Simple draw.io embedding walk-through

Created by David Benson, last modified on May 09, 2016

Embedding draw.io as an application within another app (where you're storing the diagram data in the host app) takes around 15 minutes to get a basic example running.

Taking an example Stack Overflow page as a template, we've created our own, slightly altered version of the question. Double click on the diagram to open draw.io in a new window, make some diagram edits then click save to save the changed diagram back to the page.

You can repeat the process to make future edits to the diagram. Let's take a look at both the flow of this process and the code in the page that allows this to happen. If you view the page source you'll see on line 1695:

```
<img class="drawio" style="cursor:default;" src="....
```

The image itself is placed in the page as a base64 encoded dataURI. We're doing this because there's no actual back-end in this example (refresh and you will lose your changes), we've nowhere to write an image to persist it.

The diagram data itself is embedded within the compressed text section of the PNG. When you double click on the image we open a new window and load draw.io in embedded mode in that window:

Collapse source

```
<script>
 // Edits an image with drawio class on double click
 document.addEventListener('dblclick', function(evt)
 {
   var url = 'https://www.draw.io/?embed=1&ui=atlas&spin=1&modified=unsavedChanges&proto=json';
   var source = evt.srcElement || evt.target;

   if (source.nodeName == 'IMG' && source.className == 'drawio')
   {
     if (source.drawIoWindow == null || source.drawIoWindow.closed)
     {
       // Implements protocol for loading and exporting with embedded XML
       var receive = function(evt)
       {
         if (evt.data.length > 0 && evt.source == source.drawIoWindow)
         {
           var msg = JSON.parse(evt.data);

           // Received if the editor is ready
           if (msg.event == 'init')
           {
             // Sends the data URI with embedded XML to editor
             source.drawIoWindow.postMessage(JSON.stringify
               {action: 'load', xmlpng: source.getAttribute('src')}), '*');
           }
           // Received if the user clicks save
           else if (msg.event == 'save')
           {
             // Sends a request to export the diagram as XML with embedded PNG
             source.drawIoWindow.postMessage(JSON.stringify(
               {action: 'export', format: 'xmlpng', spinKey: 'saving'}), '*');
           }
           // Received if the export request was processed
           else if (msg.event == 'export')
           {
             // Updates the data URI of the image
             source.setAttribute('src', msg.data);
           }

           // Received if the user clicks exit or after export
           if (msg.event == 'exit' || msg.event == 'export')
           {
             // Closes the editor
             window.removeEventListener('message', receive);
             source.drawIoWindow.close();
             source.drawIoWindow = null;
           }
         }
       };

       // Opens the editor
       window.addEventListener('message', receive);
       source.drawIoWindow = window.open(url);
     }
     else
     {
       // Shows existing editor window
       source.drawIoWindow.focus();
     }
   }
 });
</script>
```

The "embed=1" URL parameter tells draw.io to operate in embedded mode and "protocol=json" means we're using the JSON protocol for message passing (always use this mode for now). The windows then use postMessage to communicate the diagram data over. Note that you could open in an iFrame instead of a new window, this might make the flow clearer for the user. In this case you would create an iFrame at the start of the function:

```
var iframe = document.createElement('iframe');
iframe.setAttribute('frameborder', '0');

var close = function()
{
    window.removeEventListener('message', receive);
    document.body.removeChild(iframe);
};
```

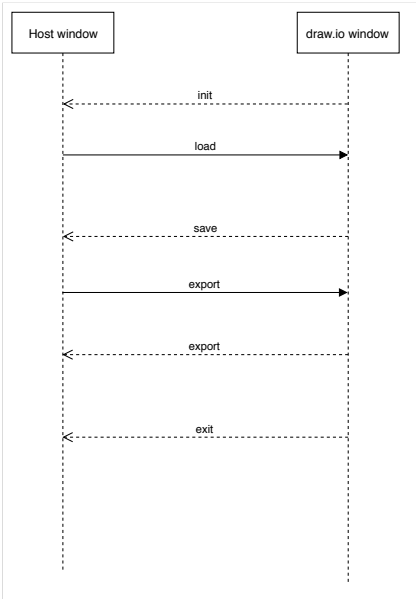
And append the iFrame to the window at the end of the function, after starting to listen for messages:

```
window.addEventListener('message', receive);
iframe.setAttribute('src', editor);
document.body.appendChild(iframe);
```

You might also want to make the iFrame completely cover the initial page via CSS:

```
iframe {
        border:0;
        position:fixed;
        top:0;
        left:0;
        right:0;
        bottom:0;
        width:100%;
        height:100%
    }
```

When the draw.io application is loaded in the new window/iFrame it's just the static application, there's no data. We use the message passing protocol to load in the data to draw.io, after receiving the "init" message from draw.io to tell us that it is ready.



The user then edits the diagram in the editor and clicks save when finished. The reason for additional messages after "save" is we provide granularity for requesting another format, in this case by asking for the PNG+XML format via an "export". In our case we also exit the editor when the export is complete, but you could leave it open and wait for the user to explicitly "exit" the editor before closing it.

The XML embedded PNG as a base64 dataURI is saved back on top of the src attribute of the image and this results in the page being updated with the new image.

Lastly, it's important to distinguish between the static draw.io application and the data flow in embedded mode. draw.io is loaded as a static application in the draw.io window, but the diagram data is passed entirely client-side between windows, it's never sent back to, or sourced from, the draw.io application server. This means you control and store your data, just using draw.io to provide diagramming functionality on that data.

No labels