QX.js Scripting Guide

QUARKXPRESS 2018

Contents

Overview

JavaScript in QuarkXPress

QX.js is a JavaScript development platform that lets you extend and automate QuarkXPress by providing programmatic access to various application commands and to the Document Object Model (DOM) of Layouts within QuarkXPress Projects. Being the most ubiquitous programming language in the world, as well being the lingua franca of the Web, JavaScript integration opens up amazing automation capabilities for anybody who is familiar with this language.

QX.js provides JavaScript objects such as app and fs, and classes such as Layout, Project, and PrintLayoutOptions that correspond to the application's object model. You can interact with these objects by synchronously reading their properties and calling methods (synchronously or asynchronously) to perform operations on them.

The platform is built on top of Chromium Embedded Framework (CEF) that allows QuarkXPress to apply the power of the Blink web browser engine and the V8 JavaScript engine - the same web browser engine and JavaScript engine used by Google Chrome - to the scripting interface of the application as well as its Document Object Model . V8 is Google's open source high-performance JavaScript engine, written in C++. It implements ECMAScript as specified in ECMA-262, and runs on Windows 7 or later and macOS 10.10+.

QXML DOM

The QXML DOM (QuarkXPress Markup Language Document Object Model) is the JavaScript programming interface for accessing a QuarkXPress layout's object model. QXML DOM is based on the *WHATWG HTML Living Standard Custom Elements specification*, which is a part of the W3C DOM specification. Every QXML DOM element is a custom HTML5 element that represents a corresponding QuarkXPress layout object with a custom tag name along with its properties and styling. As it is based on the W3C HTML standard, you can expect to be immediately productive with scripting the QXML DOM if you are already familiar with scripting the HTML DOM. Like HTML DOM elements, all QXML custom elements are styled via custom CSS properties based on the W3C CSS Specification. The *HTML 'style' attribute* is used to specify the inline CSS properties to style QXML custom elements.

Getting Started

Introduction

QuarkXPress application is built upon robust extensible architecture framework and allows its system integrators to extend its functionality as per their customers need. You can do this by writing a C++ XTension using the C++ XDK of the QuarkXPress application. You can develop a QuarkXPress XTension only if you are familiar to these technologies and development tools like XCode on Mac or Microsoft Visual Studio on windows platform. Java scripting support in QuarkXPress 2018 (i.e. qx.js) allows its system integrators as well as the direct users to extend the functionality of application without writing a C++ XTension. It is not only the learning of aging technologies like C++ and development tools in the modern era, you also have to pay the license cost for development tools. Also, you must have a working knowledge of programming for the platform for which you are developing the XTensions.

You can do almost everything through JavaScript except adding the fundamental capabilities into the software. To do such type of extensibility, you need to write a C++ XTensions using it's C++ XDK. Through JavaScript, you can create projects, layout and boxes etc. You can modify its geometry and content using QXML DOM APIs. You can import text or graphic type of content into layout boxes and format the content. Any action that can change the layout or 'its content can be scripted. There even are a few things you can do in JavaScript which are fundamentally missing in the software or that you cannot do using the application's user interface.

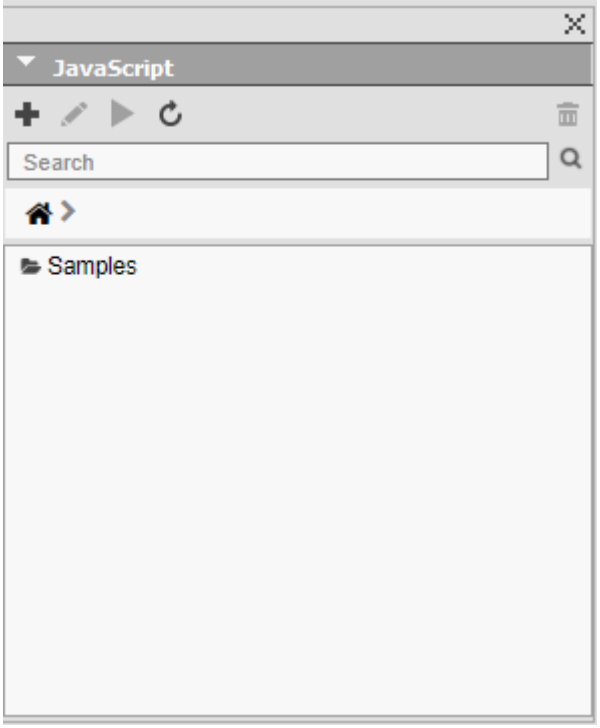
You can create and display dialog boxes for simple user interactions like showing alert, confirmation or getting input from user. Very soon, you can create dialogs, palette and menus through JavaScript. You can not only communicate with file system for reading and writing text files using the JavaScript but communicate with other applications also.

Java Scripting can do everything from very small tasks (like changing the background color of the box) to providing complete features. You can invoke one JavaScript into another JavaScript that allows you to create re-usable scripts. You can start with simple scripts that do only one thing and move on to scripts that automate your entire publishing workflow.

Working with the JavaScript palette

Open the **JavaScript** palette by choosing **Window** > **JavaScript**.

JavaScript Palette



Use the **JavaScript** palette to perform the following functions:

- Add a new JavaScript (✚).
- Edit the selected JavaScript in the registered javascript code editor on your machine.
- Run the selected JavaScript in the palette.
- Refresh the palette to see newly installed scripts.
- Delete the script from the palette.
- Search for your script by name.
- Navigate to different folders under the scripts root directory.

Running a script

To run a script, open the **JavaScript** palette (**Window** > **JavaScript**) and select the script to run. Execute the script using the **Run** ▶ button in the palette. You can also execute the script by double clicking it or through context menu.

Once the script begins executing, the following actions are performed:

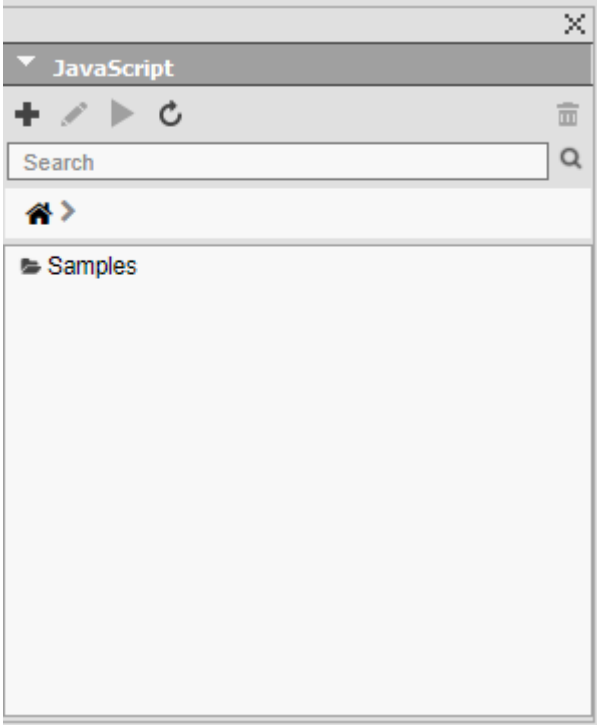
- A compound undo is created to record all of the changes performed through the script allowing you to undo if necessary.
- The **JavaScript** palette displays a progress bar, showing the progress of the running script. The progress bar is terminated when the script execution ends, either successfully or with failure. You can see the status of script execution for each script in the palette.

- The **JavaScript** palette places a green tick beside each successfully completed script and a red tick beside each failed one. You can view a failed scripts error using the tooltip.

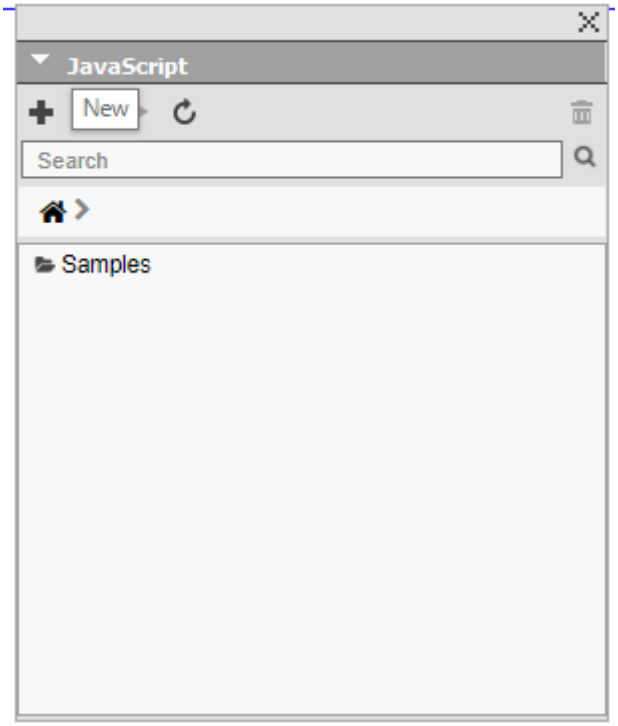
Writing your "HelloWorld" JavaScript

To create a "hello world" JavaScript sample in QuarkXPress, follow these steps:

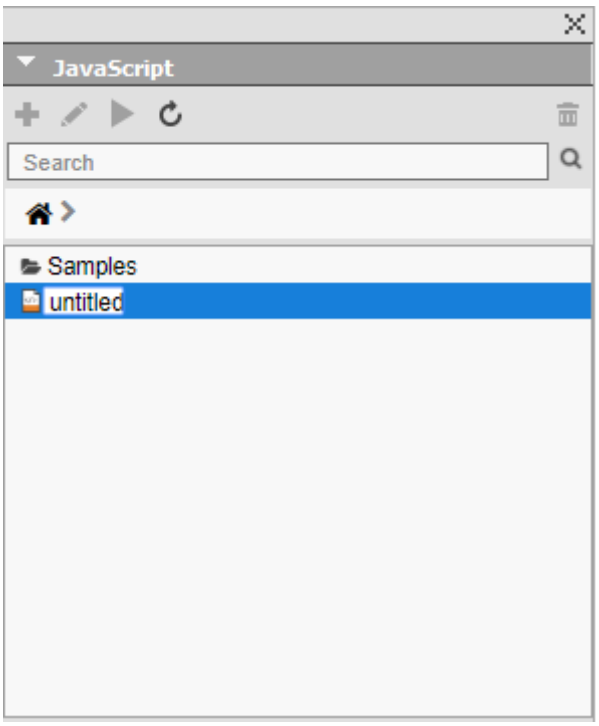
1. Open the **JavaScript** palette (**Window** > **JavaScript**).



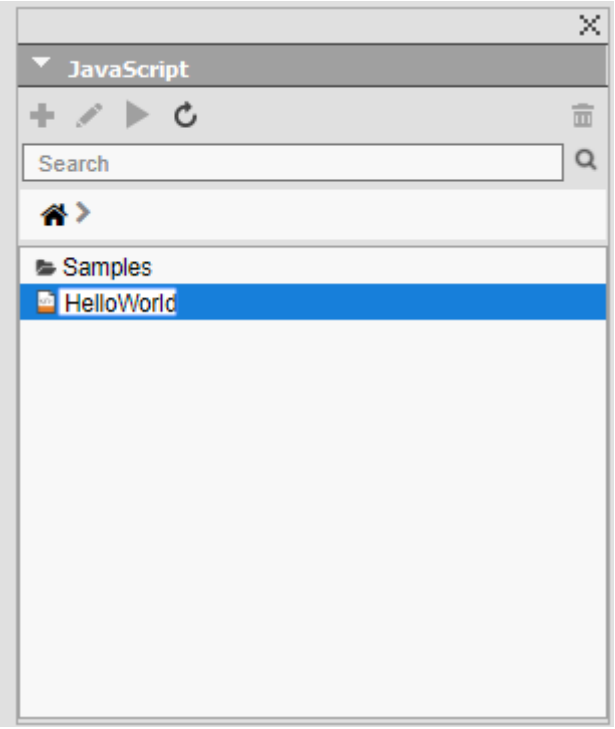
2. Click the icon.



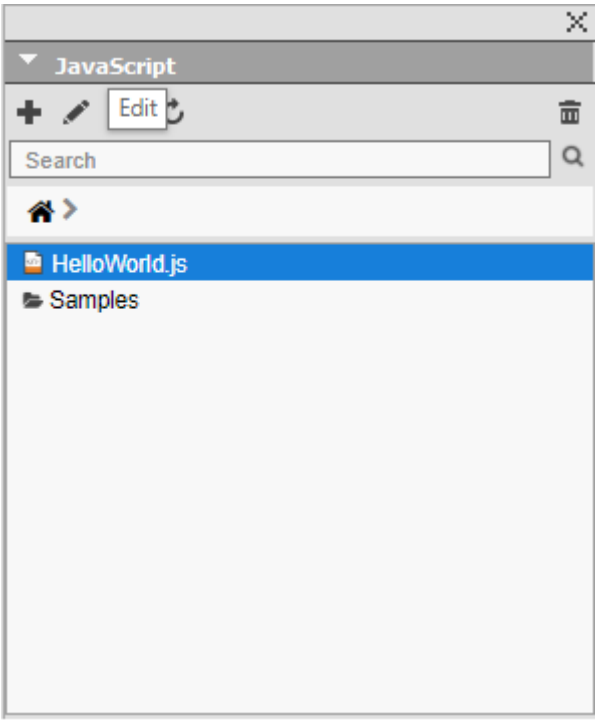
A new entry with the name 'untitled.js' will be added to the list of scripts.



3. Rename the script according to your preference.

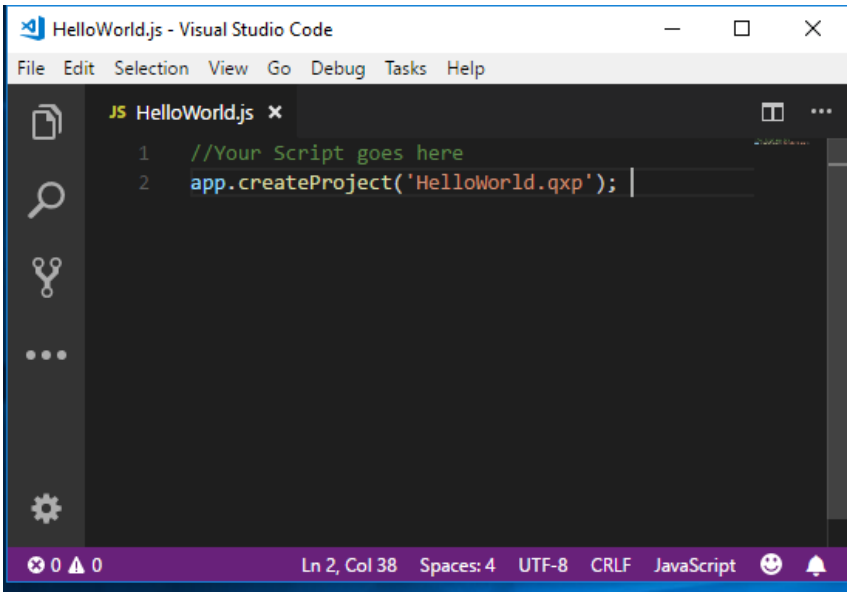


4. Initially the file will be empty. To edit the script, click the **Edit** button. The file will open for editing in the registered editor.

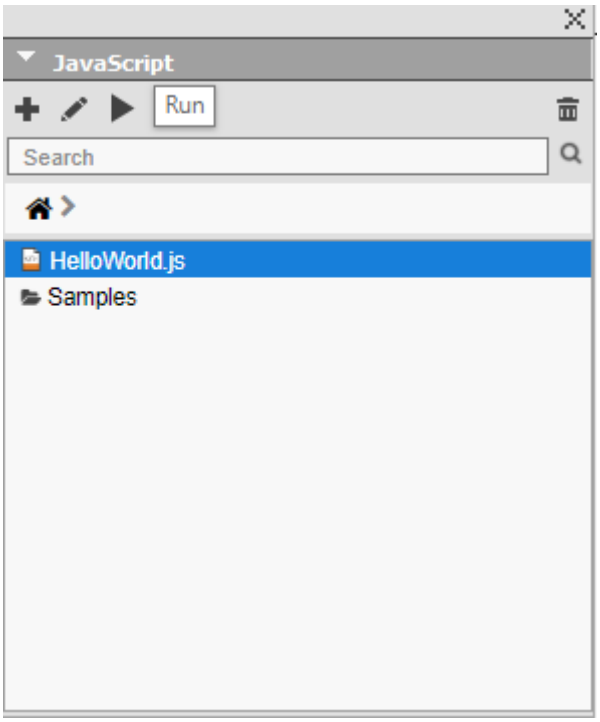


5. Add the following code to the script and save it.

```
app.createProject('HelloWorld.qxp');
```



6. To run the script, double click the script name or select the script from the list and click the **Run** button on the **JavaScript** palette.



Editing the sample scripts

You cannot directly edit the scripts shipped with the application. If you want to edit these scripts, do so by duplicating them.

Duplicate these scripts using the **JavaScript** palette:

1. Navigate to the folder that contains the script you want to duplicate.
2. Select the script or folder, right click, and choose **Copy**.

3. Navigate to your home directory (e.g. `C:\Users\<user>\Documents\Quark\QuarkXPress 2018\js` on Windows and `<Machine-name>\<volume>\Users \<user>\Documents\Quark\QuarkXPress 2018\js` on macOS).
4. Right click in the empty area and **Paste**. The selected scripts or folder will be copied to your home directory.
5. (Optional) If you want to change the name, you can do it through context menu. Navigate to the location (`right click > Open Folder Location` or `Open File Location)`.
6. Change the name and click Update in the **JavaScript** palette.
7. Select the script and click **Edit**. It will open the script in the register code editor. Modify it and save the changes before running it.

Installing new scripts

Installing new scripts in QuarkXPress is simple: put the scripts in the '***js***' folder. The **JavaScript** palette displays the scripts from two different locations on your machine; from the application folder and from the user documents folder.

Place new scripts in the user document folder, which resides here:

Windows®: `\Users\<username>\Documents\Quark\QuarkXPress 2018\js.`

macOS®: `/Users /<username>/Documents/Quark/QuarkXPress 2018/js`.

Introduction to QuarkXPress JavaScript objects

This chapter covers all the JavaScript objects exposed by the QuarkXPress application and XTensions. Use these objects to invoke its feature from the JavaScript environment. There are mainly two global objects (***app*** and ***fs***) exposed for java scripting. Use the '***app***' object to interact with the QuarkXPress application to perform various tasks. The following JavaScript types are exposed by the application:

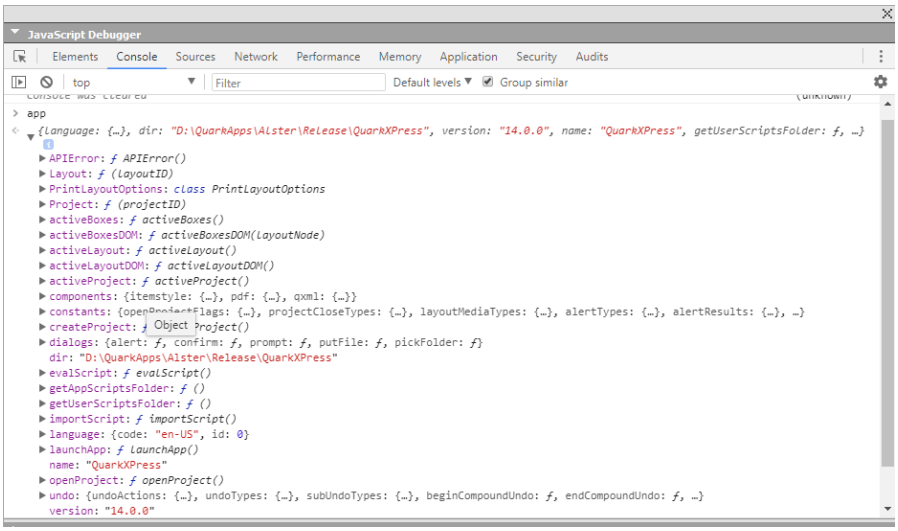
- Classes
- Objects
- Constants

Understanding the QuarkXPress object model

The application is built on an extensible architecture framework and consists of many XTensions and required components (i.e. RCs). So, java scripting support is also provided by these XTensions for its features. We will refer to these XTensions as components in the JavaScript environment. In QuarkXPress 2018, very few components are exposing JavaScript support. Access any component in JavaScript using 'app.components'. Similarly, when you think about the artifacts produced using this application the very first item that comes to our mind is **Project**. The **Project** can have many layouts and each layout can have spreads and pages. Further, these pages can have boxes and finally these boxes hold the content.

Understanding the object model is the key to finding the object you want to work with, and your best guide to Java Scripting is your knowledge of QuarkXPress itself. Quickly see all the JavaScript objects exposed from QuarkXPress using the JavaScript debugger. Open the JavaScript debugger (**Window > JavaScript Debugger**) and select the **Console** tab. Type the following code on console and press Enter.

```
app // Press enter key
```



Working with application (app)

QX.js provides you an ***app*** module to interact with the QuarkXPress application. You already have seen an example of app object usage in the “helloworld” sample.

This chapter shows you how to do the following:

- Retrieve active elements, including:
  - Project.
  - Layout & DOM.
  - Box(s) & DOM.
- Perform basic file management tasks supported by QuarkXPress, including:
  - Creating new project.
  - Opening an existing project.
- Open other type of documents, including:
  - Text files.
  - Java Script files.
  - PDF.
  - Images.
- Perform basic scripting operations, including:
  - Executing any JavaScript.
  - Importing any JavaScript from any other location in the world.

Working with active elements

Most of the application’s basic operations are done on the active element (current project, layout or box). This section shows how to do them using JavaScript.

Getting the current project

Everything in QuarkXPress is stored in the project and all user actions are done on the current project. So, there must be some way to get the current project.

Use the following script to get the current project:

```
let myProj = app.activeProject();
```

Getting current layout and DOM

Use the following script to get the active layout and DOM:

```
let myLayout = app.activeLayout();// Get the active layout
let myDom = myLayout.getDOM();
```

Use the following script to get the DOM of the active layout directly:

```
let myDom = app.activeLayoutDOM();
```

Getting box(s) selection and DOM

Use the following script to get the currently selected box or group box (i.e. multiple selection of boxes) and DOM:

```
let myBoxes = app.activeBoxes();
```

- myBoxes represents the following JSON object:

```
{boxIDs: [], boxUIDs: []}
```

- `boxIDs`: an array of `boxID`s. The `boxid` is temporary and meaningful until the project is opened.
- `boxUIDs`: an array of `boxUID`s. It is the unique ID of the box within a layout.

Use the following script to retrieve the DOM of multiple boxes:

```
let myBoxesDom = app.activeBoxesDOM();
```

- `myBoxesDom` represents the NodeList of <qx-box> type elements.

➡ Please see the QXML section for more details about QXML DOM elements.

Working with QuarkXPress Files

The QuarkXPress application supports working with Projects, Books and Libraries. But in this version, we are only supporting the QuarkXPress Project using JavaScript.

In this section, we will describe how to create and open a project using JavaScript. You already have seen the 'HelloWorld' example to create a simple project.

Use the following JavaScript to create a new project:

```
let myProj = app.createProject();
```

This creates a project with default parameter values (the name of the project will be 'Test1.qxp' and the layout name will be 'Layout 1'). The layout type will always be 'print' and all of the layout properties like orientation, page size, margins will have the default values. If you want to create a project with custom layout properties then you have to construct an object of type 'PrintLayoutOptions'.

The following JavaScript shows how to create a project with different layout dimensions:

```
let plo = new app.PrintLayoutOptions();

plo.autoTextBox = false; //

plo. colCount  = 3;

plo. doubleSided = false;

plo. gutterWidth = "10mm";

plo. pageCount = 10;

plo. pageDirection = false;

plo. pageHeight = "400mm";

plo. pageMargins = ["2mm", "2mm", "2mm", "2mm"];

plo.pageWidth = "300mm";

let myProj = app.createProject("abc.qxp", "print",
app.constants.layoutMediaTypes.PRINT_MEDIA_TYPE, plo);
```

This script will create a project (named "qbc.qxp") and a print layout (named "print") with the given layout dimensions.

Use the following JavaScript to open a project from a file system:

```
let myProj = app.openProject("D:\\QXP\\abc.qxp");
```

When QuarkXPress opens the project, it runs verification steps to check its integrity. If the integrity check fails the user is prompted to take the necessary action. For example, if the fonts used in the project are missing, it will notify the user.

If you do not want any alerts thrown, then you can supply the appropriate flags. For more detail, refer to the API Reference Guide.

Use the following JavaScript to open a project with different flags:

```
// Do not show missing font dialog

let openFlags = app.constants.openProjectFlags.SKIPMISSINGFONT;

let myProj = app.openProject("D:\\QXP\\abc.qxp", openFlags);

// Update all the modified images while opening the project

openFlags = app.constants.openProjectFlags.REIMPORT_ALL_PICS;
```

```
myProj = app.openProject("D:\\QXP\\abc.qxp", openFlags);
```

Working with external applications

Java Scripting in QuarkXPress allow you to view or edit any kind of files through JavaScript. This allows you to edit the referenced content in the native application.

Use the following JavaScript to open a pdf document from a file system:

```
let myPDF = "D:\\PDFS\\ebook.pdf";

app.launchApp(myPDF);
```

This script when executed, will open the 'ebook.pdf' in the registered application.

Working with re-usable JavaScript

Java Scripting in QuarkXPress allow you to re-use existing scripts. You can either import the script into your main script or execute the script to do a specific task. You can create a library of methods which you or other JavaScript developers can reuse. You can also create multiple tasks based JavaScript to create a more meaning workflow using these tasks based JavaScript.

Use the following JavaScript to import a JavaScript.

```
app.importScript("https://code.jquery.com/jquery-3.3.1.min.js");
//This will import the jQuery library in current execution
context.

app.importScript("C:/DATA/myLib.js"); //This will import the
myLib.js from file system
```

In the above example we are importing 'jquery-3.3.1.min.js' from CDN URL and 'myLib.js' from a local file system. You can also execute JavaScript from your main JavaScript.

Use the following JavaScript to execute a JavaScript.

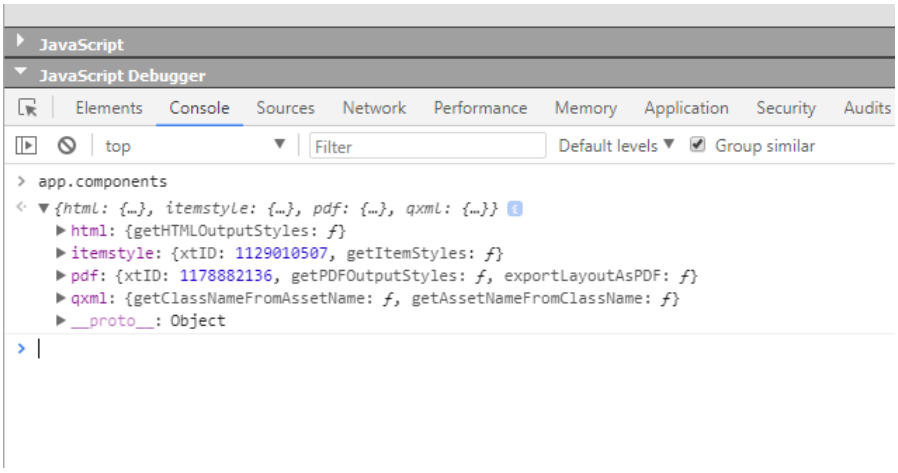
```
let userScriptFolder = app.getUserScriptsFolder();

let myScript = userScriptFolder + '\\' + 'CreateProject.js';

app.evalScript('', myScript);

// Your script goes here...
```

In the above example, we already have a JavaScript which creates a print project with some custom settings. So, there is no need to re-write the script again to create a similar project, just reuse it.

Working with QuarkXPress XTensions

QuarkXPress is built on extensible architecture and comes with a lot of XTensions as part of a package or application bundle. There are many system integrators who are creating XTensions for QuarkXPress. In JavaScript we will reference these XTensions through components terminology, be it a core XTension or a required component.

Use the JavaScript Debugger to see the list of available components:.



Working with PDF

The pdf module represents the PDFFilter XTension of QuarkXPress application in the java scripting world. Use its properties and methods for PDF related operations. In this version very limited functionalities are exposed for JavaScript users.

The following basic operations are allowed through this module:

- Know the number of pages in the PDF document.
- Know the available PDF output styles in the application.
- Export any print layout.

Getting PDF of a layout

You can generate a PDF for any print layout using the ***app.components.pdf*** module. It is recommended that you use the “Layout” module to generate the PDF.

Use the following JavaScript to export a layout as PDF:

```
let myProj = app.openProject(“D:\\QXP\\HelloWorld.qxp”);

let myLayout = app.activeLayout();

let myPDF = “HelloWorld.pdf”;

app.components.pdf.exportLayoutAsPDF(myLayout.layoutID, myPDF);
```

In this script, you are supplying two parameters:

- the layout identifier
- the name of the PDF document to generate.

By default, the PDF is generated silently (i.e. all the possible UI is suppressed). For example, if one or more images are missing it will not warn you and you cannot view the progress for long PDF documents. You can control this behavior by supplying `OutputSuppressUIFlags`.

Use the following script to suppress the progress:

```
app.components.pdf.exportLayoutAsPDF(myLayout1.layoutID, myPDF1,
app.constants.outputSuppressUIFlags.kOutputUI_SuppressOutputWap
|app.constants.outputSuppressUIFlags.kOutputUI_SuppressProgressWa
p);
```

In this script, the PDF is generated based on the default output style. If you want, you can generate the PDF using custom PDF settings. Supply these settings through the desired PDF output style.

Use the following script to generate the PDF using an output style:

```
let outputStyle = "Press - High Quality/High Resolution";

app.components.pdf.exportLayoutAsPDF(myLayout1.layoutID, myPDF1,
app.constants.outputSuppressUIFlags.kOutputUI_SuppressOutputWap,
outputStyle );
```

In this script, we are passing the output style in the last parameter. This parameter can take either the name of the output style or it's ID. So, we can also write this script as:

```
let outputStyle = 3;//Press - High Quality/High Resolution;

app.components.pdf.exportLayoutAsPDF(myLayout1.layoutID, myPDF1,
app.constants.outputSuppressUIFlags.kOutputUI_SuppressOutputWap,
outputStyle );
```

Find the number of pages in the document

Use the following script to reveal the number of pages in the pdf document:

```
let pages = pdf.getPDFNumPages("D:\\QXP\\Helloworld.pdf");

console.log("Number of pages: " + pages);
```

Get a list of PDF output styles

Output styles are assets and may be multiple types (PDF, EPUB, HTML etc). You can get all these output styles using the **Project** class, but getting a pecific type of output style is only possible through its component (i.e. creator).

Use the following script to get list of PDF output styles:

```
let styles = app.components.pdf.getPDFOutputStyles();

styles.assets.forEach(function(asset) {

  console.log("name:" + asset.name);console.log("id:" +
asset.id);
```

```
});
```

In this script the method returns a JSON object (i.e. {type, assets[{}]}). The “type” represents the kind of asset and assets represent an array of objects. Each asset object consists of two values (**name** and **id**).

```
<· ▼{type: 7, assets: Array(10)} ℹ
   ▼assets: Array(10)
     ▶0: {id: 2, name: "Default PDF Output Style"}
     ▶1: {id: 3, name: "Press - High Quality/High Resolution"}
     ▶2: {id: 4, name: "Print - Medium Quality/Medium Resolution"}
     ▶3: {id: 5, name: "Screen - Medium Quality/Low Resolution"}
     ▶4: {id: 6, name: "Screen - Low Quality/Low Resolution"}
     ▶5: {id: 7, name: "PDF/X-1a:2001"}
     ▶6: {id: 8, name: "PDF/X-4"}
     ▶7: {id: 9, name: "PDF/A-2b"}
     ▶8: {id: 10, name: "PDF/X-4 & PDF/A-2b"}
     ▶9: {id: 11, name: "PDF/X-4 & PDF/A-2u"}
      length: 10
```

Working with Projects and Layouts

The work you do in QuarkXPress is centered around Projects and Layouts. Most of the time you interact with a single layout (i.e. the active layout) but your work is saved on the disk in the form of a Project only. One project can have many layouts of different types. You can layouts of type print, digital or both in a single project. So, the project acts as a container for the layouts as well as all the resources (colors, style sheets, fonts etc.). This section shows you how to perform basic operations on the projects and layouts.

In the JavaScripting environment (i.e. qx.js), a QuarkXPress project is represented by the 'Project' class and the layouts in a project by the 'Layout' class.These are JavaScript classes so you need to create an instance of these classes to access their properties or methods.

You can do the followings using the 'Project' and 'Layout' classes:

- Save or down-save the project
- Close the project
- Discover the location of the project on the disk
- Query the resources available in a project
- Add new layouts to a project
- Delete a layout from the project
- Duplicate a layout in the project
- Discover the unique identity of the layout
- Generate a PDF of a layout

Basic operations using the Project class

In this section we see code snippets for doing basic operations using the Project class.

How to close a project

Use the following script to close a project:

```
let curProj = app.activeProject();
```

```
curProj.closeProject();
```

In this script, we first get the reference to the active project. This is the instance of 'Project' class for the current project in the application. Once you get the reference, you can call any method on this object. If you have edits in the project it will not ask for you to save the project. If you want the user to confirm the saving of the project, you need to provide the closing flag.

Use the following code snippet to ask the user to confirm before closing the project:

```
let curProj = app.activeProject();

curProj.closeProject(app.constants.projectCloseTypes.QUERY_CLOSE_
PROJECT);
```

How to save a project on the disk

Use the following script to save a project:

```
let curProj = app.activeProject();

p.saveProject("D:\\abc.qxp");
```

This script will save the project on Windows OS at `D:\abc.q`xp. All of the `saveProject` method parameters are optional for a project which is already saved on the disk. If it is a new project then you must provide the location where you want to save it.

If you want to save the project in the previous version of the application, use the following code snippet.

```
let curProj = app.activeProject();

p.saveAsProject("D:\\abc.qxp", true);
```

How to add a new layout in the project

Use the following code snippet to add a new layout in the current project:

```
let curProj = app.activeProject();

let plo= new app.PrintLayoutOptions("500mm", "300mm", ["36",
"36", "36", "36"], false, 5, "2mm", true, 6, true);

p.createLayout('FF',
app.constants.layoutMediaTypes.PRINT_MEDIA_TYPE, plo);
```

This script first gets the reference to the current project and then initializes the layout properties for the new layout. The layout properties can be get and set using the `PrintLayoutOptions` class. You can also provide the measurement unit along with the property value. If the measurement unit is missing, QuarkXPress will use the measurement unit from the layout or application defaults.

How to discover resources in the project

All of the resources (a.k.a. assets) are stored at the project level. The following code snippet shows you how to query different kind of resources from the project.

Use the following code snippet to discover the paragraph style sheets used in this project:

```
let myProj = app.activeProject;

myProj.getAssets(app.constants.assetTypes.kAssetParaStyle);
```

This code snippet requests all of the paragraph style sheets. The `getAssets` method takes the type of asset as input. All of the asset types are defined in the `app.constants.assetTypes` namespace. For more details about available asset types, please see the QX.js API Reference.

Basic operations using the Layout class

Most of the operations on a layout will be performed using the QXML DOM based APIs. The QXML DOM is retrieved directly from an 'app' object but if you already reference a layout object, then you can instead get the QXML DOM from that.

```
let myLayout = app.activeLayout();

let myDom = myLayout.getDOM();

// Do the operations using QXML DOM APIs.
```

To retrieve layout properties like page margins or gutter width of any layout, use this object.

```
let myLayout = app.activeLayout();

let plo= myLayout.getPrintLayoutOptions();
```

Use the following code snippet to duplicate a layout:

```
let myLayout = app.activeLayout();

myLayout.duplicate();
```

This code will duplicate a layout in the current project. To duplicate a layout with a custom name, you must provide the name. You can provide the custom layout properties using `PrintLayoutOptions` class but all the properties will not honored.

Another important operation is generating th PDF output of the layout. You do this using the Layout object. This is explained in the *Working with PDF* section.

Working with dialogs

QX.js includes the dialogs module which supports four important types of modal dialog boxes on the same lines of JavaScript. These dialog boxes can be used to raise an alert, get confirmation on any input, receive input from users, and to have a dialog with the custom input field.

Dialog modules also override the default alert, prompt, and confirm dialogs available in the global space of JavaScript.

Alert dialog box

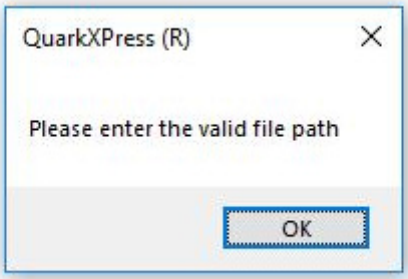
An alert dialog box is generally used to show a warning message to users. For example, if one input field requires that the user enter some text, but the user does not provide the input, then as a part of validation, you can use an alert box to show a warning message.

An alert box can also be used for friendlier messages. An alert box provides only one button, **OK** to proceed.

```
app.dialogs.alert("Please enter the valid file path")
```

or

```
alert("Please enter the valid file path")
```



Confirmation dialog box

A confirmation dialog box is generally used to receive a users consent for an option. It displays a dialog box with two buttons: **OK** and **Cancel**.

If the user clicks the **OK** button, the Window method `confirm()` will return true. If the user clicks the **Cancel** button, then `confirm()` returns false.

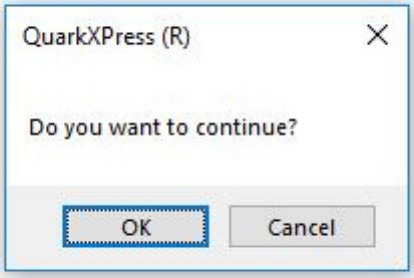
You can use a confirmation dialog box as follows:

```
var retValue = app.dialogs.confirm("Do you want to continue?");
```

or

```
var retValue = confirm("Do you want to continue?");
```

```
retValue ? console.log("User wants to continue! ") :
console.log("User does not want to continue! ");
```



Prompt dialog box

The prompt dialog box is very useful when you want to pop-up a text box to get user input. This enables you to interact with the user. The user is required to fill in the field and then click **OK**.

This dialog box is displayed using a method called `prompt()` which takes two parameters:

- (i) a label you want to display in the text box
- (ii) a default string to display in the text box.

This dialog box has two buttons: **OK** and **Cancel**. If the user clicks the **OK** button, the dialogs module method `prompt()` returns the value the user entered into the text box.

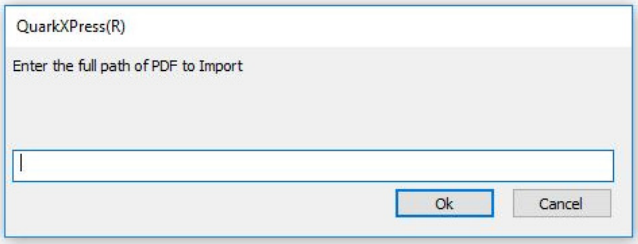
If the user clicks the **Cancel** button, the dialogs module method `prompt()` returns `null`.

```
var pdfFilePath  = app.dialogs.prompt("Enter the full path of PDF
to Import")
```

or

```
pdfFilePath  =  prompt("Enter the full path of PDF to Import")
```

```
pdfFilePath ? console.log(pdfFilePath) : console.log("Error");
```


QuarkXPress(R)
Enter the full path of PDF to Import
Ok
Cancel

Working with the file system

fs module

QX.js includes the **fs** module, which provides an API for interacting with the file system on your computer. **fs** modules APIs are inspired from the Node.js 'fs' module API.

Common uses for the file system module:

- Read files
- Create files
- Update files
- Delete files

Asynchronous and Synchronous

Every method in the **fs** module has synchronous as well as asynchronous forms, except read and write files. Methods take the last parameter as the completion function callback and the first parameter of the callback function as error. It is better to use an asynchronous method instead of the asynchronous method, as the former never blocks the script or application during its execution, whereas the second one does.

File path

Most **fs** operations accept file paths that can only be specified in the form of a string. String form paths are interpreted as UTF-8 character sequences identifying the absolute or relative filename. Relative paths will work only on Windows and they will be resolved relative to the current QuarkXPress location.

Best Practices

- Use `app.dir` properties to access or create a file path relative to QuarkXPress location.
- Use the `app.getUserScriptsFolder()` method to access or create a file relative to the user path JavaScript.
- Use a forward slash (/) as a file separator to create a cross-platform file path.

➡ A file path as an *URL* object or string using the file: protocol is not supported.

The following sections in this chapter provide a set of good examples on major File I/O methods.

Reading Files

Use the `fs.readFileSync()` method to read the physical file synchronously.

Method Signature: `fs.readFileSync(fileName)`

Parameters:

- `filename`: Full path and name of the file as a string.

➡ An asynchronous version of the read file is not available in the current version.

Example

```
let data = fs.readFileSync('C:/DATA/file.txt');
console.log(data);
```

To read file from user documents:

```
let userDocuments = app.getUserScriptsFolder()+'/../../../';
let data = fs.readFileSync(userDocuments+'file.txt');
```

Samples Using this method

- *Mail merge*

Writing Files

Use the `fs.writeFileSync()` method to write data to a file. If a file already exists then the existing content will be overwritten, otherwise a new file is created.

Method Signature: `fs.writeFile(filename, data)`

Parameters:

- `filename`: Full path and name of the file as a string.
- `Data`: The content to be written in the file as a string.

➡ An asynchronous version of the writefilesync() is not available in the current version.

Example

```
let filePath = app.getUserScriptsFolder()+'/../../../file.txt';
fs.writeFileSync(filePath, "file content");
```

This will create a new file in the user documents folder if the file does not exist.

Delete Files

Use the `fs.unlink()` method to delete an existing file.

Method Signature: `fs.unlink(path, callback);`

Parameters:

- `filename`: Full path and name of the file as a string.
- `callback`: A function gets called when file delete operation completes. No arguments other than a possible exception are given to the completion callback.

Example

```
fs.unlink('test.txt', function () {
      if (err) throw err;
     console.log('delete operation complete.');
});
```

Use the `fs.unlinkSync('file path')` method to delete an existing file synchronously.

```
let filePath = app.getUserScriptsFolder()+'/../../../file1.txt';
fs.unlinkSync(filePath);
```

Read Directory

Use the `fs.readdir()` method to read the contents of a directory.

Method Signature: `fs.readdir(path, callback);`

Parameters:

- `filename`: Full path and name of the file as a string.
- `callback`: A function gets called when the reading directory operation completes. The callback takes two arguments (err, files) where files is an array of the names of the files.

Example

Use the following example to read the directory contents:

```
let userDocumentsPath =
app.getUserScriptsFolder()+'/../../../../Documents'
fs.read(userDocumentsPath , function (files) {
      console.log('array of files.', files);
});
```

Use the `fs.readdirSync(path)` method to reads the contents of a directory synchronously.

```
let userDocuments = fs.readdirSync(userDocumentsPath ) ;
console.log(userDocuments);
```

File information

Use the `fs.stat()` method to retrieve information about a file.

Method Signature: `fs.stat(path, callback);`

Parameters:

- `path`: Full path and name of the file as a string.
- `callback`: A function gets called when an operation completes. The callback takes two arguments, (err, stats) where stats is an *fs.Stats* object.

Two useful methods available in the `fs.Stats` class which can be used to check file type:

- `stats.isDirectory()`: Returns true if the file type is a directory.
- `stats.isFile()`: Returns true if the file type is a simple file.

Example

The following example retrieves the file information:

```
fs.stat('test.txt', function (fileStat) {
     console.log(fileStat);
})
```

Use the `fs.statSync(path)` method to retrieve the information about a file synchronously.

```
let filePath = app.getUserScriptsFolder()+'/../../../file1.txt';
let fileStat = fs.stat(filePath);
console.log(fileStat.isFile());
```

File Exist

Use the `fs.exist()` method to test whether or not the given path exists by checking with the file system.

Method Signature: `fs.stat(path, callback);`

Parameters:

- `path`: Full path and name of the file as a string.
- `callback`: A function gets called when the operation completes. The callback has only one boolean parameter.

Use the `fs.existSync(path)` method to test file existence synchronously.

Create Directory

Use the `fs.mkdir()` method to create a directory.

Method Signature: `fs.mkdir(path,[mode], callback)`

Parameters:

- `path`: This is the directory name including path.
- `mode`: This is the directory permission to be set. Defaults to 0777. *Currently not supported*.
- `callback`: This is the callback function. No arguments other than a possible exception are given to the completion callback.

Example

The following example creates the directory in the documents folder of the user.

```
let folderPath = app.getUserScriptsFolder()+'/../../../myFolder';
fs.mkdir(folderPath , function ()  {
    console.log("Directory created successfully!");
} );
```

Use the `fs.mkdirSync(path)` method to create a directory synchronously.

```
let fileStat = fs.stat(folderPath);
console.log(fileStat.isFile());
```

Remove Directory

Use the `fs.rmdir()` method to remove a directory.

Method Signature: `fs.rmdir (path, callback)`

Parameters:

- `path`: This is the directory name including path.
- `callback`: This is the callback function. No arguments other than a possible exception are given to the completion callback.

Example

The following example deletes the 'myFolder' directory in the documents folder of the user:

```
let folderPath = app.getUserScriptsFolder()+'/../../../myFolder';
fs.rmdir(folderPath , function ()  {
    console.log("Directory deleted successfully!");
} );
```

Use the `fs.rmdirSync(path)` method to delete directory synchronously.

Copy Files

Use the `fs.copyFile()` to asynchronously copy files. By default, if the file already exists in the destination, it will be overwritten.

Method Signature: `fs.copyFile(src, dst, flags, callback)`

Parameters:

- `src`: Source path of the file to copy.
- `dst`: Destination path of the copy operation.
- `flags`: An optional integer that specifies the behavior of the copy operation. Default: 0 *Currently not supported.*
- `callback`: This is the callback function. No arguments other than a possible exception are given to the completion callback.

Example

The following example copies the ' file.txt' file from the documents folder to the "C" drive of the user system:

```
let src = app.getUserScriptsFolder()+'/../../../file.txt';
let dest = "C:\\file.txt";
fs.copyFile(src, dest, 0, () => {
     console.log("file copied successfully")
})
```

Use the `fs.copyFileSync(src, dst, flag)` method to copy file synchronously.

Introduction to QuarkXPress Markup Language (QXML)

The QuarkXPress Markup Language (QXML) is the JavaScript programming interface for accessing a QuarkXPress layout's object model. Every QXML DOM element is a custom HTML5 element that represents a corresponding QuarkXPress layout object with a custom tag name along with its properties and styling. Every element starts with namespace `'qx-'`

Get QXML DOM

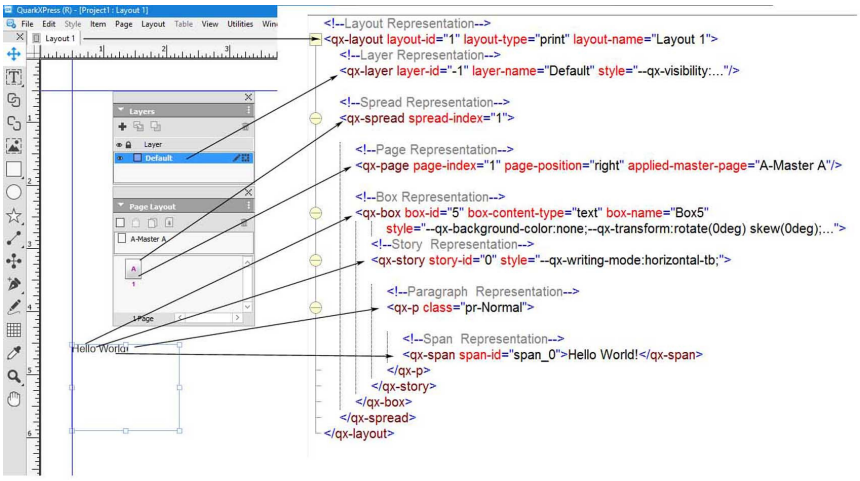
Use the `app.activeLayoutDOM()` or `app.activeBoxesDOM()` API methods to get the XML DOM.

QXML DOM structure

A basic QXML DOM looks like this:

```
<!--Layout Representation-->
<qx-layout layout-id="1" layout-type="print" layout-name="Layout 1">
  <!--Layer Representation-->
  <qx-layer layer-id="-1" layer-name="Default" style="--qx-visibility:..."/>

  <!--Spread Representation-->
  <qx-spread spread-index="1">

    <!--Page Representation-->
    <qx-page page-index="1" page-position="right" applied-master-page="A-Master A"/>

    <!--Box Representation-->
    <qx-box box-id="5" box-content-type="text" box-name="Box5"
        style="--qx-background-color:none;--qx-transform:rotate(0deg) skew(0deg);...">
      <!--Story  Representation-->
      <qx-story story-id="0" style="--qx-writing-mode:horizontal-tb;">

        <!--Paragraph  Representation-->
        <qx-p class="pr-Normal">

          <!--Span  Representation-->
          <qx-span span-id="span_0">Hello World!</qx-span>
        </qx-p>
      </qx-story>
    </qx-box>
  </qx-spread>
</qx-layout>
```

Mapping with QuarkXPress objects:



QXML Elements

Various QXML elements are:

Element Name	Description
`<qx-layout>`	Represents layout
`<qx-layer>`	Represents layer
`<qx-spread>`	Represents spread
`<qx-page>`	Represents page
`<qx-box>`	Represents box
`<qx-story>`	Represents story within a box
`<qx-p>`	Represents paragraph in a story
`<qx-span>`	Represents text within paragraph
`<qx-img>`	Represents an image
`<qx-a>`	Represents hyperlink for both text as well as box
`<qx-bookmark>`	Represents a hyperlink anchor

Style QuarkXPress layout objects

The 'class' attribute in HTML

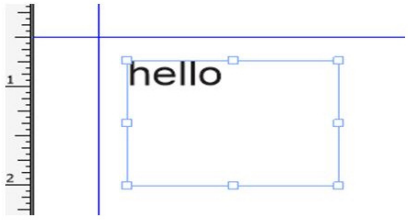
HTML `class` *attribute* specifies one or more CSS class names for an element:

```
<!DOCTYPE html>
<html>
<head>
<style>
h1.intro {
    color: blue;
}
</style>
</head>
<body>

<h1 class="intro">Header 1</h1>

</body>
</html>
```

Header 1

Using the HTML 'class' attribute in QXML

All QuarkXPress resources (Paragraph Style Sheets, Character Style Sheets, Item Styles etc.) can be referred through the HTML `class` attribute:




```
<qx-box>
  <qx-story>
    <qx-p class="pr-Heading">
      <qx-span>hello</qx-span>
    </qx-p>
  </qx-story>
</qx-box>
```

The 'style' attribute in HTML

HTML `style` attribute specifies an inline style for an element. The value of the style attribute contains CSS property-value pairs: `"property: value;"`. You can have as many property value pairs as you want, each separated by semicolon.

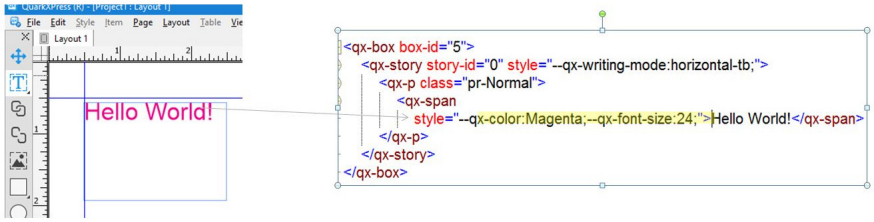
```
<!DOCTYPE html>
<html>
<body>

<h1 style="color:blue;text-
align:center">This is a header</h1>
<p style="color:green">This is a
paragraph.</p>

</body>
</html>
```

This is a header

This is a paragraph.

Using the HTML 'style' attribute in QXML

Similarly, inline styling for each element in QXML is specified by the HTML `style` attribute. We have extended the *`CSSStyleDeclaration`* Object and added custom CSS properties with starting with namespace `'-qx-'`:



QXML DOM manipulations and API calls

Some of the use cases like script to create a box on a layout of an existing project and save the project require mixing of QXML DOM manipulations (create a box on a layout of an existing project) and API calls (save the project). Since all QXML DOM operations are asynchronous in nature, as per *JavaScript even loop concepts* , all the API calls need to placed in a *JavaScript Promise Object*:

```
var layout = app.activeLayoutDOM()
var newBox = document.createElement("qx-box");
newBox.setAttribute("box-content-type", "text");
newBox.style.qxLeft = "0.5in";
newBox.style.qxRight = "4in";
newBox.style.qxTop = "6in";
newBox.style.qxBottom = "8in";
newBox.style.qxPage = 1;
var pages=layout.getElementsByTagName('qx-page')[0];
pages.parentNode.appendChild(newBox);
Promise.resolve().then(function() {
     app.activeProject().saveProject();
})
```

Working with Layouts

QXML for layout

The `<qx-layout>` node contains following attributes:

Attribute	Value	Description	
`layout-name`	text	Name of active layout	
`layout-id`	number	Unique ID of active layout as generated by QuarkXPress	
`layout-type`	`print|digital`	Layout type of active layout	
		`print`	Layout of type Print
		`digital`	Layout of type Digital

Getting the QXML DOM for active layout

The first step in writing a script is to retrieve the QXML DOM. The QXML DOM for the active layout can be retrieved using the API method `app.activeLayoutDOM()`

With root note `<qx-layout>`, `app.activeLayoutDOM()` returns the complete DOM of the active layout including all layers, spreads, pages and boxes along with content of all boxes.

Working with Layers

QXML for layer

Layer is represented by the `<qx-layer>` node. It contains the following attributes:

Attribute	Value	Description
`layer-name`	text	Name of layer
`layer-id`	number	Unique ID of layer as generated by QuarkXPress

Create new layer

To create a new layer:

1. Get the DOM of the active layout.
2. Create a new `<qx-layer>` element.
3. Set the `layer-name` attribute in the `<qx-layer>` element.
4. Append the `<qx-layer>` element as a child of the `<qx-layout>` node.

Sample JavaScript:

```
var layout = app.activeLayoutDOM();
var layer = document.createElement('qx-layer');
layer.setAttribute('layer-name','My Layer');
layout.appendChild(layer);
```

Delete an existing layer

To delete an existing layer:

1. Get the DOM of the active layout.
2. Get the `<qx-layer>` node corresponding to the layer to be deleted.
3. Remove the `<qx-layer>` child node from the `<qx-layout>` node.

Sample JavaScript:

```
var layout = app.activeLayoutDOM();
var layer = layout.querySelector("qx-layer[layer-name='My
Layer']");
layout.removeChild(layer);
```

Hide a layer

To hide a layer:

1. Get the DOM of the active layout.
2. Get the `<qx-layer>` node .
3. Set the `qxVisibility` styling attribute to `hidden.`

```
var layout = app.activeLayoutDOM();
var layer = layout.querySelector("qx-layer[layer-name='My
Layer']");
layer.style.qxVisibility = 'hidden';
```

Lock a layer

To lock a layer:

1. Get the DOM of the active layout.
2. Get the `<qx-layer>` node .
3. Set the `qxLocked` styling attribute to `true.`

```
var layout = app.activeLayoutDOM();
var layer = layout.querySelector("qx-layer[layer-name='My
Layer']");
layer.style.qxLocked = 'true';
```

Suppress output for a layer

To suppress output for a layer:

1. Get the DOM of the active layout.
2. Get the `<qx-layer>` node .
3. Set the `qxSuppressOutput` styling attribute to `true.`

```
var layout = app.activeLayoutDOM();
var layer = layout.querySelector("qx-layer[layer-name='My
Layer']");
layer.style.qxSuppressOutput = 'true';
```

Keep runaround for layer

To remote keep runaround property for a layer:

1. Get the DOM of the active layout.
2. Get the `<qx-layer>` node .
3. Set the `qxKeepRunaround` styling attribute to `false.`

```
var layout = app.activeLayoutDOM();
var layer = layout.querySelector("qx-layer[layer-name='My
Layer']");
layer.style.qxKeepRunaround = 'false';
```

Specify layer color

To specify the color for a layer using RGB values:

1. Get the DOM of the active layout.
2. Get the `<qx-layer>` node .
3. Specify the RGB color value `qxColor` styling attribute.

```
var layout = app.activeLayoutDOM();
var layer = layout.querySelector("qx-layer[layer-name='My
Layer']");
layer.style.qxColor= 'rgb(240,55, 217)';
```

Working with Spreads

QXML for spread

Spread is represented by the `<qx-spread>` node. It contains the following attributes:

Attribute	Value	Description
`spread-index`	number	Index of the spread

Create new spread

To create a new spread:

1. Get the DOM of the active layout.
2. Create a new `<qx-spread>` element.
3. Set the `spread-index` attribute in the `<qx-spread>` element.
4. Append the `<qx-spread>` element as a child of the `<qx-layout>` node.

Sample JavaScript:

```
var newSpread = document.createElement("qx-spread");
newSpread.setAttribute("spread-index", "2");
var page = document.createElement('qx-page');
page.setAttribute("page-index", "2");
page.setAttribute("applied-master-page", "A-Master A");
page.setAttribute("page-position", "right");
newSpread.appendChild(page);
var spread = layout.getElementsByTagName('qx-spread')[0];
spread.parentNode.appendChild(newSpread);
```

Delete existing spread

To delete existing spread:

1. Get the DOM of the active layout.
2. Get the `<qx-spread>` node corresponding to the spread to be deleted.
3. Remove the `<qx-spread>` child node from the `<qx-layout>` node.

Sample JavaScript:

```
var layout=app.activeLayoutDOM();
var spreads=layout.getElementsByTagName('qx-spread')[1];
layout.removeChild(spreads);
```

Working with Pages

QXML for page

Page is represented by the `<qx-page>` node. It contains the following attributes:

Attribute	Value	Description	
`page-index`	number	Index of the page	
`page-position`	`left | right`	Specifies whether a page should be on the left or right of the spine	
		`left`	Specifies that page is on left side of the spine
		`right`	Specifies that page is on right side of the spine
`applied-master-page`	text	Identifies the master-page that is applied to a page	

Create a new page

To create a new page:

1. Get the DOM of the active layout.
2. Create a new `<qx-page>` element.
3. Set the `page-index` attribute in the `<qx-page>` element.
4. Set the `page-position` attribute in the `<qx-page>` element.
5. Set the `applied-master-page` attribute in the `<qx-page>` element.
6. Append the `<qx-page>` element as a child of the `<qx-spread>` node.

Sample JavaScript:

```
var layout = app.activeLayoutDOM();
var spread= layout.getElementsByTagName('qx-spread')[0];
var page = document.createElement('qx-page');
page.setAttribute('page-index','2');
page.setAttribute('page-position','right');
page.setAttribute('applied-master-page','A-Master A');
```

```
spread.appendChild(page);
```

Delete existing page

To delete an existing page:

1. Get the DOM of the active layout.
2. Get the `<qx-page>` node corresponding to the page to be deleted.
3. Remove the `<qx-page>` child node from the `<qx-spread>` node.

Sample JavaScript:

```
var layout = app.activeLayoutDOM();
var spread= layout.getElementsByTagName('qx-spread')[0];
var page = layout.querySelector("qx-page[page-index='2']");
spread.removeChild(page);
```

Working with Boxes

QXML for box

Box is represented by the `<qx-page>` node. It contains the following attributes:

Attribute	Value	Description	
box-id	number	Unique ID of the box as generated by QuarkXPress	
box-name	text	Name of the box	
box-content-type	text| picture	Specifies what exactly does the box hold eg. Pictures, Text	
		text	Specifies that content of box is 'text'.
		picture	Specifies that content of box is 'picture'.
class	text	Name of Item Style applied on box	

Create new box

To create a new box:

1. Get the DOM of the active layout.
2. Create a new `<qx-box>` element.
3. Set the `box-content-type` attribute in the `<qx-box>` element.
4. Set the `qxLeft,qxRight,qxTop and qxBottom` styling properties in the `<qx-box>` element.
5. Set the `qxPage` styling property in the `<qx-box>` element.
6. Append the `<qx-box>` element as a child of the `<qx-spread>` node.

Sample JavaScript:

```
var layout=app.activeLayoutDOM();
var newBox = document.createElement("qx-box");
newBox.setAttribute("box-content-type", "text");
newBox.style.qxLeft = "0.5in";
newBox.style.qxRight = "4in";
```

```
newBox.style.qxTop = "6in";
newBox.style.qxBottom = "8in";
newBox.style.qxPage = 1;
var pages=layout.getElementsByTagName('qx-page')[0];
pages.parentNode.appendChild(newBox);
```

Delete existing box

To delete an existing box:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node corresponding to the spread to be deleted.
3. Remove the `<qx-box>` child node from the `<qx-spread>` node.

Sample JavaScript:

```
var layout = app.activeLayoutDOM();
var spread=layout.getElementsByTagName('qx-spread')[0];
var box = spread.getElementsByTagName('qx-box')[0];
spread.removeChild(box);
```

Apply box borders

Border width

To apply a border width:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBorderWidth` styling property to `'0.1in'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBorderWidth='0.1in';
```

Border style

To apply a border style:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBorderWidth` styling property to `'0.1in'`.
4. Set the `qxBorderStyle` styling property to `'dotted'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBorderWidth='0.1in';
box.style.qxBorderStyle='dotted';
```

Border color

To apply a border color:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .

3. Set the `qxBorderWidth` styling property to `'0.1in'`.
4. Set the `qxBorderColor` styling property to `'yellow'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBorderWidth='0.1in';
box.style.qxBorderColor='yellow';
```

Border shade

To apply a border shade:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBorderWidth` styling property to `'0.1in'`.
4. Set the `qxBorderShade` styling property to `'0.2'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBorderWidth='0.1in';
box.style.qxBorderShade='0.2';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Border opacity

To apply border opacity:

1. Get the DOM of te active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBorderWidth` styling property to `'0.1in'`.
4. Set the `qxBorderOpacity` styling property to `'0.7'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBorderWidth='0.1in';
box.style.qxBorderOpacity='0.7';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Border mix blend mode

To apply a border mix blend mode:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBorderWidth` styling property to `'0.1in'`.
4. Set the `qxBorderMixBlendMode` styling property to `'darken'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBorderWidth='0.1in';
box.style.qxBorderMixBlendMode='darken';
```

Border gap color

To apply a border gap color:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBorderWidth` styling property to `'0.1in'`.
4. Set the `qxBorderStyle` styling property to `'dotted'`.
5. Set the `qxBorderGapColor` styling property to `'cyan'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBorderWidth='0.1in';
box.style.qxBorderStyle='dotted';
box.style.qxBorderGapColor='cyan';
```

Border gap shade

To apply a border gap shade:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBorderWidth` styling property to `'0.1in'`.
4. Set the `qxBorderStyle` styling property to `'dotted'`.
5. Set the `qxBorderGapColor` styling property to `'cyan'`.
6. Set the `qxBorderGapShade` styling property to `'0.6'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBorderWidth='0.1in';
box.style.qxBorderStyle='dotted';
box.style.qxBorderGapColor='cyan';
box.style.qxBorderGapShade='0.6';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Border gap opacity

To apply a border gap opacity:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBorderWidth` styling property to `'0.1in'`.
4. Set the `qxBorderStyle` styling property to `'dotted'`.
5. Set the `qxBorderGapColor` styling property to `'cyan'`.
6. Set the `qxBorderGapOpacity` styling property to `'0.4'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBorderWidth='0.1in';
box.style.qxBorderStyle='dotted';
```

```
box.style.qxBorderGapColor='cyan';
box.style.qxBorderGapOpacity='0.4';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Border gap mix blend mode

To apply a border gap mix blend mode:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBorderWidth` styling property to `'0.1in'`.
4. Set the `qxBorderStyle` styling property to `'dotted'`.
5. Set the `qxBorderGapColor` styling property to `'cyan'`.
6. Set the `qxBorderGapMixBlendMode` styling property to `'lighten'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBorderWidth='0.1in';
box.style.qxBorderStyle='dotted';
box.style.qxBorderGapColor='cyan';
box.style.qxBorderGapMixBlendMode='lighten';
```

Apply background color, shade and opacity

Background color

To apply a background color:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBackgroundColor` styling property to `'cyan'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBackgroundColor='cyan';
```

Background shade

To apply background shade:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBackgroundColor` styling property to `'cyan'`.
4. Set the `qxBackgroundShade` styling property to `'0.8'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBackgroundColor='cyan';
box.style.qxBackgroundShade='0.8';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Background opacity

To apply background opacity:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxBackgroundColor` styling property to `'cyan'`.
4. Set the `qxBackgroundOpacity` styling property to `'0.3'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxBackgroundColor='cyan';
box.style.qxBackgroundOpacity='0.3';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Specify box position

To specify box position:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxTop` styling property to `'3in'`.
4. Set the `qxBottom` styling property to `'8in'`.
5. Set the `qxLeft` styling property to `'3in'`.
6. Set the `qxRight` styling property to `'7in'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxTop='3in';
box.style.qxBottom='8in';
box.style.qxLeft='3in';
box.style.qxRight='7in';
```

Apply runaround

To apply runaround:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxWrapFlow` styling property to `'both'`.
4. Set the `qxWrapMarginTop` styling property to `'5pt'`.
5. Set the `qxWrapMarginBottom` styling property to `'5pt'`.
6. Set the `qxWrapMarginLeft` styling property to `'5pt'`.
7. Set the `qxWrapMarginRight` styling property to `'5pt'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxWrapFlow='both';
box.style.qxWrapMarginTop='5pt';
box.style.qxWrapMarginBottom='5pt';
box.style.qxWrapMarginLeft='5pt';
box.style.qxWrapMarginRight='5pt';
```

Apply drop shadow

Shadow color

To apply shadow color:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxShadowColor` styling property to `'magenta'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxShadowColor='magenta';
```

Shadow shade

To apply shadow shade:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxShadowShade` styling property to `'0.7'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxShadowShade='0.7';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Shadow opacity

To apply shadow opacity:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxShadowOpacity` styling property to `'0.5'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxShadowOpacity='0.5';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Shadow distance

To apply shadow distance:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxShadowDistance` styling property to `'0.9in'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxShadowDistance='0.9in';
```

Shadow angle synchronized

To apply shadow angle synchronized:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxShadowAngleSynchronized` styling property to `'true'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxShadowAngleSynchronized='true';
```

Shadow multiply

To apply shadow multiply:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxShadowMultiply` styling property to `'false'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxShadowMultiply='false';
```

Shadow runaround

To apply shadow runaround:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxShadowRunaround` styling property to `'true'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxShadowRunaround='true';
```

Shadow inherit opacity

To apply shadow inherit opacity:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxShadowInheritOpacity` styling property to `'true'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxShadowInheritOpacity='true';
```

Shadow transform

To apply shadow transform:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxShadowTransform` styling property to `'rotate(20deg) skew(40deg)'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxShadowTransform='rotate(20deg) skew(40deg)';
```

Shadow knockOut

To apply shadow knockOut:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxShadowKnockOut` styling property to `'true'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxShadowKnockOut='true';
```

Shadow filter

To apply shadow filter:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxShadowFilter` styling property to `'blur(0.1in)'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxShadowFilter='blur(0.1in)';
```

Apply angle and skew

To apply angle and skew:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxTransform` styling property to `'rotate(10deg) skew(30deg)'`.

```
var layout = app.activeLayoutDOM();
var box= layout.getElementsByTagName('qx-box')[0];
box.style.qxTransform='rotate(10deg) skew(30deg)';
```

Apply column properties

Column count

To apply column count:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxColumnCount` styling property to `'2'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxColumnCount='2';
```

Column gap

To apply column gap:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxColumnCount` styling property to `'2'`.
4. Set the `qxColumnGap` styling property to `'0.3in'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxColumnCount='2';
boxes.style.qxColumnGap='0.3in';
```

Column rule width

To apply column rule width:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxColumnCount` styling property to `'2'`.
4. Set the `qxColumnRuleWidth` styling property to `'4pt'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxColumnCount='2';
boxes.style.qxColumnRuleWidth='4pt';
```

Column rule style

T apply column rule style:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxColumnCount` styling property to `'2'`.
4. Set the `qxColumnRuleStyle` styling property to `'dotted'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxColumnCount='2';
boxes.style.qxColumnRuleStyle='dotted';
```

Column rule color

To apply column rule color:

1. Get the DOM of the active layout.

2. Get the `<qx-box>` node .
3. Set the `qxColumnCount` styling property to `'2'`.
4. Set the `qxColumnRuleColor` styling property to `'cyan'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxColumnCount='2';
boxes.style.qxColumnRuleColor='cyan';
```

Column rule shade

To apply column rule shade:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxColumnCount` styling property to `'2'`.
4. Set the `qxColumnRuleShade` styling property to `'0.7'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxColumnCount='2';
boxes.style.qxColumnRuleShade='0.7';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Column rule opacity

To apply column rule opacity:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxColumnCount` styling property to `'2'`.
4. Set the `qxColumnRuleShade` styling property to `'0.3'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxColumnCount='2';
boxes.style.qxColumnRuleOpacity='0.3';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Apply padding

Inset text on All edges

To apply Inset Text on All edges:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxPadding` styling property to `'2pt'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxPadding='2pt';
```

Inset text on top,bottom,left,right edges

To apply inset text on top,bottom,left,right edges:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxPaddingTop` styling property to `'5pt'`.
4. Set the `qxPaddingBottom` styling property to `'7pt'`.
5. Set the `qxPaddingLeft` styling property to `'10pt'`.
6. Set the `qxPaddingRight` styling property to `'12pt'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxPaddingTop='5pt';
boxes.style.qxPaddingBottom='7pt';
boxes.style.qxPaddingLeft='10pt';
boxes.style.qxPaddingRight='12pt';
```

Apply mix blend mode

To apply mix blend mode:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxMixBlendMode` styling property to `'darken'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxMixBlendMode='darken';
```

Apply maintain aspect ratio

To apply maintain aspect ratio:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxMaintainAspectRatio` styling property to `'true'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxMaintainAspectRatio='true';
```

Apply suppress output for box

To apply suppress output for box:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .

3. Set the `qxSuppressOutput` styling property to `'true'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxSuppressOutput='true';
```

Apply Text properties

Text transform

To apply text transform:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxTextTransform` styling property to `'rotate(40deg)'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxTextTransform='rotate(40deg)';
```

Text mix blend mode

To apply text mix blend mode:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxTextMixBlendMode` styling property to `'lighten'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxTextMixBlendMode='lighten';
```

Text flip horizontal

To apply text flip horizontal:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxTextFlipHorizontal` styling property to `'true'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxTextFlipHorizontal='true';
```

Text flip vertical

To apply text flip vertical are:-

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxTextFlipVertical` styling property to `'true'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxTextFlipVertical='true';
```

Apply first baseline minimum

To apply first baseline minimum are:-

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxFirstBaselineMinimum` styling property to `'ascent'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxFirstBaselineMinimum='ascent';
```

Apply first baseline offset

To apply first baseline offset are:-

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxFirstBaselineOffset` styling property to `'0.5in'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxFirstBaselineOffset='0.5in';
```

Apply align items

To apply align items:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxAlignItems` styling property to `'justify'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxAlignItems='justify';
```

Apply inter para max space

To apply inter para max space:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxAlignItems` styling property to `'justify'`.
4. Set the `qxInterParaMaxSpace` styling property to `'0.25in'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxAlignItems='justify';
boxes.style.qxInterParaMaxSpace='0.25in';
```

Apply run text around all sides

To apply run text around all sides:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxRunTextAroundAllSides` styling property to `'true'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxRunTextAroundAllSides='true';
```

Working with Images

QXML for image

Image is represented by the `<qx-img>` node. It contains the following attributes:

Attribute	Value	Description	
`src`	text	Source of Image, i.e, the path from where the image is to be taken	
`src#page=`	text	Import particular page of a PDF file	
`src#page&bounding box=`	`mediabox` | `cropbox` | `bleedbox` | `trimbox`	Bounding box of PDF file	
		`mediabox`	Uses the size of the page, not including space for bleeds or registration marks.
		`cropbox`	Uses the size of the page plus space for any specified bleeds and space allotted for registration marks. It equals the size of the layout space from the PDF Boxer XT software standpoint.
		`bleedbox`	Uses the size of the page plus space for bleeds.
		`trimbox`	Uses the size of the page after trimming is done. This option does not include registration marks and is unaffected by any bleeds applied to the layout space when the PDF is created.

Import image in picture box

To import an image in a picture box:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` and `<qx-img>` node.
3. Set the 'src' attribute to 'file:///C:/test.jpg'.

Sample JavaScript:

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].setAttribute('src','file:///C:/test.jpg');
```

Change existing image

To change an existing image:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` and `<qx-img>` node.
3. Change the 'src' attribute to 'file:///C:/new_test.jpg'.

Sample JavaScript:

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].setAttribute('src','file:///C:/new_test.jpg');
```

Apply Offsets

Offset across

To apply offset across:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxOffsetAcross` styling property to `'0.5in'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxOffsetAcross='0.5in';
```

Offset down

To apply offset down:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.

4. Set the `qxOffsetDown` styling property to `'0.5in'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxOffsetDown='0.5in';
```

Apply transform

Scale horizontally

To scale horizontally:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxTransform` styling property to `'scaleX(0.5)'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxTransform='scaleX(0.5)';
```

Scale vertically

To apply scale vertically:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxTransform` styling property to `'scaleY(0.5)'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxTransform='scaleY(0.5)';
```

Rotate

To apply rotate:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxTransform` styling property to `'rotate(30deg)'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxTransform='rotate(30deg)';
```

Skew

To apply skew:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxTransform` styling property to `'skew(40deg)'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxTransform='skew(40deg)';
```

Apply color

To apply color to a monochrome image:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxColor` styling property to `'cyan'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxColor='cyan';
```

Apply shade

To apply shade to a monochrome image:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxShade` styling property to `'0.7'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxShade='0.7';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Apply opacity

To apply opacity to a monochrome image:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxOpacity` styling property to `'0.9'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
```

```
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxOpacity='0.9';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Apply mix blend mode

To apply mix blend mode to an image:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxMixBlendMode` styling property to `'darken'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxMixBlendMode='darken';
```

Apply background color

To apply background color to a monochrome image:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxBackgroundColor` styling property to `'magenta'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxBackgroundColor='magenta';
```

Apply background shade

To apply background shade to a monochrome image:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxBackgroundShade` styling property to `'0.8'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxBackgroundShade='0.8';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Apply background opacity

To apply background opacity to a monochrome image:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Get the `<qx-img>` node.
4. Set the `qxBackgroundOpacity` styling property to `'0.5'`.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var images = boxes[0].getElementsByTagName('qx-img');
images[0].style.qxBackgroundOpacity='0.5';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Flip image vertically

To apply flip image vertically:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxImgFlipVertical` styling property to `'true'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxImgFlipVertical='true';
```

Flip image horizontally

To flip image horizontally:

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxImgFlipHorizontal` styling property to `'true'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxImgFlipHorizontal='true';
```

Suppress output of image

To suppress output of image are:-

1. Get the DOM of the active layout.
2. Get the `<qx-box>` node .
3. Set the `qxImgSuppressOutput` styling property to `'true'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
boxes.style.qxImgSuppressOutput='true';
```

Working with Paragraphs

QXML for paragraph

Paragraph is represented by `<qx-p>` node. It contains following attributes:

Attribute	Value	Description
`class`	text	Name of paragraph style sheet and space separated H&J Style, Bullet and Numbering Style, Para Shading Style, Hanging Character Set

Create new paragraph

To create a new paragraph:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>`, `<qx-p>` and `<qx-span>` tags.
3. Create a new `<qx-p>` element.
4. Create a new `<qx-span>` element.
5. Append the `<qx-span>` element as child of `<qx-p>` element.
6. Append the `<qx-p>` element as child of `<qx-story>` node.

Sample JavaScript:

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var paras=boxes.getElementsByTagName('qx-p')[0];
var span=paras.getElementsByTagName('qx-span')[0];
var newPara=document.createElement('qx-p');
var newSpan=document.createElement('qx-span');
newSpan.innerText='NEW';
newPara.appendChild(newSpan);
paras.parentNode.appendChild(newPara);
```

Delete existing paragraph

To delete an existing paragraph:

1. Get the DOM of the active layout.

2. Get the `<qx-box>, <qx-story>` and `<qx-p>` node corresponding to the paragraph to be deleted.
3. Remove the desired `<qx-p>` child node from `<qx-story>` node.

Sample JavaScript:

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var story=boxes.getElementsByTagName('qx-story')[0];
var paras=story.getElementsByTagName('qx-p')[0];
story.removeChild(paras);
```

Apply margin

Margin top

To apply margin top:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxMarginTop` styling property to `'0.069in'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[1];
para.style.qxMarginTop='0.069in';
```

Margin bottom

To apply margin bottom:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxMarginBottom` styling property to `'0.042in'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxMarginBottom='0.042in';
```

Margin left

To apply margin left:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxMarginLeft` styling property to `'0.111in'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxMarginLeft='0.111in';
```

Margin right

To apply margin right:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxMarginRight` styling property to `'0.222in'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxMarginRight='0.222in';
```

Apply text indent

To apply text indent:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxTextIndent` styling property to `'0.208in'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxTextIndent='0.208in';
```

Apply line height

To apply line height:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxLineHeight` styling property to `'17.4pt'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxLineHeight='17.4pt';
```

Apply text align

To apply text align:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxTextAlign` styling property to `'justify'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxTextAlign='justify';
```

Apply initial letter and initial letter count

To apply initial letter and initial letter count:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .

3. Set the `qxInitialLetterCount` styling property to `'2'`.
4. Set the `qxInitialLetter` styling property to `'4'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxInitialLetterCount='2';
para.style.qxInitialLetter='4';
```

Apply orphans and widows

Orphans

To apply orphans:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxOrphans` styling property to `'2'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxOrphans='2';
```

Widows

To apply widows:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxWidows` styling property to `'3'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxWidows='3';
```

Break inside

To apply break inside:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxBreakInside` styling property to `'auto'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxBreakInside='auto';
```

Break after

To apply break after:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .

3. Set the `qxBreakAfter` styling property to `'avoid'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxBreakAfter='avoid';
```

Apply grid properties

Lock text to grid

To apply lock text to grid:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxLockTextToGrid` styling property to `'textbox'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxLockTextToGrid='textbox';
```

Text grid alignment

To apply text grid alignment:

1. Get the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxTextGridAlignment` styling property to `'topline'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxTextGridAlignment='topline';
```

Specify minimum distance between bullet and text

To specify the minimum distance between a bullet and text:

1. et the DOM of the active layout.
2. Get the `<qx-p>` node inside the `<qx-box>` node .
3. Set the `qxBnMinGap` styling property to `'0.2in'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
para.style.qxBnMinGap='0.2in';
```

Working with Spans

QXML for spans

Span is represented by `<qx-span>` node. It contains following attributes:

Attribute	Value	Description
`id`	id	Unique Element ID to create anchor
`class`	text	Name of Character style sheet and space separated text shading style

Create new span

To create a new span:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` and `<qx-p>` element.
3. Create new `<qx-span>` element.
4. Append the `<qx-span>` element as child of `<qx-p>` node.

Sample JavaScript:

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var paras=boxes.getElementsByTagName('qx-p')[0];
var newSpan=document.createElement('qx-span');
newSpan.innerText='NEW SPAN';
paras.appendChild(newSpan);
```

Delete existing span

To delete an existing span:

1. Get the DOM of the active layout.
2. Get the `<qx-box>`,`<qx-p>`,`<qx-span>` node corresponding to the span to be deleted.
3. Remove the `<qx-span>` child node from `<qx-p>` node.

Sample JavaScript:

```
var layout=app.activeLayoutDOM();

var boxes=layout.getElementsByTagName('qx-box')[0];
var paras=boxes.getElementsByTagName('qx-p')[0];
var spans=paras.getElementsByTagName('qx-span')[0];
paras.removeChild(spans);
```

Apply font size

To apply font size:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxFontSize` styling property to `'18pt'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxFontSize='18pt';
```

Apply color

To apply color:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxColor` styling property to `'magenta'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxColor='magenta';
```

Apply baseline shift

To apply baseline shift:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxBaselineShift` styling property to `'3pt'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxBaselineShift='3pt';
```

Apply font weight

To apply font weight:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxFontWeight` styling property to `'bold'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxFontWeight='bold';
```

Apply font style

To apply font style:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxFontStyle` styling property to `'italic'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxFontStyle='italic';
```

Apply font family

To apply font family:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxFontFamily` styling property to `'arial'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxFontFamily='arial';
```

Apply shade

To apply shade:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.

4. Set the `qxShade` styling property to `'0.6'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxShade='0.6';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Apply opacity

To apply opacity:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxOpacity` styling property to `'0.4'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxOpacity='0.4';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Apply letter spacing

To apply letter spacing:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxLetterSpacing` styling property to `'25'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxLetterSpacing='25';
```

Apply text properties

Text shadow

To apply text shadow:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.

4. Set the `qxTextShadow` styling property to `'true'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxTextShadow='true';
```

Text caps

To apply text caps:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxTextCaps` styling property to `'all-caps'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxTextCaps='all-caps';
```

Text decoration line

To apply text decoration line:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxTextDecorationLine` styling property to `'line-through'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxTextDecorationLine='line-through';
```

Text position

To apply text position:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxTextPosition` styling property to `'super'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxTextPosition='super';
```

Text superior

To apply text superior:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxTextPosition` styling property to `'super'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxTextSuperior='super';
```

Text outline

Toapply text outline:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxTextOutline` styling property to `'true'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxTextOutline='true';
```

Apply text stroke

Stroke color

To apply stroke color:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxStrokeColor` styling property to `'yellow'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxStrokeColor='yellow';
```

Stroke width

To apply stroke width:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxStrokeWidth` styling property to `'0.25pt'`.

```
var layout=app.activeLayoutDOM();
```

```
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxStrokeWidth='0.25pt';
```

Stroke shade

To apply stroke shade:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxStrokeShade` styling property to `'0.4'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxStrokeShade='0.4';
```

Can be specified in the 0 - 1 range (0=0% and 1 = 100%).

Example values: 0 = 0%, 0.25 = 25%, 0.50 = 50% and 1 = 100%.

Stroke linejoin

To apply stroke linejoin:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxStrokeLinejoin` styling property to `'round'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxStrokeLinejoin='round';
```

Stroke miterlimit

To apply stroke miterlimit:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxStrokeLinejoin` styling property to `'miter'`.
5. Set the `qxStrokeMiterlimit` styling property to `'7'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxStrokeLinejoin='miter';
span.style.qxStrokeMiterlimit='7';
```

Apply language

To apply language:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` node and `<qx-p>` node .
3. Get the desired `<qx-span>` node inside the `<qx-p>` node.
4. Set the `qxLanguage` styling property to `'zh-Hans'`.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box')[0];
var para=boxes.getElementsByTagName('qx-p')[0];
var span=para.getElementsByTagName('qx-span')[0];
span.style.qxLanguage='zh-Hans';
```

Working with Hyperlinks

URL hyperlinks

URL hyperlinks on text

To apply URL hyperlinks on text:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` , `<qx-p>` and `<qx-span>` node .
3. Create the element `<qx-a>` and set its 'href' attribute to 'www.google.com' and 'title' attribute to 'google'.
4. Append the span inside the newly created `<qx-a>`.
5. Append the `<qx-a>` inside the paragraph.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box');
var paras = boxes[0].getElementsByTagName('qx-p');
var spans=boxes[0].getElementsByTagName('qx-span');
var url_anchor=document.createElement('qx-a');
url_anchor.setAttribute('href','www.google.com');
url_anchor.setAttribute('title','google');
url_anchor.appendChild(spans[0]);
paras[0].appendChild(url_anchor);
```

URL hyperlinks on box

To apply URL hyperlinks on box:

1. Get the DOM of the active layout.
2. Get the desired `<qx-spread>` and `<qx-box>` node .
3. Create the element `<qx-a>` and set its 'href' attribute to 'www.google.com' and 'title' attribute to 'google'.
4. Append the box inside the newly created `<qx-a>`.
5. Append the `<qx-a>` inside the spread.

```
var layout=app.activeLayoutDOM();
var spreads=layout.getElementsByTagName('qx-spread');
var boxes=layout.getElementsByTagName('qx-box');
var url_anchor=document.createElement('qx-a');
url_anchor.setAttribute('href','www.google.com');
url_anchor.setAttribute('title','google');
url_anchor.appendChild(boxes[0]);
```

```
spreads[0].appendChild(url_anchor);
```

Anchor hyperlinks

Creating new anchor

To create a new anchor:

1. et the DOM of the active layout.
2. Get the desired `<qx-box>` and `<qx-p>` node .
3. Create the element `<qx-bookmark>` and set its 'id' attribute to 'anchor'.
4. Create the element `<qx-span>` and set its inner text to 'NEW BOOKMARK'.
5. Append the bookmark inside the paragraph.
6. Append the span inside the paragraph.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var paras= boxes[0].getElementsByTagName('qx-p');
var new_bookmark=document.createElement('qx-bookmark');
new_bookmark.setAttribute('id','anchor');
var new_span=document.createElement('qx-span');
new_span.innerText='NEW BOOKMARK';
paras[0].appendChild(new_bookmark);
paras[0].appendChild(new_span);
```

Anchor hyperlinks on text

To apply anchor hyperlinks on text:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` , `<qx-p>` and `<qx-span>` node .
3. Create a new bookmark as mentioned in the above section.
4. Create the element `<qx-a>` and set its 'href' attribute to the id of the bookmark.
5. Append the span inside the newly created `<qx-a>`.
6. Append the `<qx-a>` inside the paragraph.

```
var layout = app.activeLayoutDOM();
var boxes = layout.getElementsByTagName('qx-box');
var paras=boxes[0].getElementsByTagName('qx-p');
var spans=paras[0].getElementsByTagName('qx-span');
var anchor_span=boxes[1].getElementsByTagName('qx-span');
anchor_span[0].innerHTML='I am Anchor for Box 1';
var new_bookmark=document.createElement('qx-bookmark');
new_bookmark.setAttribute('id','anchor');
var list=anchor_span[0].parentNode;
list.insertBefore(new_bookmark,list.childNodes[0]);
var new_anchor_url=document.createElement('qx-a');
new_anchor_url.setAttribute('href','#anchor');
new_anchor_url.appendChild(spans[0]);
paras[0].appendChild(new_anchor_url);
```

Page hyperlinks

Page hyperlinks on text

To apply page hyperlinks on text:

1. Get the DOM of the active layout.
2. Get the desired `<qx-box>` , `<qx-p>` and `<qx-span>` node .
3. Create the element `<qx-a>` and set its 'href' attribute to '#page=2'.
4. Append the span inside the newly created `<qx-a>`.
5. Append the `<qx-a>` inside the paragraph.

```
var layout=app.activeLayoutDOM();
var boxes=layout.getElementsByTagName('qx-box');
var paras = boxes[0].getElementsByTagName('qx-p');
var spans=boxes[0].getElementsByTagName('qx-span');
var page_anchor=document.createElement('qx-a');
page_anchor.setAttribute('href','#page=2');
page_anchor.appendChild(spans[0]);
paras[0].appendChild(page_anchor);
```

Page hyperlinks on box

To apply page hyperlinks on box:

1. Get the DOM of the active layout.
2. Get the desired `<qx-spread>` and `<qx-box>` node .
3. Create the element `<qx-a>` and set its 'href' attribute to '#page=2'.
4. Append the box inside the newly created `<qx-a>`.
5. Append the `<qx-a>` inside the spread.

```
var layout=app.activeLayoutDOM();
var spreads=layout.getElementsByTagName('qx-spread');
var boxes=layout.getElementsByTagName('qx-box');
var page_anchor=document.createElement('qx-a');
page_anchor.setAttribute('href','#page=2');
page_anchor.appendChild(boxes[0]);
spreads[0].appendChild(page_anchor);
```

API Reference

Click here to see the API Reference.