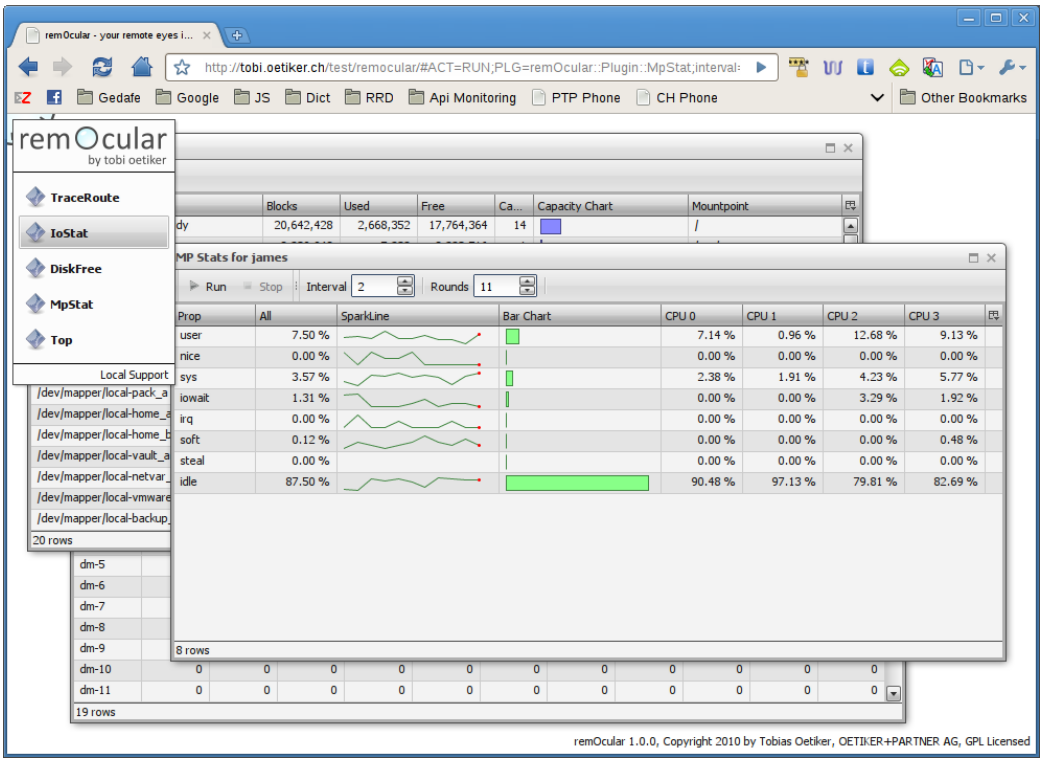
The Joy of writing JavaScript Applications

How qooxdoo puts the fun into programming for the web

Tobias Oetiker

FISL12

I always wondered ... how is this done?



1 Introduction

The Browser: my application platform

- Netscapes original Plan: Application in the Browser.
- JavaScript graduated with Web 2.0
- Fast JS: Nitro, V8, Carakan, Tracemonkey, Cranckshaft.
- Even MS has joined in with IE9.
- Still inconsistancies but minor.

It is said that people at Netscape back in the nineties had the vision of escaping the Microsoft dominance by enhancing their browser so that it could become a platform of its own for running client side applications. It is also said that MS was not thrilled by this thought.

Netscape is no more but the vision has become a reality. The Web 2.0 hype sparked a slew of highly interactive web applications that used JavaScript snipets on the browser to enhance the user experience.

Qooxdoo: applications in the browser

- Web 2.0 — a lot about the look and feel.
- Applications running in the browser.
- Back to client/server computing.
- Qooxdoo is for Js what Qt was for C++.

Qooxdoo features

- Turns JS into a grown-up OO language.
- No HTML or CSS knowledge required.
- Cross Browser: >= FF 1.5, Safari 3, Chrome, IE6, Opera8.
- Multilingual (gettext support).
- Full API Documentation.
- Widgetset for mobilde devices.

- Perfect Extensibility.
- LGPL, EPL
- Fun!

Qooxdoo is way more than yet another JavaScript widget collection. Apart from a cool collection of widgets, it introduces fully object oriented programming to the JavaScript world. Similar to the way OO got introduced in the Perl world, the Qooxdoo folks designed a framework that provides all of the OO bits that were left out of JavaScript's initial design.

2 Hello World

Jump right in

- Try the Playground `http://demo.qooxdoo.org/current/playground/`
- Download Qooxdoo `http://qooxdoo.org/download/`

Some claim Qooxdoo has a steep learning curve since it does just publish some JavaScript files you can link into your web page. While there are ways todo this all the same, I think it is actually a good thing since Qooxdoo's main objective is to provide an environment for writing standalone, browser based applications. With such a scope in mind, the developer should treat herself to a decent programming environment.

Generating the first application

- Point your path to `qooxdoo-1.4.1-sdk/tool/bin`
- Change directory to your development space.
- Run `create-application.py -name hello`
- CD into the `hello` directory.
- Run `generate.py source`
- Point your browser to `hello/source/index.html`

Qooxdoo comes with many sensible defaults. One could argue, that a lot of Qooxdoo's appeal comes from the many defaults. Normally when I start to write a program from scratch I am faced with way too many decisions at once. I often spend considerable time mulling about seemingly trivial decisions instead of just starting to program. Qooxdoo takes a lot of this "freedom" away by setting a standard on how to write your application. Many of these defaults can be changed, but I found that they are actually quite a good aproximation to my optimal programming environment, so there is no immediate need to change them.

generated files

```
hello/generate.py
hello/config.json
hello/source/resource/hello/test.png
hello/source/translation/readme.txt
hello/source/class/hello/test/DemoTest.js
hello/source/class/hello/Application.js
hello/source/index.html
hello/Manifest.json
hello/readme.txt
```

source code: hello/source/class/hello/Application.js

```
1  /* Tell qooxdoo that we need the resources in hello/*
2  #asset(hello/*)
3  */
4  qx.Class.define(                    ,
5  {
6    extend : qx.application.Standalone,
7    members :
8    {
9      main : function()
10     {
11       // Call super class
12       this.base(arguments);
13       // Enable logging in debug variant
14       if (qx.core.Variant.isSet(          ,     ))
15       { // native logging capabilities
16         qx.log.appender.Native;
17         // additional cross-browser console.
18         // Press F7 to toggle visibility
19         qx.log.appender.Console;
20       }
21       // Create a button
22       var button1 = new qx.ui.form.Button(
23                                          ,                 );
24       // Document is the application root
25       var doc = this.getRoot();
26       // Add button to document at fixed coordinates
27       doc.add(button1, {left: 100, top: 50});
28       // Add an event listener
29       button1.addListener(         , function(e) {
30         alert(              );
31       });
32     }
33   }
34 });
```

The original Qooxdoo hello world application, modified to fit the slide.

3 The Qooxdoo OO Features

Class definition

In its most basic form, a Qooxdoo class is very simple.

```
1 qx.Class.define(                );
```

In reality you would use something like this

```
1 qx.Class.define(                , {
2   // declare constructor, members, ...
3 });
```

A regular class can then be instantiated

```
1 var myClass = new my.cool.Class;
```

Class inheritance

The map contains the meat of the class.

```
1 qx.Class.define(                ,
2 {
3   extend : my.great.SuperClass,
4   construct : function() { ... },
5   destruct  : function() { ... }
6 });
```

Embrace and extend.

Instance Members

Instance members reside in the members map.

```
1 qx.Class.define(                , {
2   members: {
3     foo : VALUE,
4     bar : function() { ... }
5   }
6 });
```

Use `new` to create an instance.

```
1 var myClass1 = new my.cool.Class;
2 myClass1.foo = 3.141;
3 myClass1.bar();
```

Calling the Superclass

```
qx.Class.define(                ,
{
  extend : my.great.SuperClass,
  construct : function(x) {
    this.base(arguments, x); // superclass constructor
  }
  members : {
    foo : function(x) {
      this.base(arguments, x);
    }
  }
});
```

The `this.base` construct works for both constructor and member functions.

The `arguments` object/map used in this example is a native JavaScript feature. Inside a function call it contains all the information about how the function was called: a list of the arguments passed to the function as well as pointers to the function itself.

class access control

There is the following naming convention for class members.

```
publicMember
_protectedMember
__privateMember
```

In the Qooxdoo build process names of private members get mangled to prevent outside access.

static, abstract and singleton classes

```
qx.Class.define(                  , {
  type :
  statics: { ... };
});
```

Neither members nor constructors are allowed in static classes.

```
qx.Class.define(                    , {
  type :
});
```

Abstract classes must be sub-classed for use.

```
qx.Class.define(                     , {
  type :
});
var instance = my.singleton.Class.getIntance()
```

There is only one instance which gets created on the first call.

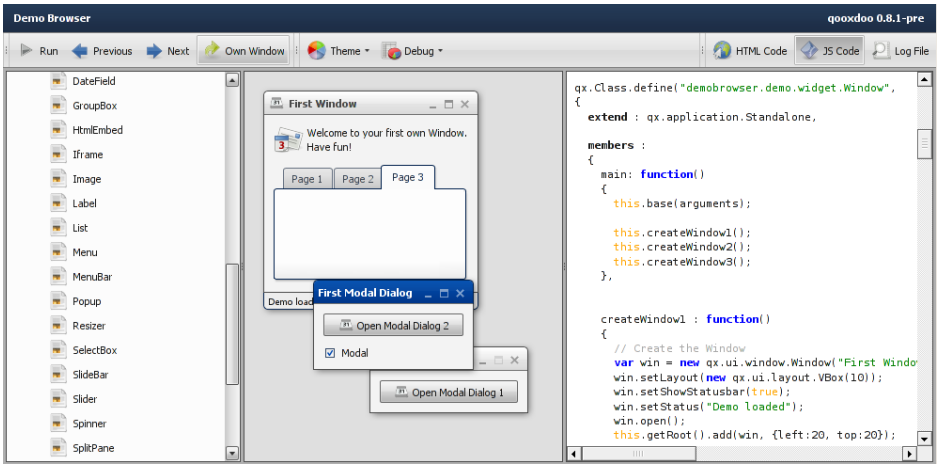
Browser specific code

Normally Qooxdoo takes care of all browser differences, but if you must intervene ...

```
1  members: {
2    foo: qx.core.Environment.select(
3                    , {
4                        : function() {
5          // Internet Explorer or Opera
6        },
7                  : function() {
8          // All other browsers
9        }
10     }
11   )
12 }
```

4 Working with Qooxdoo

The demo Browser

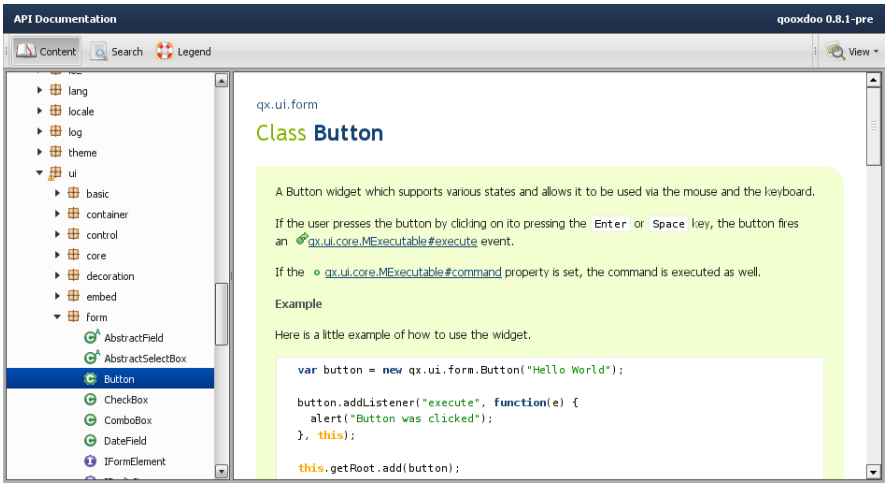



```
1 $ cd $QX/application/demobrowser/
2 $ ./generate.py build
3 $ gnome-open build/index.html
```

Or surf to `http://demo.qooxdoo.org/current/demobrowser`

For me the demobrowser is the quickest way of seeing how to write Qooxdoo. Select a widget on the tree at the left and activate the JS Code toggle. Now you can see both the running program as well as the JavaScript code. The rest is mostly cut and paste.

The API Documentation




```
$ cd $QX/framework
$ ./generate.py api
$ gnome-open api/index.html
```

Or surf to `http://demo.qooxdoo.org/current/apiviewer`

The Qooxdoo API documentation is generated directly from embedded javadoc in the Qooxdoo JS source files. You can apply the same process to your own Qooxdoo application to get a api viewer for your own code.

The Qooxdoo generator

Python is the sole dependency

- `generator.py` is the tool
- it gets called by `generate.py`

The generator has many functions

- source - prep code for running in souce
- source-hybrid - pre-compiled source version (faster loading)
- build - prep code for deployment
- api - build api doc
- lint - check your code for common errors
- pretty - fix the code layout
- translation - generate locale files

Running your Qooxdoo program in source

Use source code during development

```
1 $ cd hello
2 $ ./generate.py source-hybrid
3 $ gnome-open source/index.html
```

As long as you do not use any new classes, press reload in the browser to see changes.

To run a Qooxdoo application, the code for each class you used must be loaded. This can easily be 30 or more files. When calling the generator with the option source it will create an JavaScript file in `source/script/hello.js` which takes care of loading these class files. While developing you may want to try the `lint` option as well, to catch some frequent mistakes.

Deploying your Qooxdoo program

```
1 $ cd hello
2 $ ./generate.py build
3 $ cp -rp build ~/public_html/hello
```

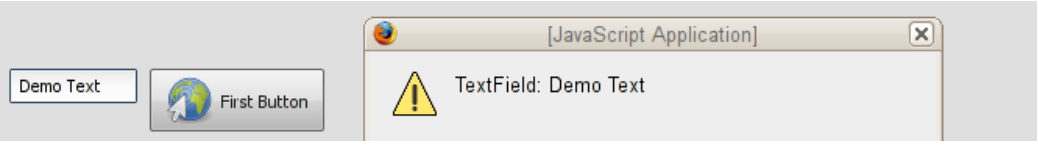
- only two js files
- code gets optimized and compressed
- no external dependencies

The Qooxdoo generator builds a fully custom js file containing all the Qooxdoo classes required to run your application. It also compresses and optimizes the code in this step. You will notice that the first source and build run will take quite some time. This is because Qooxdoo creates cache files of all classes involved. If you run the build for a second time things will run much quicker.

5 Programming with Qooxdoo

Button, TextField and some Action

```
1  // Create a textfield
2  var tf1 = new qx.ui.form.TextField(           );
3  // Add button to root
4  root.add(tf1, {column: 0, row: 0});
5  // Create a button
6  var bt1 = new qx.ui.form.Button(
7                    ,                   );
8  // Add button to root
9  root.add(bt1, {column: 1, row: 0});
10 // Add an event listener
11 bt1.addListener(          , function(e) {
12   // closure !!
13   this.info(              +tf1.getValue());
14   alert(               + tf1.getValue());
15 });
```



Try F7 to see inline console!

In this first example there is already a closure. The variable `tf1` is used inside the event handler. The function is passed as a reference and takes the access to the TextField object with it.

The Layout Manager

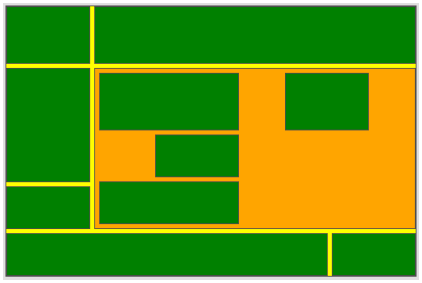
- Qooxdoo Widgets can contain other widgets.
- Layout manager positions child widgets.
- `qx.ui.container.Composite` basic
- `qx.ui.container.Scroll` draws scroll bars
- `qx.ui.window.Window` directs children to an inner composite pane.
- Layout manager set at construction time
- Modified with `setLayout` method.

Container and Layout

```
1  // a container with horizontal layouyt manager
2  var hbox =  new qx.ui.layout.HBox();
3  hbox.setSpacing(4); // set property
4
5  // assign layout
6  var ctr1 = new qx.ui.container.Composite(hbox);
7  ctr1.setWidth(600); ctr1.setHeight(40);
8  // layout properties: position
9  root.add(ctr1,{column: 0, row: 1, colSpan: 2});
10
11 var tf2 = new qx.ui.form.TextField(                );
12 var bt2 = new qx.ui.form.ToggleButton(            );
13 bt2.addListener(               , function(e) {
14     // modify widget property
15     tf2.setAllowGrowY(e.getData());
16        this.info(                            +e.getData());
17 });
18 ctr1.add(tf2);ctr1.add(bt2);
```

The container widget together with an associated layout object can arrange widgets on screen giving the user high level control over the operation. Here the toggle button lets us choose if the text field should grow vertically to fill the available space or not.

Grid Layout



- `qx.ui.layout.Grid`
- fully dynamic
- ideal for dialogs
- one widget per cell
- row and column spans
- minimal and maximal column and row sizes
- fixed row and column sizes

The grid widget has all the flexibility of a html table plus a great deal more. It is the ideal basis for laying out dialog boxes or complex screen setups.

About the Qooxdoo Layout Widgets

- A container widget needs a layout manager to place its children.
- The layout manager object has properties.
- Every widget has basic properties like: alignment, growability, shrinkability, stretchability, margins, padding, width and height.
- Each widget can have layout-specific properties.
- Layout properties get checked as the widget is added to a layout.

For me the best way to understand how layouts work was to first try them out in the demo browser and then use them in a little program of my own.

Localized Applications

```
1  var lmgr = qx.locale.Manager.getInstance();
2  var bt3 = new qx.ui.form.ToggleButton(
3      this.tr(            )
4  );
5  root.add(bt3, {column: 1, row: 3});
6  bt3.addListener(               , function(e) {
7      var lang =  e.getData() ?      :     ;
8      lmgr.setLocale( lang );
9      this.info(                   +lang);
10 });
```

- add locale to `config.json`
- `./generate.py translation`
- translate `de.po`
- `./generate.py source`

Qooxdoo locale files are normal .po files. You can use any of the existing kde/gnome tools for updating your translations. Qooxdoo will automatically pick the language that best matches the locale settings in your browser.

Calling code on the Server

- JSON RPC for transport
- various language bindings `http://qooxdoo.org/contrib/project\#backend`
- often minimal server code
- async with callbacks
- `qx.io.Rpc`

Organizing the code into multiple classes

- Object orientation “by the book”.
- One file per class.
- Java’s file name based approach.
- Supported by the generator.
- Ideal for code re-use.
- Use Inline Docs!
- `./generate.py api`

The textclick class

```
/**
 * textclick combines a textfield and a button.
 */
qx.Class.define(                    ,
{
  extend : qx.ui.container.Composite,
  /**
   * @param button_text {String} button text.
   */
  construct : function(button_text) {
    this.base( arguments,
        new qx.ui.layout.HBox().set({spacing: 4})
    );
    this.__tf = new qx.ui.form.TextField();
    this.__bt = new qx.ui.form.Button(button_text);
    this.add(this.__tf);
    this.add(this.__bt);
  },


  members :
  {
  /**
   * Get a handle to the Button widget.
   */
    getButton: function() { return this.__bt },
  /**
   * Get a handle to the TextField widget.
   */
    getTextField: function() { return this.__tf },
    __bt: null,
    __tf: null
  }
});
```

Using the textclick class

```
var mywi =new osd210.ui.textclick(
           this.tr(                          ));

mywi.getButton().addListener(         , function(e) {
    mywi.getTextField().setValue(tf1.getValue());
    this.info(                   +tf1.getValue());
});

root.add(mywi,{column: 0, row: 5, colSpan: 2});
```

By splitting your application into different classes, your code becomes simpler to understand and to test, you can even re-use it in other projects.

The generator tool will merge your own classes with the relevant Qooxdoo Classes into optimized, monolithic JavaScript files, ready for deployment.