this case).

audioLinkGet

Parameters	None
Returns	{String} Link to the current audiofile.

Pressing the play button in docReader will result in an audiofile being generated. This method fetches a link to the audiofile, even the audio has not yet been generated.

audioPause

Parameters	None
Returns	Void

Pauses currently playing audio.

audioPlay

Parameters	{Boolean} isHtml5 - Deprecated, kept only for backwards compatibility. It will not make any difference which value you use. {Boolean} isResume - True = resume paused audio instead of playing audio from the beginning of the page. False = Play audio from the beginning.
Returns	Void

Starts or resumes audio playing.

audioStop

Parameters	None
Returns	Void

Stops currently playing audio.

docIdGet

Parameters	None
Returns	{String} The internal document ID.

Fetches the internal document ID of the current document.

docUrlGet

Parameters	None
Returns	{String} The URL of the current document.

Fetches the full Url of the location where the current document was retrieved from.

docJobnameGet

Parameters	None
Returns	{String} The internal job name of the current document.

Fetches the internal job name of the current document.

modsListGet

Parameters	None
Returns	{Array} A list of the docReader modules that have been loaded.

Fetches a list of all docReader modules that are currently loaded.

mp3Download

Parameters	None
Returns	Void

Triggers audio file download. Only works if mp3 download is enabled in docReader config.

pageCurrent

Parameters	None
Returns	{Number} Number of the current page.

Returns the current page number in docReader.

pageCount

Parameters	None
Returns	{Number} The total number of pages in the document.

Returns the total number of pages in the current document.

pageGoto

Parameters	{Int} pageNum - The page we want to go to.
Returns	{Bool} True if we could go to the requested page, false otherwise.

Changes to the requested page. Returns false if pageNum is less than zero, or greater than the page count in the document.

pageImageSrc

Parameters	None
Returns	{String} URL to image of current page. Image format is either SVG or PNG, depending on your configuration. Default is SVG.

Returns a URL to an image of the current docReader page.

pagePrint

Parameters	None
Returns	Void

Brings up the print document dialog in the browser.

settingsSet

Parameters	{Object} params - Settings names and values as key-value pairs.
Returns	{Boolean} Returns true if the settings were successfully updated.

Offers a way to update all docReader settings in one go. This method accepts an object containing key-value pairs for each docReader setting.

Note that the form element will be overwritten with the new values, so if you omit a setting, it will lack a value.

viewmodeGet

Parameters	None
Returns	{String} The current viewmode, either `textmode` or `layoutmode`.

Returns the current viewmode, which is either text mode or layout mode.

voiceGet

Parameters	None
Returns	{String} The currently used TTS voice.

Fetches the name of the TTS voice that is currently being used by docReader.