How to Optimize The Display of Long Texts in Oracle Digital Assistant Web Messenger

Asaf Lev, April 2020
Updated, May 2020

A good practice for messages in a bot conversation is to keep them short. Messenger have a limited area for displaying messages, so longer text responses cause the user to scroll as they read. Often also the full message is not needed for a user to understand and continue.

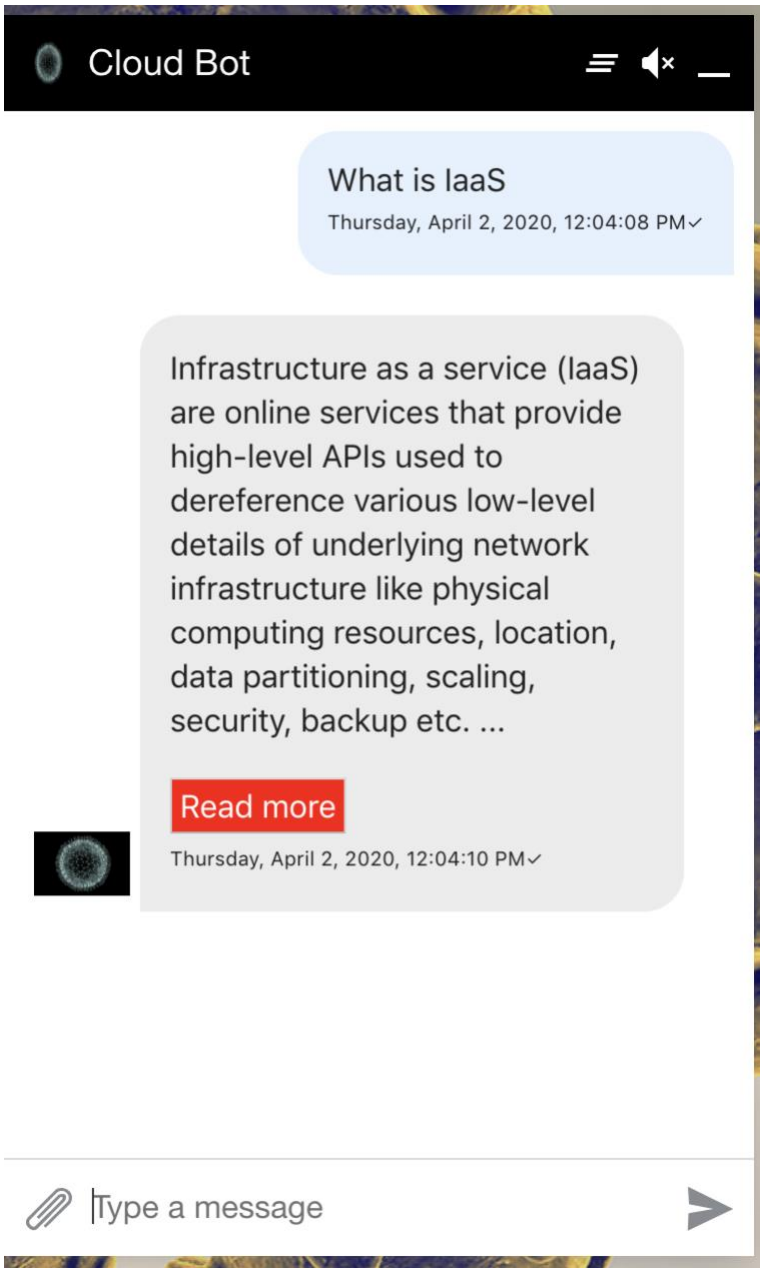
This article explains how you can optimize longer bot messages displayed in the Oracle Web Messenger client to only show the message in full if the user clicks on a link or button.

Unlike another article[1] written about this topic and published on TechExchange that explains how to send long messages in small chunks with a delay in between, this article demonstrates a client side JavaScript solution. The solution in this article dynamically identifies paragraphs in a long text message to then hide all paragraphs except the first from the initial message.

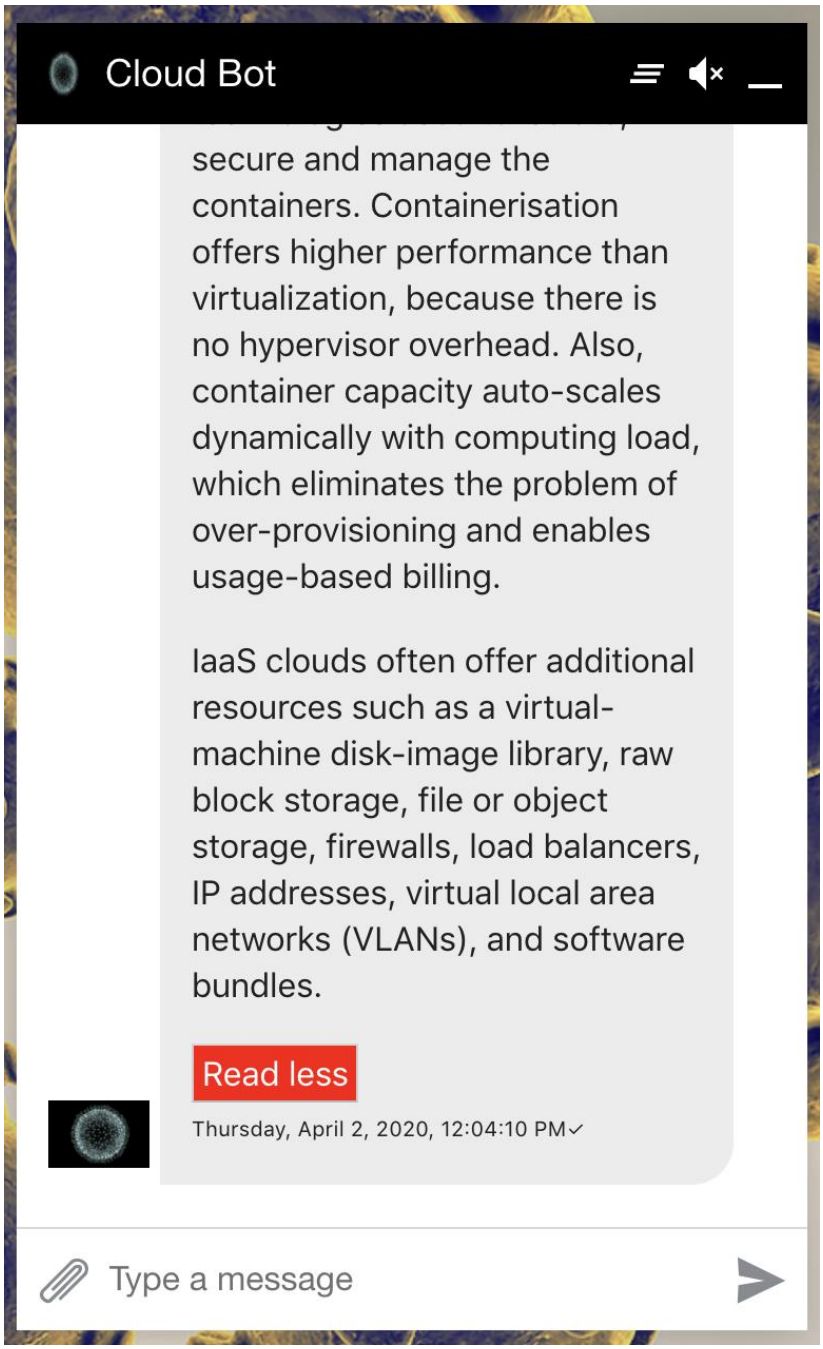
1 https://blogs.oracle.com/mobile/techexchange:-engage-users-by-splitting-long-messages-into-short-ones-no-more-too-long

Final Result

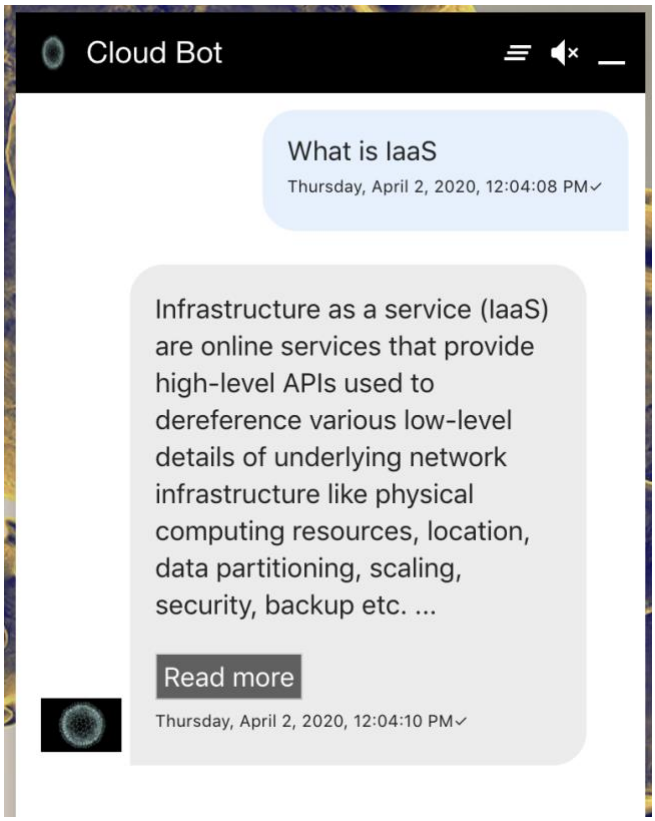
The images below show the visual result of the solution explained in this article. So assuming you setup your skill with answer intents (QnA), the response to a question might be log. The image below shows the message sent by the user as well as the partial answer.



Pressing the **Read more** button then discloses the remaining of the message along with a button to again hide the extended text information.



To indicate that the user read the initially hidden part of the message, after hiding the extended information text, the **Read more** button is shown in gray.



Implementation

This article is focused on the implementation of the hiding and showing of paragraphs in a long text message. It assumes that you know about using the Oracle Web Messenger and how to create an Oracle skill.

IF YOU DON'T MEET THE PRE-REQUISITES, I SUGGEST FOLLOWING THIS TUTORIAL FIRST:
HTTPS://DOCS.ORACLE.COM/EN/CLOUD/PAAS/DIGITAL-ASSISTANT/TUTORIAL-WEB-SDK/INDEX.HTML

The solution explained in this article relies on a delegate object that you can configure on the Web SDK. The delegate object allows code to interact with user messages and bot responses before messages are sent and responses get displayed.

Delegate object example for **Web SDK version < 20.5.1**

```
//delegate object

var delegate = {
  beforeDisplay: function (msg){
     console.log(msg);
```

```
    if (msg.type == "text"){
      msg.text = splitParagraph(msg.text);
      console.log(msg.text);
    }
    return msg;
  },
  beforeSend: function(msg){ return msg;},
  beforePostbackSend: function(msg){return msg;}
};
```

Delegate object example for **Web SDK version >= 20.5.1**

```
//delegate object
var delegate = {
  beforeDisplay: function (msg){
    console.log(msg.messagePayload);
    if (msg. messagePayload .type == "text"){
      msg. messagePayload.text = splitParagraph(msg.messagePayload.text);
      console.log(msg.messagePayload.text);
    }
    return msg;
  },
  beforeSend: function(msg){ return msg;},
  beforePostbackSend: function(msg){return msg;}
};
```

The delegate object uses the *beforeDisplay* callback function to print the bot message to the browser console and then, if the message is of type text, calls a custom function to split it by the first paragraph. The console log statement is good for development to trace what is going on but should be removed in production. All code resides in the HTML document that initiates the Web SDK. If you run this with the sample provided in the Web SDK download, then the file name is index.html.

The code below initiates the Web SDK and associates the delegate object to it. The code itself is what the Web SDK documents in its user guide, so the only change was to add the delegate.

```
function initSdk(name) {
  // Default name is Bots
  if (!name) {
    name = 'Bots';
    }
    setTimeout(() => {
      window[name] = new WebSDK(settings, generateToken);
      window[name].connect();
```

```
        Bots.setDelegate(delegate);
        init();
      });
    }
```

The code below shows the JavaScript logic added to the same HTML document to handle the paragraph split.

```
function showHideParagraphs(lidx) {
  var dots = document.getElementById("dots"+lidx);
  var moreText = document.getElementById("more"+lidx);
  var btnText = document.getElementById("myBtn"+lidx);

  if (dots.style.display === "none") {
     dots.style.display = "inline";
     btnText.innerHTML = "Read more";
     moreText.style.display = "none";

   } else {
     dots.style.display = "none";
     btnText.innerHTML = "Read less";
     moreText.style.display = "inline";
   }
}

  var gidx = 0;

  function splitParagraph(txt){
    var paragraphs = txt.split("\n\n");
    console.log(paragraphs);
    if (paragraphs.length > 1)
    {
      var html = "<p>"+paragraphs[0]+'<span id="dots'+gidx+'">...</span></p><span
      id="more'+gidx+'"class="more">';

      for (var idx = 1; idx < paragraphs.length; idx++){
        html += "<p>"+paragraphs[idx]+ "</p>";
      }

      html += '</span><button class="readMore"
           onclick="showHideParagraphs('+gidx+')" id="myBtn'+gidx+'">Read
           more</button>';

      gidx++;

      return html;

     }

    else
      return txt;

  }
```

The code identifies a paragraph ("\n\n") and split the text accordingly. It then adds three dots to indicate that there is more text to display and a button for the user to unhide the remaining text.

The *showHideParagraph* function is referenced from the button's *onClick* event and dynamically switches between showing or hiding the additional content. It also uses CSS to show and hide buttons.

The following CSS styles needs to be added to the HTML document to style the "Read more" and "Read less" button. The CSS includes a *display:none* style to show or hide buttons according to the state of the long text display.

```
<style>
  .readMore
  {
    color: #fff!important;
    background-color: red;
    font-size: 12pt;
    padding: 4px;
    outline: 0;
    -moz-outline: 0;
    border: 0;
  }
 .readMore:hover
 {
    color: #fff!important;
    background-color: #606060!important;
 }
.more
 {
    display: none;
 }
</style>
```