ReadSpeaker webReader 3.8 Skinning Guide, 20 November 2023

ReadSpeaker webReader Skinning Guide

Disclaimer

Please note that ReadSpeaker can only guarantee that webReader is WCAG-compliant when the default skin is used. Any changes to the design of webReader may render the product non-compliant.

We therefore advise you to review your custom skin against the Web Content Accessibility Guidelines.

Working with Skins

Skins can be used to customize the appearance of ReadSpeaker webReader. Every part of the graphical interface in webReader can be altered with standard CSS (Cascading Style Sheets).

Skins can either be built from scratch or be adaptations of the default skin. Which method you choose depends on what you want to accomplish with your skin. If it's merely a matter of changing colors or fonts, then you should work with the default skin and only define CSS rules that change the existing appearance. But if you want to do something completely different, it might be better to start with a blank slate.

Please note that when we release a minor update of ReadSpeaker webReader (this happens twice per year), functionalities can be moved within the product or removed from it. This means that you might have to update your skin for it to work as intended.

The Skin Files

A webReader skin consists of at least two files, a CSS file and a Javascript file. All files should have the same name, except for the extension. They should be placed in a folder with the same name, but without extension.

For example, if the skin is named mySampleSkin:

- mySampleSkin/ <== folder
    - mySampleSkin.css <== CSS file
    - mySampleSkin.js <== Javascript file

The skin folder is expected to be served from the same location as the webReader.js file in a folder named `skins`.

Example: If the webReader.js file is in `https://domain.com/webReader.js`, then the folder of the skin will be `https://domain.com/skins/mySampleSkin`.

Configuring webReader to Use a Skin

In order to use a customized skin, you need to tell webReader to look for it. This is done by adding the `skin` parameter to the script reference.

```
<script src="//domain.com/webReader.js?pids=wr&amp;skin=mySampleSkin"></script>
```

If you are using the cloud hosted scripts when developing the skin, you need to add references to the skin files to your HTML code.

Please note that the references should be added before the `webReader.js` file.

```
<link rel="stylesheet" href="//domain.com/mySampleSkin/mySampleSkin.css" />

<script src="//domain.com/mySampleSkin/mySampleSkin.js"></script>

<script
src="//cdn1.readspeaker.com/script/[CID]/webReader/webReader.js?pids=wr"></script>
```

Note that [CID] needs to be replaced with the customer id for your ReadSpeaker account (don't include the brackets).

The Anatomy of the Player

There are two versions of the player, one used on mobile devices and one used on computers. We will refer to them as the mobile UI and the desktop UI in this guide.

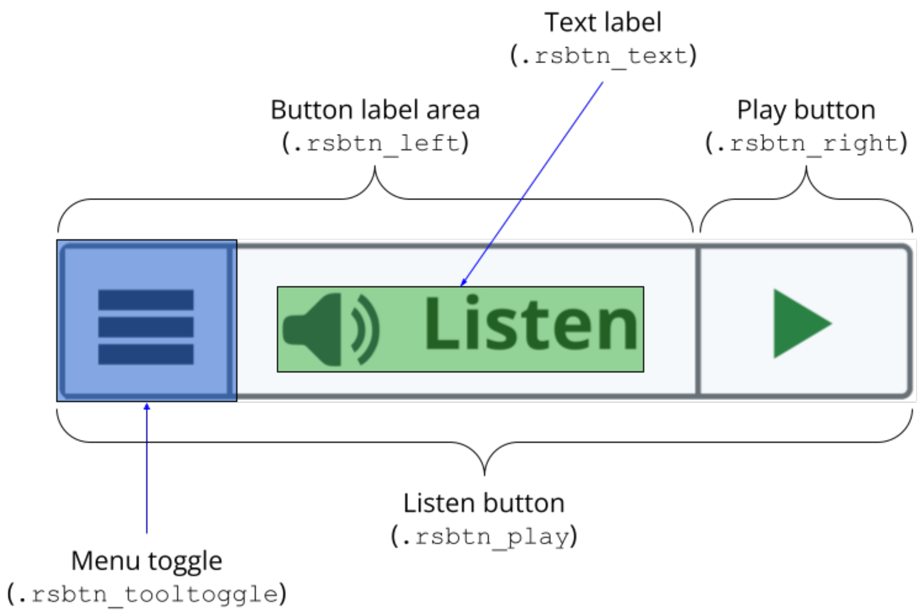
The screenshots show the most important parts of the different components, but there are elements that are not included in the screenshots. Look at the code in the browser's developer tools to get the full picture.

The Desktop UI

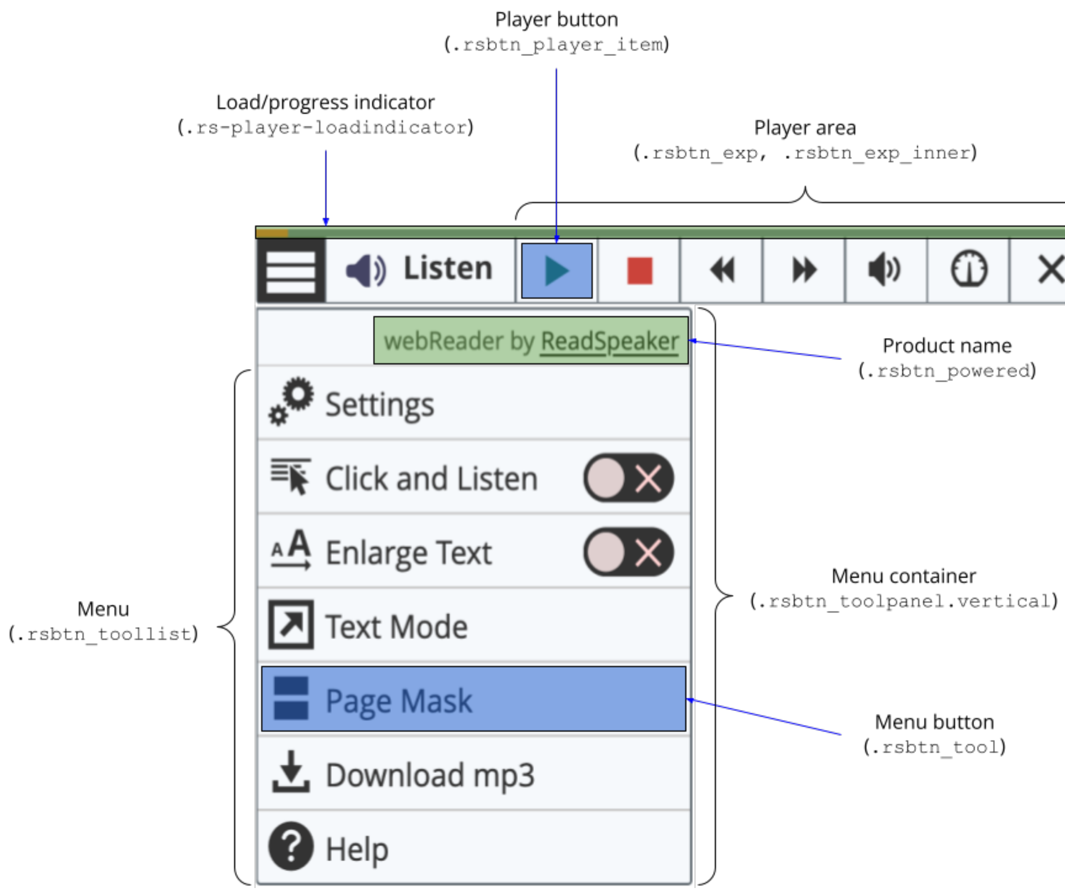
The Listen Button and Player

Before the user has interacted with webReader, and when the service is inactive, it is represented by the Listen button. A single click on it will start the reading and open up the player controls.

The Listen button also contains a button that toggles the visibility of the menu, where additional webReader related functions and settings are located.

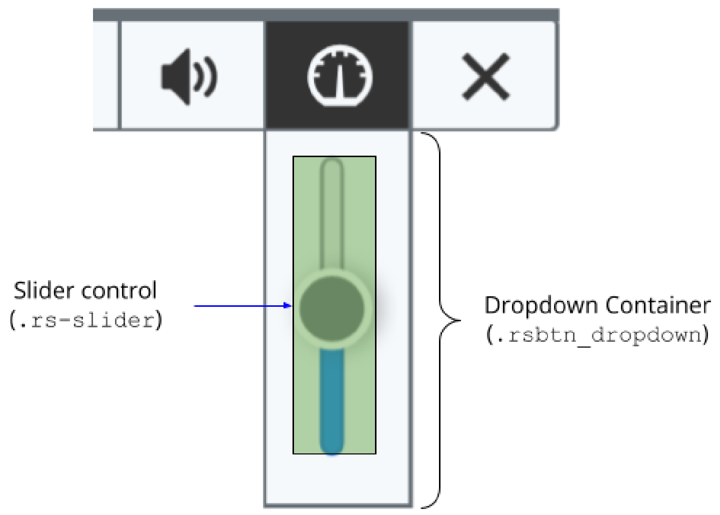


When the user starts the reading, the Listen button expands into a player with buttons for controlling the playback.

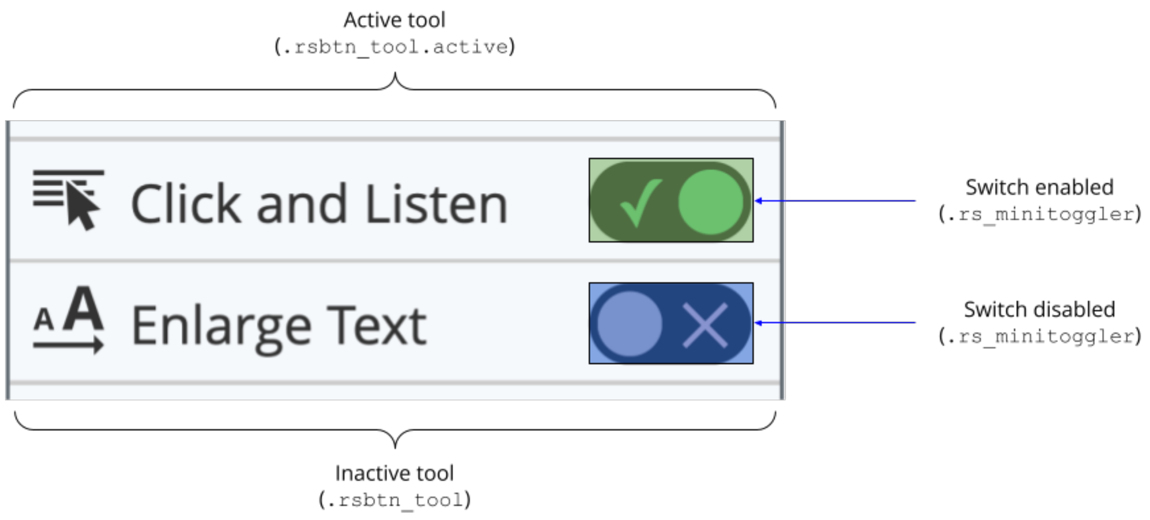


Starting with version 3.5.0, webReader's toolbar is a vertical menu by default. It contains functions that are not directly used for controlling the audio, but rather tools that enhance the reading experience in general. This is also where you find the Settings menu and the Help page. Some menu items have a toggle switch that gives a visual indication of whether the feature is active or not (see below).

The volume and speed buttons have drop-down panels with slider controls. Starting with version 3.5.0, these are standard HTML5 input range elements.

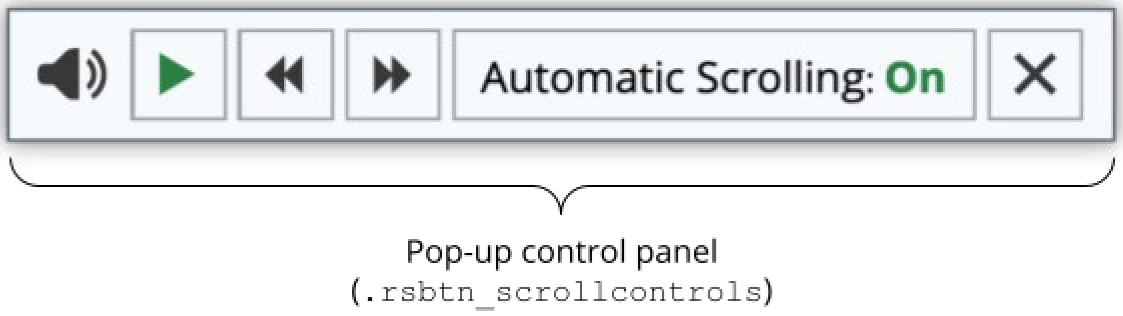


The functions Click and Listen and Enlarge Text in the menu can have two different states, active or inactive. A toggle switch gives a clear visual indication whether the feature is enabled or not. The `.active` class on the surrounding button element controls the state of the toggle switch.



The Pop-Up Control Panel

When webReader is active, a small control panel will appear in the bottom right corner so that you can easily pause or resume the reading, as well as turn automatic scrolling on or off.



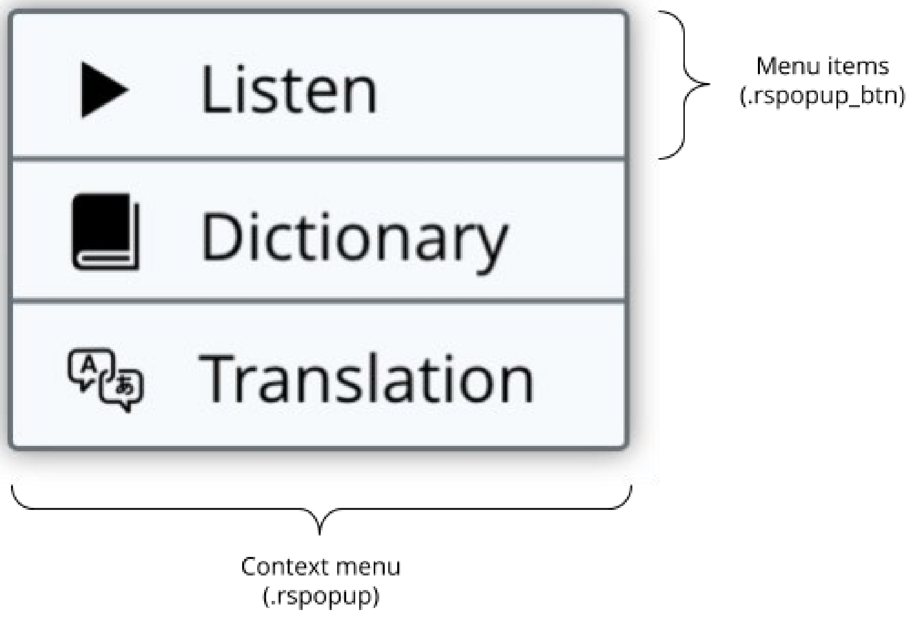
Note: When the Enlarge Text feature is enabled, the pop-up control panel will appear at the top right corner of the screen, so as to not interfere with the enlarged text box.

The Context Menu

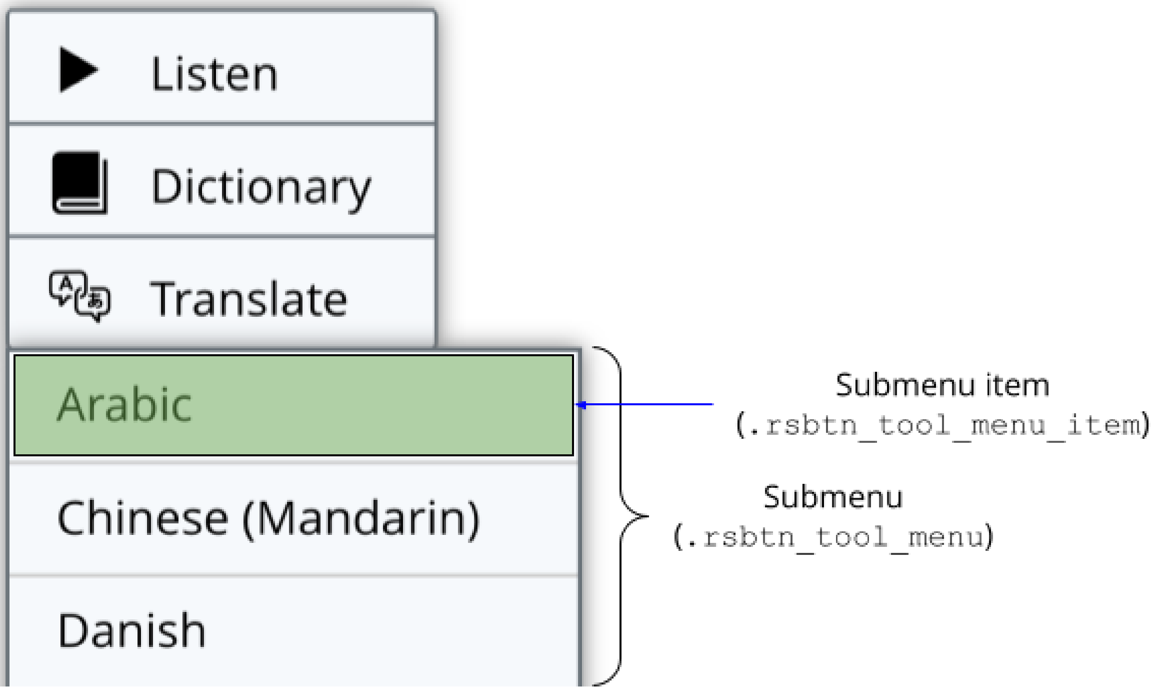
The context menu appears when the user selects text on the page. Depending on the direction of the selection and the available screen space, the context menu may appear on different sides of the mouse pointer, but always adjacent to it.

Depending on your configuration, the context menu may have all or a subset of the items in the screenshot below.

Note that the context menu is available even if the user has not interacted with the Listen button before. In the webReader Help section, the context menu is referred to as the pop-up menu for selected text.



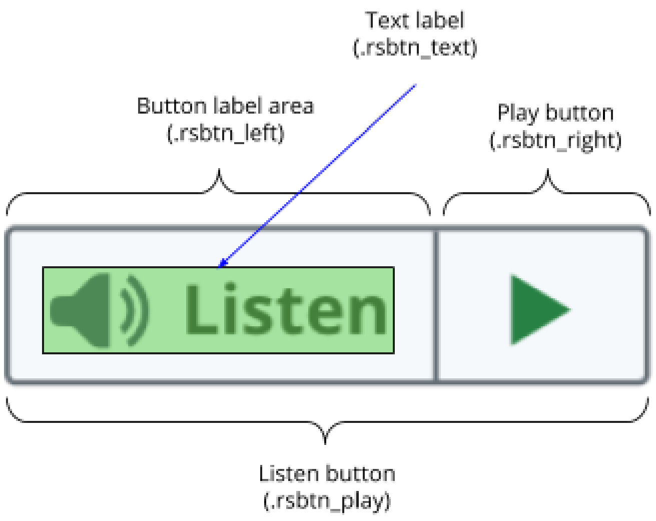
Menu options may have a submenu that will open either below or above the menu item, depending on the available space.



The Mobile UI

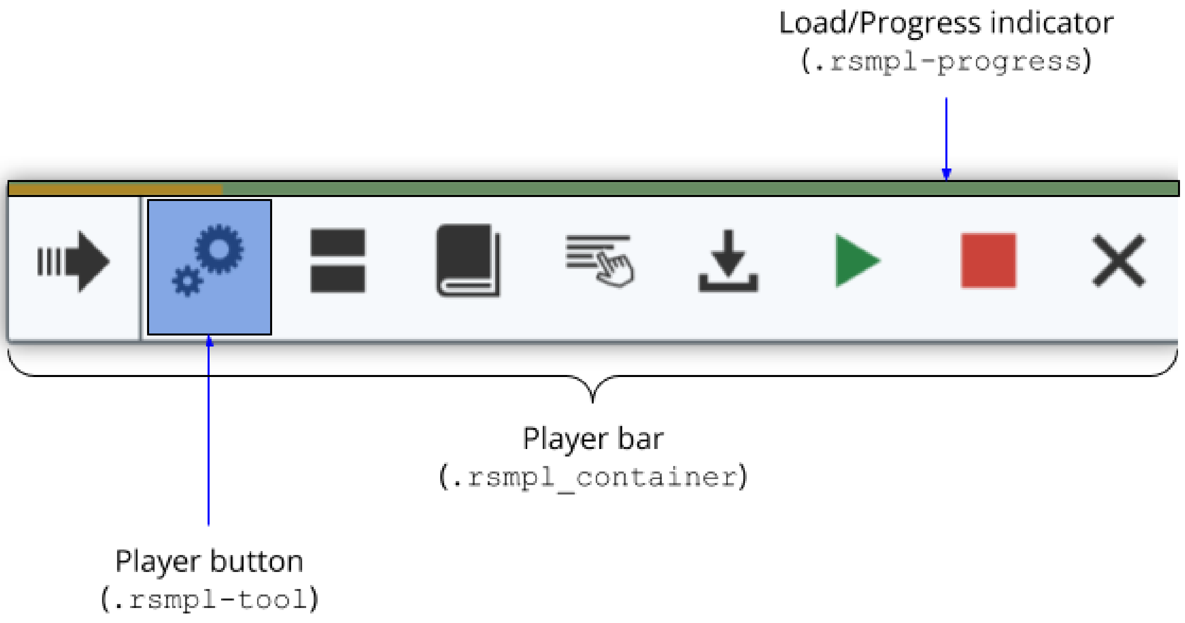
The mobile UI differs from the desktop UI in order to compensate for the smaller screen size. These are the main components:

The Listen Button



This is the same as in the desktop UI, but it lacks the toolbar toggle.

The Player Bar



This is where the playback controls and tool buttons are located in the mobile version. By default, the player bar slides out from the bottom right corner of the screen, to make it easy to reach with one hand.

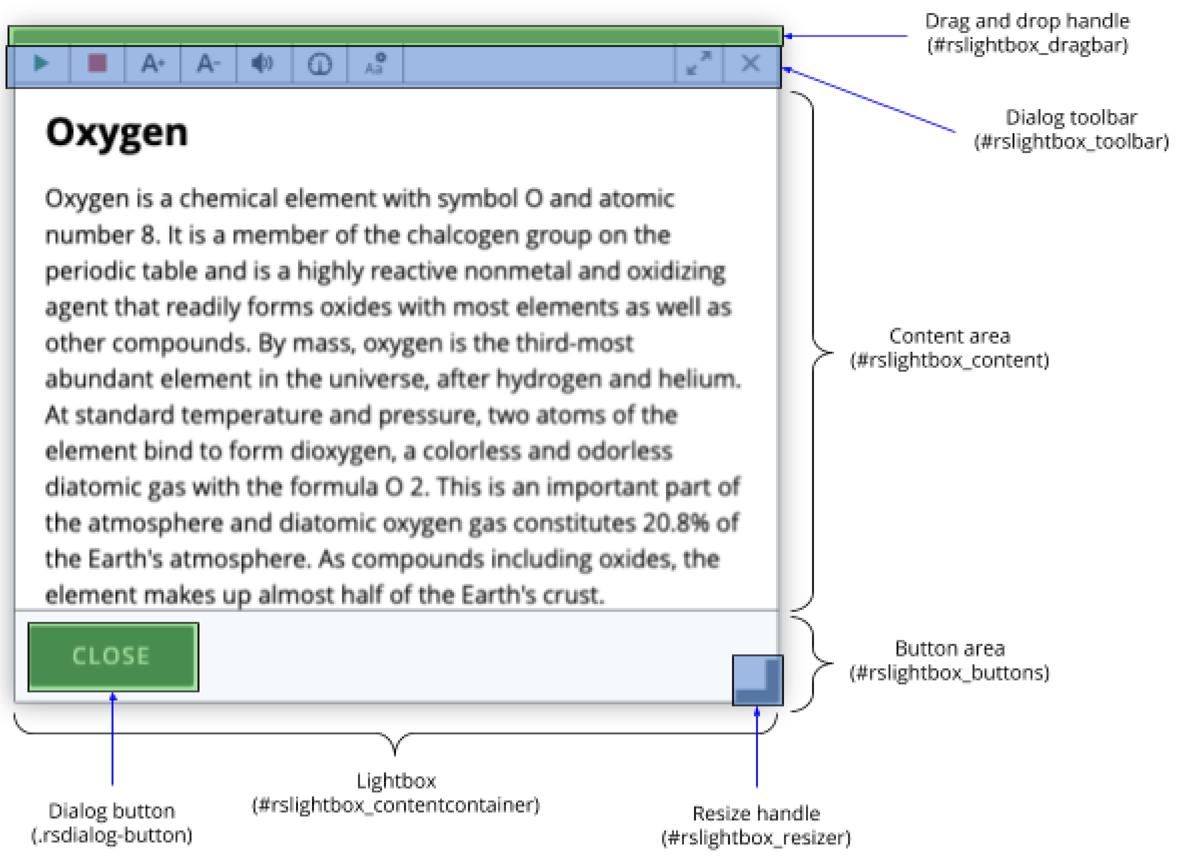
The vertical placement can be changed in the configuration. See the Configurations page on the webReader Developer site for more information.

The Lightbox

The lightbox, or webReader pop-up window, is common for both the desktop and mobile UIs, and is used by different functions like Translation, Text Mode, Settings, and others, for focusing on certain content. The overall layout is the same regardless of which function uses it.

The lightbox floats above the rest of the page content and is modal, meaning that no other elements can be interacted with while the lightbox is open.

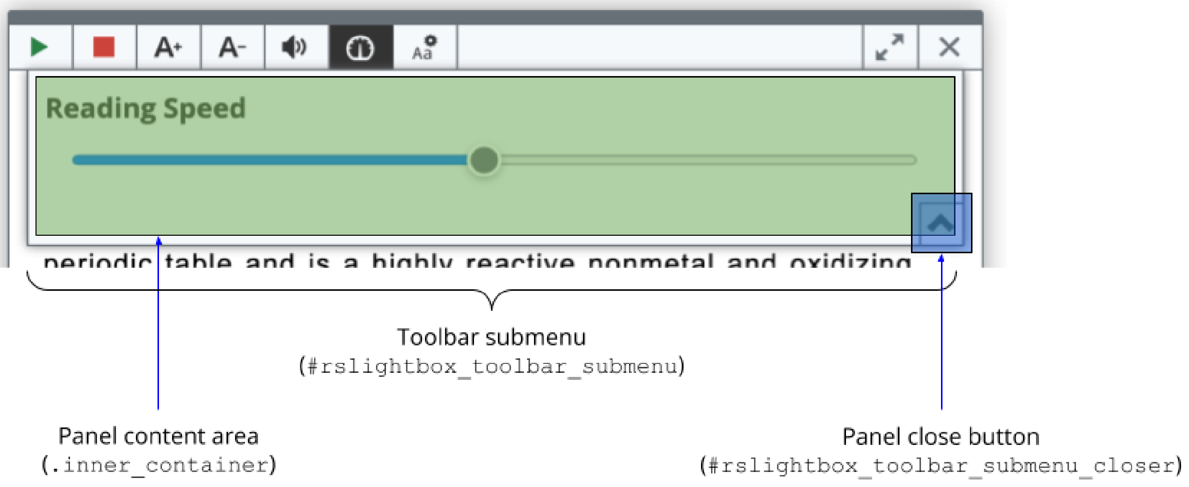
Even though different functions use the lightbox, it is one single HTML element and most of its subparts are referenced by ID, not class.



Oxygen

Oxygen is a chemical element with symbol O and atomic number 8. It is a member of the chalcogen group on the periodic table and is a highly reactive nonmetal and oxidizing agent that readily forms oxides with most elements as well as other compounds. By mass, oxygen is the third-most abundant element in the universe, after hydrogen and helium. At standard temperature and pressure, two atoms of the element bind to form dioxygen, a colorless and odorless diatomic gas with the formula O 2. This is an important part of the atmosphere and diatomic oxygen gas constitutes 20.8% of the Earth's atmosphere. As compounds including oxides, the element makes up almost half of the Earth's crust.

The toolbar buttons can have drop-down panels for settings that do not need to be immediately visible.



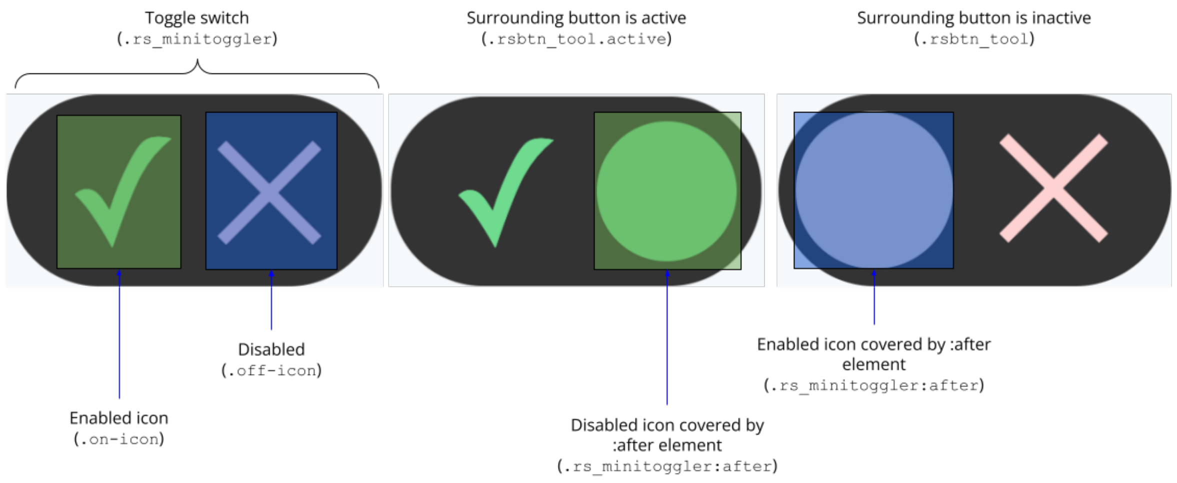
Reading Speed
periodic table and is a highly reactive nonmetal and oxidizing
Toolbar submenu
(#rslightbox_toolbar_submenu)
Panel content area
(.inner_container)
Panel close button
(#rslightbox_toolbar_submenu_closer)

The lightbox inserts an overlay that covers the entire webpage. The overlay can be restyled by targeting the #rs_overlay element.

Other Components

Toggle Switch

Let's take a closer look at the different parts of the toggle switch.



The switch consists of a container (.rs_minitoggler), a checkmark icon, and a cross icon. Depending on the state, one of the icons will be covered by a circle, which is defined as an `:after` element of the container.

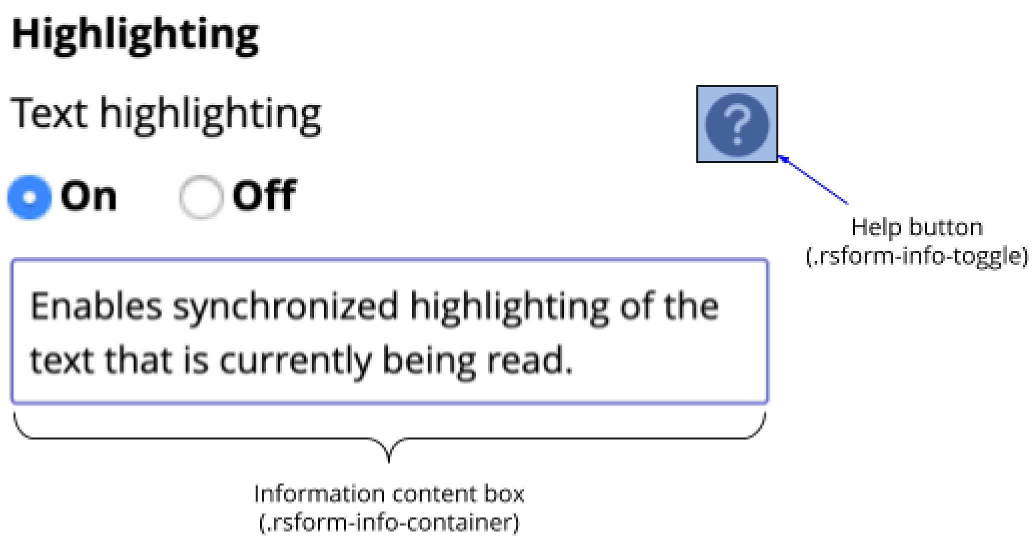
The state of the toggle switch is determined by its parent `button` (`.rsbtn_tool`) element.

Form Controls

The Settings lightbox contains different types of form controls, most of which are standard HTML radio, select, and button elements.

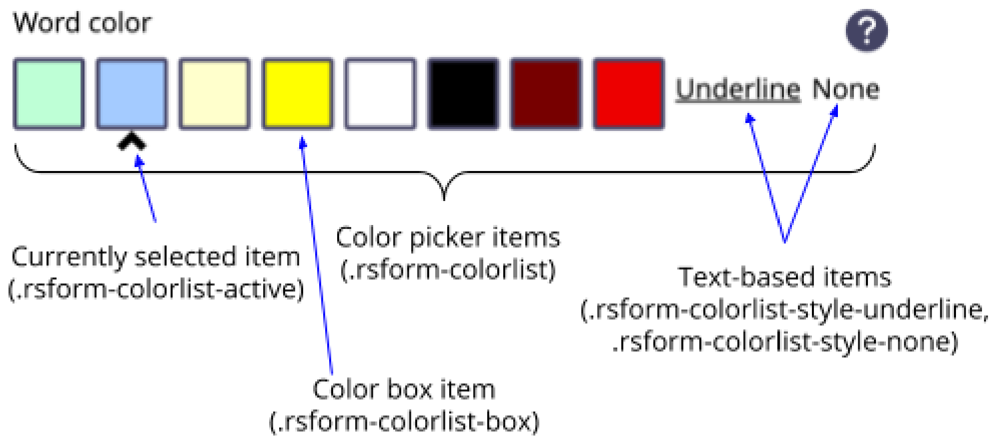
Information Box

The Settings lightbox has built-in help texts that expand when the user clicks on the help buttons.


Highlighting
Text highlighting
On
Off
Enables synchronized highlighting of the text that is currently being read.
Help button (.rsform-info-toggle)
Information content box (.rsform-info-container)

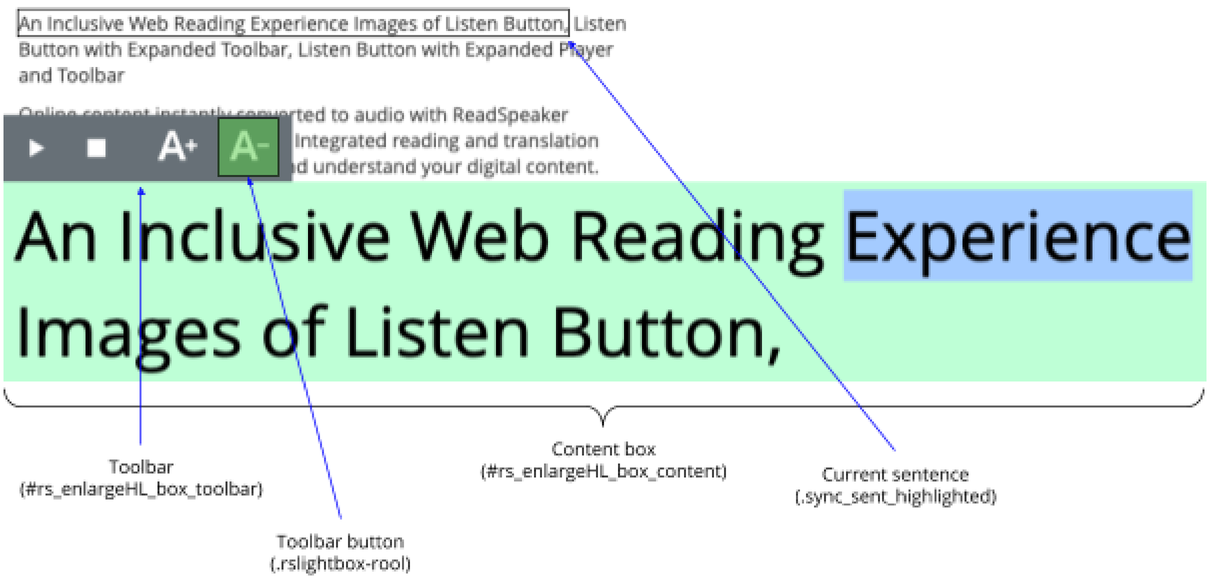
Color Picker

These are used for selecting color or style for fonts and background.


Word color
Underline
None
Currently selected item (.rsform-colorlist-active)
Color picker items (.rsform-colorlist)
Text-based items (.rsform-colorlist-style-underline, .rsform-colorlist-style-none)
Color box item (.rsform-colorlist-box)

Enlarge Text Tool

The Enlarge Text tool can be used to make the currently read sentence stand out more, so it is easier to read.


An Inclusive Web Reading Experience Images of Listen Button, Listen Button with Expanded Toolbar, Listen Button with Expanded Player and Toolbar

Online content instantly converted to audio with ReadSpeaker
Integrated reading and translation
d understand your digital content.

An Inclusive Web Reading Experience Images of Listen Button,

The colors of the enlarged text content box are the same as the user's selected word and sentence highlighting colors.

There is also a toolbar for basic playback control and font size adjustment.

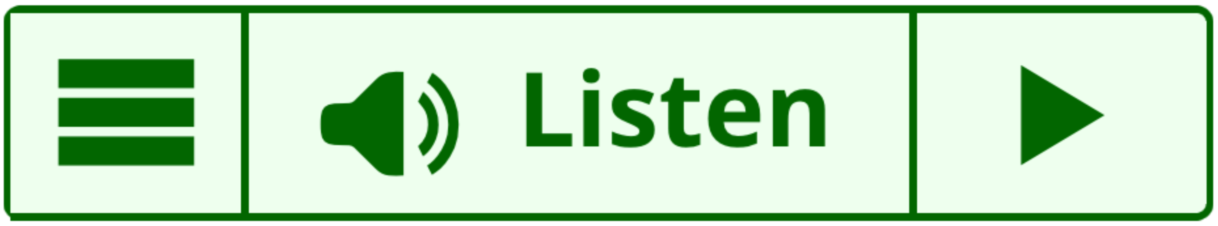
While the Enlarge Text tool is active, the currently read sentence will be marked with a rectangular border on the page.

Building Your Skin

Customizing the Default Skin

If you are generally happy with the way the default player looks but want to change some elements of it, you are best off using the default skin as a base and only defining rules for the details that you want to alter.

Let's make a simple skin based on the default skin. We are just going to change the colors to make it a bit greener. We write in LESS for brevity. This is what we are aiming for:



First of all, we are going to define the colors we will be using:

```
@foreground: #060;
@background: #efe;
@foreground-hover: #070;
@background-hover: #fff;
```

To ensure that our new rules get the highest specificity, we are going to add the name of the skin, GreenSkin. The class name will be added to the webReader container element automatically when we use the skin parameter.

```
.rsbtn {
  &.GreenSkin {
    /* CSS Rules go here */
  }
}
```

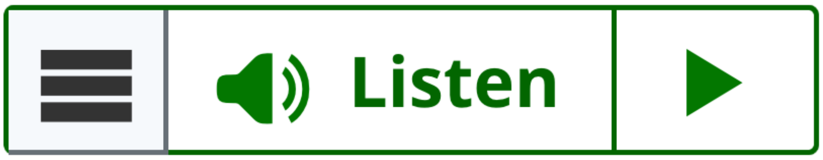
Let's start by changing the color of the button's foreground, background and border.

```
.rsbtn_play {
  background: @background;
  border-color: @foreground;

  .rsbtn_left {
    .rsbtn_text {
      span {
        color: @foreground; /* Listen label */
      }

      &::before {
        color: @foreground; /* Speaker icon */
      }
    }
  }

  .rsbtn_right {
    border-color: @foreground;

    &::before {
      color: @foreground; /* Play icon */
    }
  }
}
```

Since we want a rollover effect, we will also add some rules that kick in when the user hovers the mouse pointer over the button.

```
&:hover {
  background: @background-hover;

  .rsbtn_left {
    .rsbtn_text {
      &::before {
        color: @foreground-hover; /* Speaker icon */
      }
      .rsbtn_label {
        color: @foreground-hover; /* Listen label */
      }
    }
  }

  .rsbtn_right {
    &::before {
      color: @foreground-hover; /* Play icon */
    }
  }
}
```

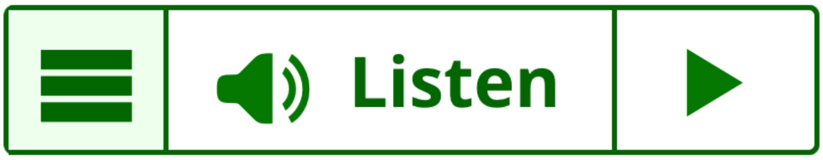
The Listen button now looks like this when you hover the mouse pointer over it:



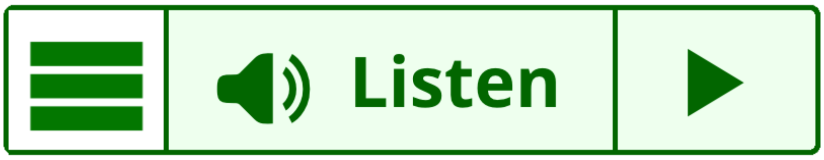
The last thing we are going to change is the menu button, which needs a little more green. We will add styling for the rollover effect at the same time.

```
.rsbtn_tooltoggle {
  background: @background;
  border-color: @foreground;

  .rsicn {
    color: @foreground;
  }
}

&.zoom-tooltoggler {
  .rsbtn_tooltoggle,
  .rsbtn_tooltoggle:hover {
    background: @background-hover;
    border-color: @foreground;

    .rsicn {
      color: @foreground-hover;
    }
  }
}
```

We now have a green button with a nice rollover effect.



The menu button now also has a rollover effect that highlights it when the mouse pointer hovers over it.



Here is the entire code:

```
@foreground: #060;
@background: #efe;
@foreground-hover: #070;
@background-hover: #fff;

.rsbtn {
  &.greenskin {

    .rsbtn_play {
      background: @background;
      border-color: @foreground;

      .rsbtn_left {
        .rsbtn_text {
          span {
            color: @foreground; /* Listen label */
          }

          &::before {
            color: @foreground; /* Speaker icon */
          }
        }
      }

      .rsbtn_right {
        border-color: @foreground;

      &::before {
        color: @foreground; /* Play icon */
      }
    }

    &:hover {
      background: @background-hover;

        .rsbtn_left {
          .rsbtn_text {
            &::before {
              color: @foreground-hover; /* Speaker icon */
            }

            .rsbtn_label {
              color: @foreground-hover; /* Listen label */
            }
          }
        }

        .rsbtn_right {
          &::before {
            color: @foreground-hover; /* Play icon */
          }
        }

      }
    }

    .rsbtn_tooltoggle {
      background: @background;
      border-color: @foreground;

      .rsicn {
        color: @foreground;
      }
    }

    &.zoom-tooltoggler {
      .rsbtn_tooltoggle,
      .rsbtn_tooltoggle:hover {
        background: @background-hover;
        border-color: @foreground;

        .rsicn {
          color: @foreground-hover;
        }
      }
    }
  }
}
```

NOTE! So far we have only changed the Listen button itself, but you will need to add more CSS in order to alter all aspects of webReader's user interface.

Your skin can contain Javascript code as well as CSS styling. Any code added to your skin's Javascript file will be executed when webReader has loaded.

Let's try printing some text to the developer console.

In our GreenSkin.js file we add the following code:

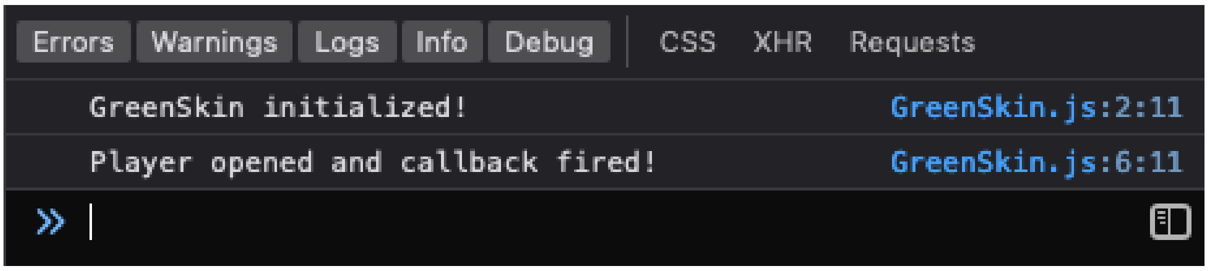
```
ReadSpeaker.q(function() {
  console.log('GreenSkin initialized!');
});
```

```
ReadSpeaker.pub.Config.item('cb.ui.open', function() {
  console.log('Player opened and callback fired!');
});
```

The `ReadSpeaker.q()` function will run as soon as webReader has finished loading and the page DOM is in place.

The second part of the code sets a player callback using webReader's Configuration API. Read more on the webReader Developer website: https://wrdev.readspeaker.com/configuration-api.

With the code uploaded to the server, this is what the console will look like after clicking the Listen button:



```
GreenSkin initialized!                GreenSkin.js:2:11
Player opened and callback fired!     GreenSkin.js:6:11
```

Read more about the available callback functions here: https://wrdev.readspeaker.com/configuration-api/callback.

Building from Scratch

If you choose to build a skin from scratch you will have to make sure that every part of webReader and its components are covered by CSS styles. This will require quite a bit of work, but will at the same time offer almost endless customization possibilities.

We will assume that if you choose to build your own skin from scratch, you have good knowledge of CSS and programming in general. Therefore, we will not describe the required steps in detail.

Disabling the Default Skin

In order to start with a blank slate, you should disable the default skin. This is done by adding the script parameter `noDefaultSkin=1` to the webReader URL:

```
<script src="//domain.com/webReader.js?[...]&amp;noDefaultSkin=1"></script>
```

NOTE! If you choose to build your skin from scratch, but not disable the default skin, you will have to add the skin name in all CSS selectors, in order to ensure the highest specificity:

For example, if you want to target the menu button, `.rsbtn_tooltoggle`, the CSS rule should include:

```
.rsbtn.MyWebReaderSkin .rsbtn_tooltoggle { [...] }
```

NOTE! The lightbox uses a separate webReader player for reading its content. It may be a good idea to include a CSS rule that hides the player:

```
#rslightbox_contentcontainer #rs_lightboxplayer {
      display: none;
}
```

Conclusion

With this you should now have all the tools required in order to create and design your own customized webReader skin.

If you come across any problems while creating your skin or if you have any questions, please reach out to the ReadSpeaker support team at support@readspeaker.com.

Don't forget to check out our ReadSpeaker webReader developer website on https://wrdev.readspeaker.com.