QCObjects

Bienvenido a QCObjects. Un framework Open Source que empodera a los full-stack developers para hacer micro-servicios y micro-frontends dentro de una arquitectura N-Tier.

Con QCObjects los desarrolladores están habilitados para programar front-end y back-end en conjunto usando una sintaxis común in pure JavaScript. It is cross-browser, cross-platform and cross-frame.

QCObjects ha sido presentado por British Herald como el marco más avanzado para el desarrollo de software moderno.

Este documento es la documentacion referencial principal!

Este repositorio y archivo léame está alojado en https://qcobjects.dev

Echa un vistazo a la página oficial de QCObjects at https://qcobjects.com

Este proyecto se adhiere al Pacto Colaboradorcode of conduct. Al participar, se espera que respete este código. Por favor reportar algún comportamiento inaceptable a info@quickcorp.cl.

Contributors are welcome!

Puedes contribuir a QCObjects siguiendo el conjunto de pautas expresadas en el archivo CONTRIBUTING.md

Video Explicativo de QCObjects

Para todos los que no tengan tiempo de leer esto hoy, aquí hay un pequeño video que explica que es QCObjects y que puedes hacer con el.

Tabla de Contenidos

    - Use:
- ALPHA RISE Startup

Principios

Aquí están Las directrices con lo que QCObjects fue hecho:

0. Deberá escribir en JavaScript para codificar una aplicación JavaScript.
1. Todo es un objeto.
2. Cada objeto tiene una definición.
3. En la interfaz, cualquier objeto puede ir apilado en el DOM o en el Virtual-DOM sin necesidad de redeclarar sus definiciones.
4. Cada objeto tiene un cuerpo.
5. La clase debería ser la definición principal de un objeto.
6. La clase debería ser fácilmente escrita como un objeto.
7. Tu Código debería estar fácilmente organizado en paquetes.
8. Debería ser posible escalar sus aplicaciones a una arquitectura limpia.
9. Un componente es una entidad que tiene un objeto como representación. El contenido de un componente debería ser posible rellenarlo remotamente como localmente. Como objeto el componente tiene cuerpo También y el cuerpo del componente es normalmente una instancia apilada del DOM element.
10. Un componente puede ser adjunto al DOM o separado del el sin afectar a su funcionalidad.
11. Un servicio de llamada puede ser extendido a escalar su funcionalidad.
12. Deberías ser capaz de importar un paquete remotamente.
13. Deberías poder escalar tu código y también controlar tus cambios en el servidor sin hacer llamadas innecesarias a fuentes remotas. No deberías necesitar codificar estos tipos de controles usted mismo.
14. Deberías ser capaz de codificar tu aplicación N-Tier en un solo lenguaje o sintaxis.
15. Deberías ser capaz de aplicar cualquier plantilla que quieras a un componente, no importa la sintaxis o el idioma en el que esta escrito.
16. Si una etiqueta HTML esta ya representada por una instancia de objeto DOM, no deberías necesitar duplicar la defunción de la instancia para representar su contenido.
17. Tu pagina principal HTML debería estar limpia, pero deberías poder enlazar lo que controla el comportamiento de la etiqueta sin afectar la sintaxis del HTML.
18. El orden de ejecución de tu código debe ser fácil de entender y leer desde el codigo y el proceso de renderizado de cada componente debería tener y ejecutar control en cuantas capas necesites.
19. Un patrón en capas(como el MVC o MVCC) debería estar presente para cada componente. No importa si defines cada capa o no.
20. El comportamiento de un componente no debe estar determinado por su proceso de renderizado
21. Es necesario que la pila de componentes se divida en el DOM hacia un árbol subyacente de elementos adjuntos.Entonces ahora existe y se llama Pila anidada de componentes de QCObject.
22. Deberías ser capaz de extender una instancia de componente. Pero deberás ser capaz de controlar su comportamiento dinámico sin afectar a la declaración inicial.
23. Deberías ser capaz de aplicar efectos visuales y animaciones simultaneas de una manera facil a una instancia de elemento DOM.
24. Deberías ser capaz de controlar los efectos visuales y animaciones de CSS como JavaScript sin afectar a su desempeño.
25. Deberías ser capaz de controlar el comportamiento de tu código Into-the-box y out-of-the-box y sobrevivir haciéndolo.

Características principales

- Plantillas Built-In personalizadas para Progressive Web Apps (PWA) y Accelerated Mobile Pages (AMP)
- Efectos UI revolucionarios.
- Backend de micro-servicios avanzados.
- La simplicidad de un maravilloso diseño de layouts.
- Herramientas CLI completamente re-utilizables.
- Arquitectura orientada a componentes y objetos
- Front-end y back-end juntos en un entorno Full-Stack
- Routing recursivo para componentes.
- Administración de componentes anidados Built-In
- Patron MCV completamente integrado (Model, View, Controller)
- Objetos de datos dinamicos
- Conceptos basados en la arquitectura N-Tier

Características Adoptadas de Apps Web Progresivas (PWA)

Prevenir recursos Render-blocking

Para prevenir los recursos Render-blocking , QCObjects ha implementado la función de fabrica paquete

Carga de recursos On-Demand

Con la arquitectura orientada a los componentes dinámicos, QCObjects renderiza cada recurso visual que esta dentro de un componente, solo cuando el componente se esta construyendo y cada componente esta conectado a un árbol llamado global.componentsStack ese es el que realmente esta apuntando a cada instancia de componente y sus a sus sub componentes. Cada vez que un componente es re-hecho, los recursos visuales están dinámicamente recargados bajo demanda de la manera mas eficiente, así que puedes olvidar esos horribles códigos donde necesitabas controlar el proceso de recarga de los recursos con otros frameworks.

Lazy-loading de imágenes y componentes (usar atributo lazy-src en vez de src en tag img)

Desde la versión 2.1.251, QCObjects te otorga una forma fácil para el Lazy load de imágenes, usando el ultima estándar para los buscadores.

```
<img src="img/preloader.svg" lazy-src="img/myrealimage.png"/>
```

En lo anterior, una imagen (ligera) precargada, es usada para ser cargada en la primera instancia y un atributo **lazy-src** es usado para cargar la imagen real después del proceso Lazy load. QCObjects cargara

todos las etiquetas declaradas dentro de un componente en el lazy mode si tiene un atributo lazy-src, después que un componente es rearmado o cargado. También, QCObjects usara Intersection Observer API (Cuando este disponible) para determinar ya sea si el lazy-src o la imagen src son visualmente útiles para ser mostradas.

El efecto del Lazy loading es altamente visible solo si la primera vez el PWA es cargado. La próxima vez, la velocidad de carga aumentara significativamente haciendo difícil para el ojo humano ver el resultado. Sin embargo esta característica hará mucho la diferencia en términos de experiencia de usuario, si existen problemas de conexión o las imágenes son muy grandes esta característica es parte de las recomendadas

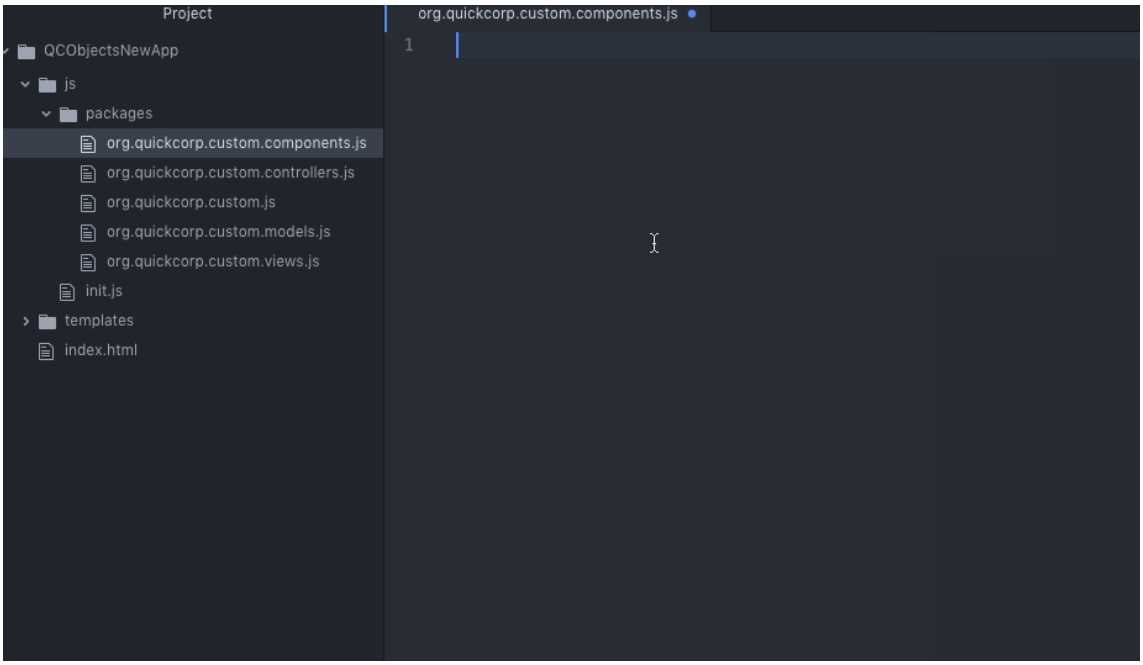
por os escritores de PWA por Mozzila Developers Network un articulo sobre Loadig progresivo. Puedes leer el articulo here

Si no quieres usar lazy loading para las imágenes, siempre puedes mantener la forma usual de carga no

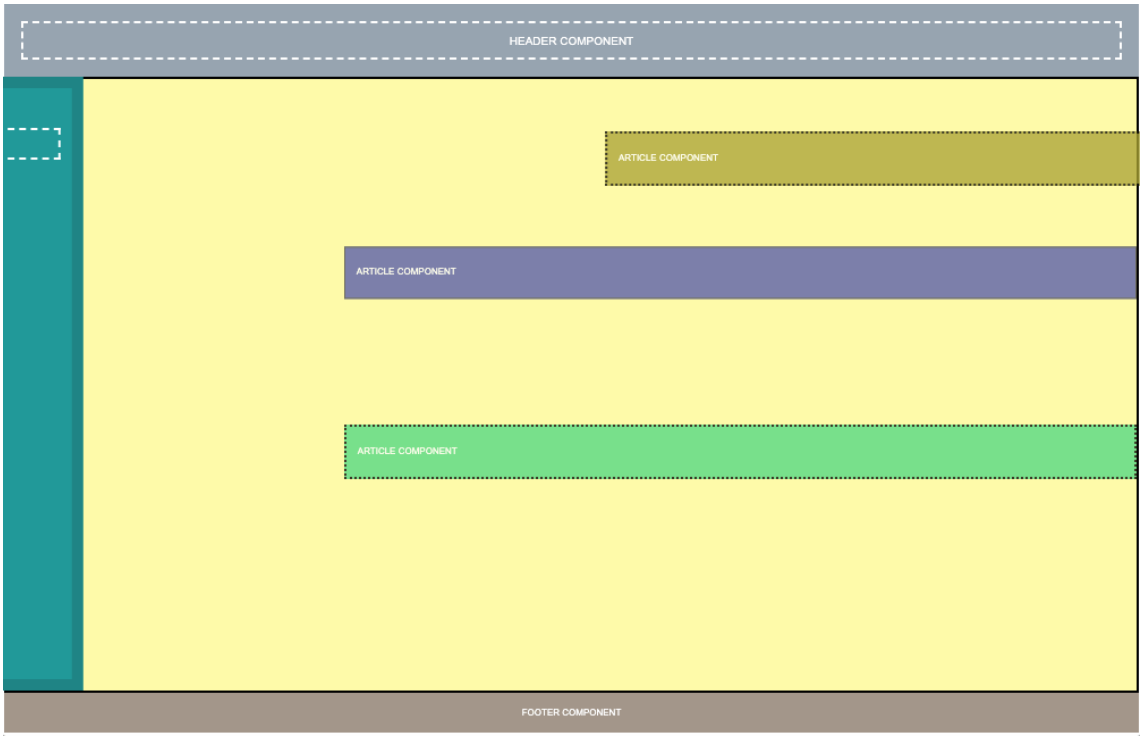
añadiendo el atributo **lazy-src** a la etiqueta y usando el tradicional **src**.

Cross Browser Javascript Framework para patrones MVC

QCObjectsEs un framework de Javascript diseñado para hacer todo mas fácil sobre la implementación de los MVC patters en el alcance de pure Javascript. No necesitas utilizar un typescript ni ningún transpiler para que corra. QCObjects. Corre directamente en el buscador y usa pure javascrpt sin ninguna dependencia de código extra. Puedes crear tus propios componentes expresados en objetos nativos reales de Javascript o objetos nativos DOM extendidos para usarlos a tu manera. Puedes también usar QCObjects] (https://qcobjects.dev) En conjunto con CSS3 frameworks como [Foundation] (https://foundation.zurb.com), [Bootstrap] (https://getbootstrap.com) Y frameworks de mobil javascript como [PhoneGap] (https://phonegap.com) y OnsenUI (https://onsen.io)



Arquitectura de Componentes Dinámicos



Especificación ECMA-262

See ECMAScript® 2020 Language Specification como referencia.

Copyright

Demo

PWA Live Demo

Mira esta demo en vivo de pure QCObjects basado en aplicación web fronted aquí: PWA QCObjects

Demo Integrada con Foundation

A continuación, una demostración en vivo usando componentes de Foundation aquí: Demo usando Foundation

Demo Integrada con Materializecss

Revisa la demostración usando MaterializeCSS aquí: Demo Using Materializecss

Demo Usando Raw CSS

Revisa esta demo usando raw CSS aquí: [Demo Using Raw CSS]

Ejemplo de QCObjects usando y manipulando objetos canvas

A continuación el código muestra como QCObjects puede manipular un objeto de lienzo directamente y dentro de componentes

```
<!DOCTYPE html>
<html>
    <head>
        <title>Demo</title>
        <script type="text/javascript" src="https://qcobjects.dev/QCObjects.js">
</script>
        <script type="text/javascript">
            var canvas1,canvas2,canvas3,container;
        CONFIG.set('relativeImportPath','src/');

            /**
             * Main import sentence.
             */
            Import('cl.quickcorp',function (){

                /**
                 * Super Container MyOwnBody
                 */
                Class('MyOwnBody',HTMLBodyElement,{
                    customAttr:'custom',
                    body:document.body  // breakes default body element and replace
with them
                });

                /**
                 * Another custom class definition
                 */
                Class('MyContainer',HTMLElement,{
                    width:400,
                    height:400,
                    customAttr:'custom attr container'
                });


                /**
                 * Another custom class definition
                 */
                Class('canvas',HTMLCanvasElement,{
                    customAttr:'custom'
                });

                /**
                 * Another custom class definition
```

```
                 */
                Class('MyCanvas2',HTMLCanvasElement,{});

                body = New(MyOwnBody); // binds to body
                body.css({backgroundColor:'#ccc'});

                container = Tag('container')[0].Cast(MyContainer); // cast any
javascript dom object to QC_Object class
                container.css({backgroundColor:'red'}); // access binding in two
directions to dom objects

                /**
                 * Instance a new custom canvas
                 */
                canvas1 = New(canvas,{
            width:100,
            height:100,
          });
                canvas2 = New(canvas,{
            width:200,
                    height:100,
          });
                canvas3 = New(canvas,{
            width:300,
                    height:50,
          });

                canvas1.css({backgroundColor:'#000000'}); // like jquery and another
style access
          canvas1.body.style.backgroundColor='#000000'; // standard javascript style
access
                canvas2.body.style.backgroundColor='#0044AA'; // standard javascript
style access
                canvas3.body.style.backgroundColor='green'; // standard javascript
style access

                canvas1.append(); //append canvas1 to body
                canvas2.attachIn('container'); // attach or append to specific tag
containers
                container.append(canvas3); // append canvas3 to custom tag binding

//                canvas1.body.remove(); // remove canvas1 from dom
                body.append(canvas3); // append canvas3 to body

          // using components
          var c1 = New(Component,{'templateURI':'templatesample.html',cached:false});
          document.body.append(c1); // appends the c1 to the body


            });

        </script>
    </head>
```

```
    <body>
        <container id="contentLoader" ></container>
    </body>
</html>
```

DevBlog

el Official DevBlog of QCObjects esta alojado en Hashnode. El DevBlog esta personalmente escrito por Jean Machuca, el autor de QCObjects Y el esta explicando en detalle como son las mejores practicas y dando los mejores tips y trucos para usar las mas avanzadas características de QCObjects.

Fork

Por favor has Fork a este projecto o crea un link a este proyecto en to archivo README.md. Lee el archivo LICENSE.txt antes de usar este código.

Conviértete en Sponsor

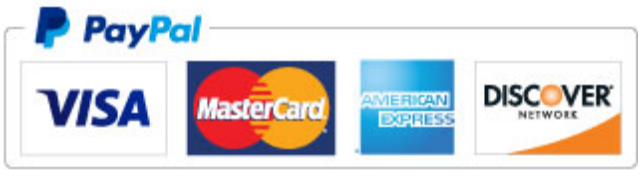
Si quieres volverte sponsor de este maravilloso proyecto puedes hacerlo aquí

Revisa el SDK de QCObjects

Puedes revisarQCObjects SDK y seguir los ejemplos para hacer tus propios componentes destacados.

Donar

Si te gustó este código por favor DONA!



Instalar

Usando QCObjects con Atom:

```
> apm install qcobjects-syntax
```

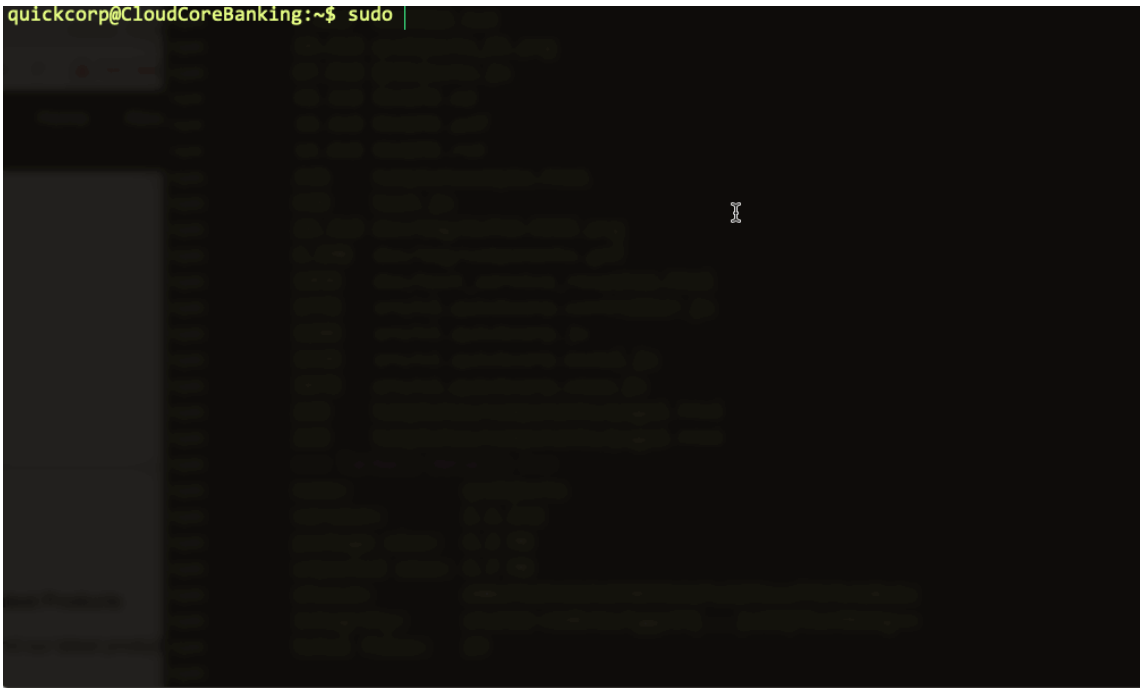
https://atom.io/packages/qcobjects-syntax

Usando QCObjects con Visual Studio Code:

https://marketplace.visualstudio.com/items?itemName=Quickcorp.QCObjects-vscode

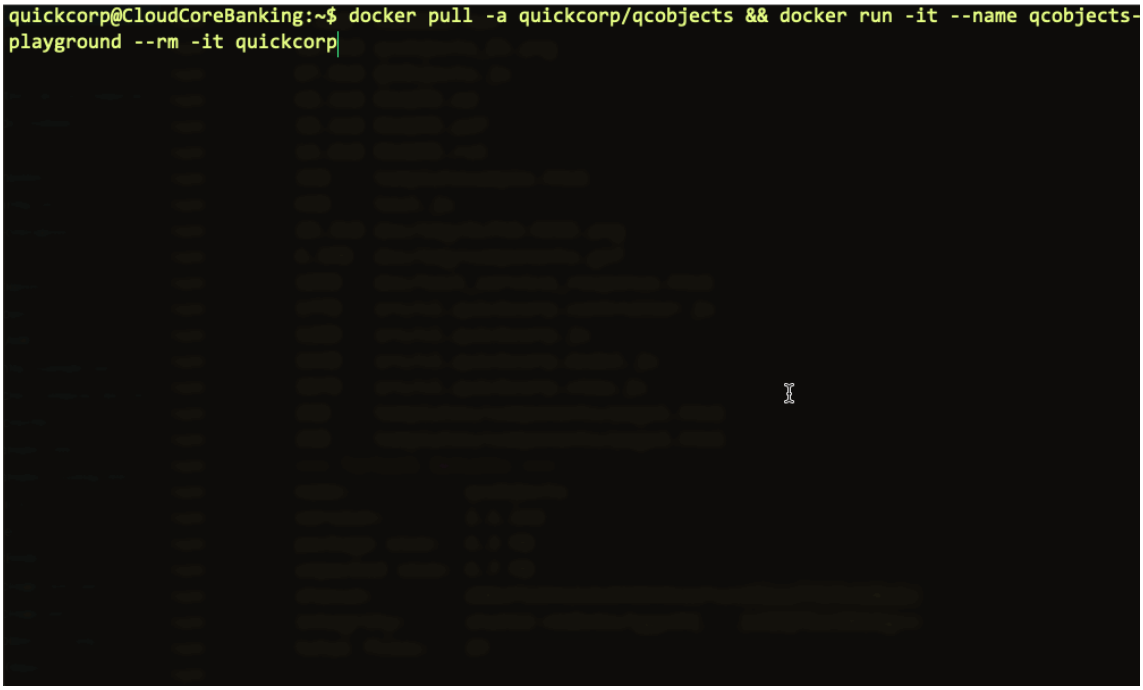
Instalando con NPM:

```
> npm install qcobjects-cli -g && npm install qcobjects --save
```



Instalando el docker playground:

```
docker pull -a quickcorp/qcobjects-playground && docker run -it --name qcobjects-
playground --rm -it quickcorp/qcobjects-playground
```



Script de instalación One-Step para Ubuntu 18.x

ATENCIÓN: Haz esto solo en una instalación de Ubuntu 18.x fresca/vacia/actual. No lo haga en un ambiente existente de producción. Se te pedira permiso sudo grant.

```
curl -L https://qcobjects.dev/install_qcobjects_ubuntu18x.sh |sh
```

ATENCIÓN: No somos responsables de el daño en la infraestructura por usar una instalación automatizada de script en una network insegura. Asegúrate de que tus repos y scripts están bajo HTTPS con su certificado valido. Para mejores resultados te recomendamos descargar el script, editarlo para tus necesidades especiales y después ejecútalo en tu maquina local.

Script de instalación One-Step for macOS

Probado en macOS Catalina 10.15.3

```
curl -L https://qcobjects.dev/install_qcobjects_macOS.sh | zsh
```

Instalar y probar QCObjects en Sistema Operativo Microsoft Windows

1.- Instala la ultima versión de NodeJS para WindowsAquí 2.- Desde el cmd instala qcobjects-cli usando npm

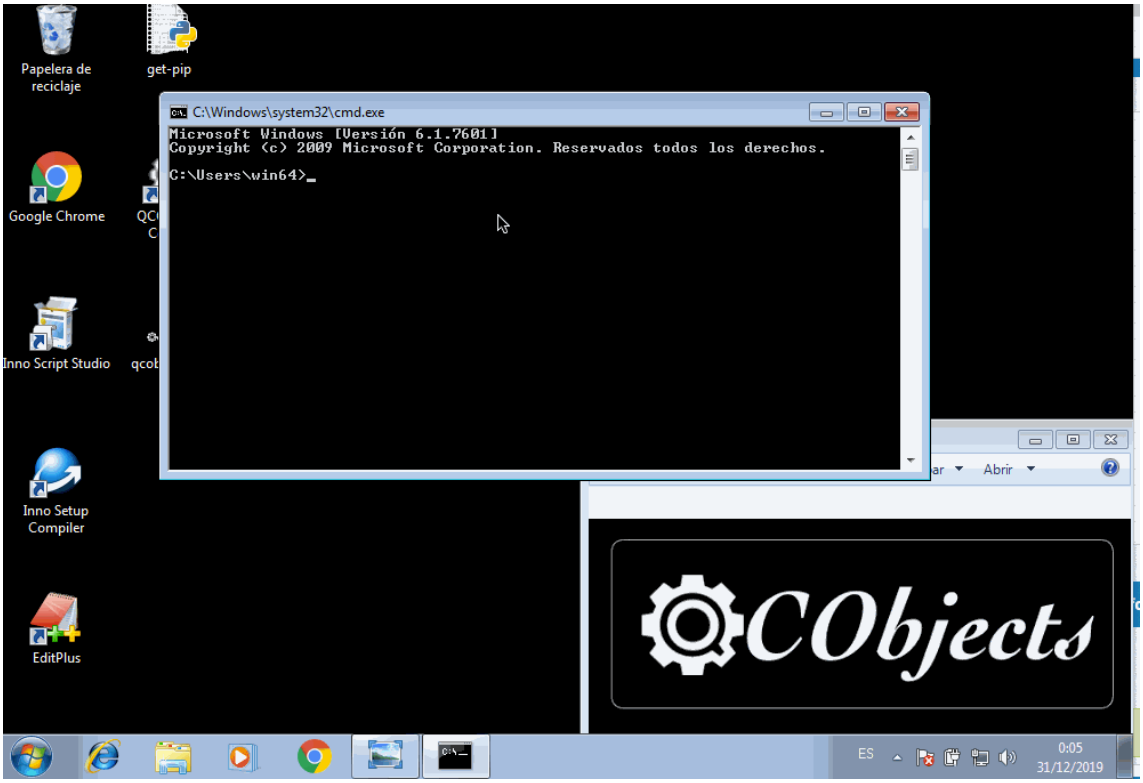
```
npm i qcobjects-cli -g
```

3.- Crea un directorio de para tu proyecto

```
md mynewproject && cd mynewproject
```

4.- Crea una nueva aplicación web progresiva de QCObjects

```
qcobjects create mynewproject --pwa
```



Instalación QCObjects Multi-Cloud

QCObjects es nativamente soportado por los mas famosos proveedores de nubes. Puedes instalar la mayoría de ellos, preparar y correr todo en un solo paso.

DigitalOcean One-Click Droplet

Si quieres olvidar apt-get y de configurar la guía, ve directo a desplegar tu proyecto usando una preconfigurada app 1-click incluyendo QCObjects CLI, QCObjects-SDK y QCObjects HTTP2 servidor Built-in. Luego giralo a Droplet VM o a Kubernetes cluster en 60 segundos o menos.

Crea tu propio Droplet de QCObjects DigitalOcean Aquí

AWS Amazon Machine Images (AMI)

Un Amazon Machine Image (AMI) otorga información requerida para lanzar una instancia. Tienes que especificar un AMI cuando quieras lanzar una instancia. Puedes lanzar múltiples instancias para un solo AMI cuando necesites múltiples instancias con la misma configuración. Puedes usar diferentes AMIs para lanzar instancias cuando necesites instancias con diferentes configuraciones.

Un AMI incluye lo siguiente:

- Uno o mas EBS snapshots, o, para instance-store-backed AMIs, una plantilla para la raíz volumen de la instancia(por ejemplo, un sistema operativo, un servidor de aplicaciones y aplicaciones).
- Lanza permisos que controla que cuenta AWS puede usar el AMI para lanzar instancias.
- Un bloqueo de dispositivos mapping que especifica los volúmenes adjuntos a la instancia cuando es lanzada.

Empieza construyendo QCObjects AMI aquí

Amazon Web Services AWS PIB (Private Amazon Machine Image)

Imagen privada que te permite construir un nuevo AMI instalando un software AWS Market place en una imagen que tu elijas del AMI disponibles en tu cuenta AWS, esto permite que conocer a mejores especificaciones internas para seguridad, gestiones y cumplimientos. Como con los Marketplace AWS AMIs estándar, cada imagen privada se comprometera a subscripción por el producto instalado y tiene uso de software facturado vía AWS Marketplace.

Empieza creando tu QCObjects Amazon Private Image aquí

Usando el código de la versión de desarrollo directo en HTML5

```
<script type="text/javascript" src="https://qcobjects.dev/QCObjects.js"></script>
```

Usando la versión minificada de código CDN desde jsDelivr

```
<script src="https://cdn.jsdelivr.net/npm/qcobjects/QCObjects.min.js"></script>
```

Usando la última versión no-minificada desde CDN jsDelivr

```
<script src="https://cdn.jsdelivr.net/npm/qcobjects/QCObjects.js"></script>
```

Usando CDN UNPKG

```
<script src="https://unpkg.com/qcobjects@latest/QCObjects.js"></script>
```

Usando CDNJS

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/qcobjects/[VERSION]/QCObjects.js">
</script>
```

Donde [VERSIÓN] corresponde a la ultima versión usando notaciones numericas, ejemplo: to use version 2.1.420:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/qcobjects/2.1.420/QCObjects.js">
</script>
```

No necesitas minificar QCObjects, pero si aún quieres usar el codigo minificado puedes hacer esto:

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/qcobjects/2.1.420/QCObjects.min.js">
</script>
```

Otra vez cambia 2.1.420 al numero de la versión que quieras usar.

Referencia

Aquí están los símbolos y conceptos esenciales de referencia QCObjects

QC_Object

Tipos básicos de todos los elementos

ComplexStorageCache

Con **ComplexStorageCache** puedes manejar el cache de cualquier objeto y subirlo en el storage local.

Uso:

```
var cache = new ComplexStorageCache({
                      index:object.id, // Object Index
                      load:(cacheController)=>{}, // A function to execute for the
first time
                      alternate: (cacheController)=>{} // The alternate function to
execute from the second time the source coude is loaded
                      });
```

Ejemplo:

```
var dataObject = {id:1,
                  prop1:1,
                  prop2:2
                 };

var cache = new ComplexStorageCache({
    index: dataObject.id,
    load: (cacheController) => {
      dataObject = {
              id:dataObject.id,
              prop1:dataObject.prop1*2, // changing a property value
              prop2:dataObject.prop2
            };
      return dataObject;
    },
    alternate: (cacheController) => {
      dataObject = cacheController.cache.getCached(dataObject.id); // setting
dataObject with the cached value
      return;
    }
  });

// Next time you can get the object from the cache
var dataObjectCopyFromCache = cache.getCached(dataObject.id);
console.log(dataObjectCopyFromCache); // will show the very same object value than
dataObject
```

asyncLoad

La función **asyncLoad** carga el código una vez en el modo asyc. Esto es útil para asegurar que el proceso inicial no replica la ejecución y no es recargado después de un código sensible.

Uso:

```
asyncLoad(()=>{
  // my code here
},args);
// Where args is an array of arguments, it can be the "arguments" special object
```

Ejemplo:

```
let doSomething = (arg1,arg2)=>{
  asyncLoad((arg1,arg2)=>{
    console.log(arg1);
    console.log(arg2);
  },arguments);
};

doSomething(1,2); // the code of doSomething will be executed once after the rest of
asyncLoad queue of functions and before the execution of Ready event.
```

Class

Esto NO es una clase de definición de ECMAScript 2015 (mira clase ECMAScript 2015 for reference).

Clase es una función especias que te ayuda a declarar la clase de una manera mas fácil y compatible. Funciona con cross-browser, y esperamos que ECMA pueda adoptar algo similar en el futuro. Para no dejar al Javascript confuso sobre esto, QCObjects usa "Class" no "class" (note the Camel Case).

Uso:

```
Class('MyClassName',MyClassDefinition);
```

Donde **MyClassDefinition** es un objeto junto a el **prototype** de QCObjects

Ejemplo:

```
Class('MyClassName',InheritClass,{
  propertyName1:0, // just to declare purpose
  propertyName2:'',
  classMethod1: function (){
    // some code here
    // note you can use "this" object
    return this.propertyName1;
  },
  classMethod2: function () {
    // some code here
    return this.propertyName2;
  }
});

var newObject = New(MyClassName,{
    propertyName1:1, // this initializes the value in 1
    propertyName2:"some value"
```

```
});
console.log(newObject.classMethod1()); // this will show number 1
console.log(newObject.classMethod2()); // this will show "some value"
```

Método QC_Append, append

Este es un método especial que hará tu vida mas fácil cuando quieras manipular el **DOM** dinámicamente. Puedes insertar un componente incluso **Component**, a un objeto **QCObjects** o a el elemento **DOM** dentro de otro **HTMLElement**.

Uso:

```
[element].append([object or element]);
```

Ejemplo:

```
// This will create a QCObjects class named "canvas" extending a HTMLCanvasElement with
a customAttr property that has a "custom" value
Class('canvas',HTMLCanvasElement,{
  customAttr:'custom'
});

// This will declare an instance canvas1 from the class canvas
let canvas1 = New(canvas,{
            width:100,
            height:100,
          });

// This will append the canvas1 object to the document body
document.body.append(canvas1);
```

El método _super_

Cuando extiendes una clase QCObject desde otra, puedes usar _super_ metodo para tener una instancia desde la definición de la clase central.

Uso:

```
_super_('MySuperClass','MySuperMethod').call(this,params)
// where this is the current instance and params are method parameters
```

Ejemplo:

```
Class('MySuperiorClass',InheritClass,{
  propertyName1:0, // just to declare purpose
  propertyName2:'',
  classMethod1: function (){
    // some code here
    // note you can use "this" object
    return this.propertyName1;
  },
});
```

```
Class('MyClassName',MySuperiorClass,{
  propertyName1:0, // just to declare purpose
  propertyName2:'',
  classMethod2: function () {
    // The next line will execute classMethod1 from MySuperiorClass
    // but using the current instance of MyClassName1
    return _super_('MySuperiorClass','classMethod1').call(this);
  }
});

var newObject = New(MyClassName,{
    propertyName1:1, // this initializes the value in 1
    propertyName2:"some value"
});
console.log(newObject.classMethod2()); // this will show the number 1
```

New

Crea una instancia de objeto de una definición de clase de QCObject.

Uso:

```
let objectInstance = New(QCObjectsClassName, properties);
// where properties is a single object with the property values
```

NOTA: En las propiedades del objeto puedes usar un solo valor o un captador también pero, solo se ejecutaran una vez.

Ejemplo:

```
Class('MyCustomClass',Object);
let objectInstance = New(MyCustomClass,{
  prop1:1,
  get randomNumber(){ // this getter will be executed once
    return Math.random();
  }
});

console.log(objectInstance.randomNumber); // it will show
console.log(objectInstance.prop1); // it will show number 1
```

InheritClass

Una sola definición de clase común QCObjects es utilizada.

ClassFactory

Usa la **ClassFactory** para tener una declaración de clase de fabrica para la clase, también puedes usar la clase de fabrica desde un paquete o desde la fila apilada.

Para recuperar la clase de fabrica de la clase fila apilada simplemente usa el nombre de la clase llamándola directamente en el código.

Ejemplo:

```
/* When you declare MyClass using Class() it is instantly added to the Class queue
stack
* and you can get the factory either using ClassFactory() or calling the name MyClass
straight in the code
*/
Class('MyClass',{
    a:1
})
console.log(MyClass == ClassFactory('MyClass')) // it will show true
```

```
/* On the other hand, ClassFactory() will be so useful when you define a Class into a
Package
*/
Package('org.quickcorp.package1',[
    Class('MyClass',{
        a:1
    })
])
console.log(MyClass == ClassFactory('MyClass')) // it will still show true
// The following line will show true as well
console.log(MyClass == ClassFactory('org.quickcorp.package1.MyClass'))
```

```
/* The interesting thing is when you have declared more than one Class using the
* same name MyClass into different packages but with different property default values
* and even properties
*/
Package('org.quickcorp.package1',[
    Class('MyClass',{
        a:1
    })
])
Package('org.quickcorp.package2',[
    Class('MyClass',{
        a:2,
        b:1
    })
])
// The last declaration of MyClass will be the one survival in the Class queue
// so the reference MyClass in the code will point to that one
console.log(MyClass == ClassFactory('MyClass')) // it will still show true

// In this case as the MyClass defined in the org.quickcorp.package1 will not be the
same
// as the one in the org.quickcorp.package2, but the MyClass in the package2 is the
last one
// The following line will show false
console.log(MyClass == ClassFactory('org.quickcorp.package1.MyClass'))

// The following line will show true
console.log(MyClass == ClassFactory('org.quickcorp.package2.MyClass'))
```

```
// The following line will show false
console.log(ClassFactory('org.quickcorp.package1.MyClass') ==
ClassFactory('org.quickcorp.package2.MyClass'))
```

Los ejemplos anteriores están intencionalmente hechos para explicar y mostrar como el alcance de la definición de clase en QCObjects es protejida, llevada y reflejada en una ClassFactory.

Así que vas a querer usar la ClassFactory cuando necesites completar un control sobre el alcance cuando se extienden las Clases.

Ejemplo

```
// When normally you extend a Class using the Class queue you do:
Class('MyExtendedClass',MyInheritClass,{
    extendedProp1: 'value of prop',
    extendedProp2: 2
})
```

```
/* But to protect the scope from misleading by reference, you can asure that
MyInheritClass
is the one you want to extend by declaring it into a package and then extend it
*/
Package('org.quickcorp.mypackage1',[
    Class('MyInheritClass',{
        sourceProp:1
    }),
])

// The following code is a definition of MyExtendedClass into a different package
// org.quickcorp.package2
// extending MyInheritClass using ClassFactory to retreive the Class from the source
package
// org.quickcorp.mypackage1
Package('org.quickcorp.mypackage2',[
    Class('MyExtendedClass',ClassFactory('org.quickcorp.mypackage1.MyInheritClass'),{
        extendedProp1: 'value of prop',
        extendedProp2: 2
    })
])

// this will show the number 1 (as the inherited default value of sourceProp)
console.log(New(MyExtendedClass).sourceProp)
```

_Crypt

Con _Crypt puedes codificar en serie objetos a passphrase

Ejemplo (1):

```
var _string = New(_Crypt,{string:'hello world',key:'some encryption md5 key'});
console.log(_string._encrypt());
console.log(_string._decrypt()); // decodes encrypted string to the source
```

Ejemplo (2):

```
_Crypt.encrypt('hola mundo','12345678866');
_Crypt.decrypt('nqCelFSiq6Wcpw==','12345678866');
```

GLOBAL

GLOBAL es una clase especial de QCObject para conseguir alancé global. Tiene un conjunto y consigue un método que te ayude a manejar propiedades internas Globales.

Ejemplo:

```
GLOBAL.set('globalProperty1','some value in global scope');
var globalProperty1 = GLOBAL.get('globalProperty1');
```

CONFIG

CONFIG es una clase inteligente que maneja los ajustes generales de tu aplicación. Puedes tener las propiedades ya sea desde config.json o desde la memoria previamente guardado en la llamada set().

Uso desde memoria:

1.- En su código inicial, configura los valores iniciales de CONFIG:

```
CONFIG.set('someSettingProperty','some initial value');
```

2.- Luego puede acceder a él desde cualquier parte de su código utilizando el método get:

```
var someSettingProperty = CONFIG.get('someSettingProperty');
```

Uso desde config.json:

1.- Necesitas indicar primero que estas usando el archivo config.jso mediante el ajuste "useConfigService" el valor para la verdad.

```
CONFIG.set('useConfigService',true); // using config.json for custom settings config
```

2.-Una vez peparaste el valor anterior QCObjects lo sabra y mirara el siguiente ajuste dentro del archivo config.json en la carpeta basePath de tu aplicación.

Uso desde config.json encriptado:

También existe una forma de usar el archivo encriptado config.json con el fin de proteger tus ajustes, robots que pueden robar tu data no protegida desde la aplicación web (como las llaves de arrastre API)

El archivo encriptado json va en https://config.qcobjects.dev, pon tu configuración dominantes y tu contenido config.json. La herramienta encriptará tu json y podrás copiar el contenido incriptado para insertarlo en tu archivo config.json. QCObjects sabrá si la data esta encriptada y tu proceso para decodificar la data sera mas transparente para ti.

Propiedades dinámicas de CONFIG

A veces necesitaras establecer el valor de la fuente que no sea estática, como el ENV vars u otras fuentes personalizadas de data dinámica. Para tener valor usando CONFIG de una fuente dinámica tienes que usar un procesador. Existen procesadores comunes predefinidos como $ENV (solo si esta disponible en CLI, Collab o Node) y $config (disponible en todos los ambientes).

Los procesadores son llamados como una meta de valor ya sea en el config.json o en la Clase CONFIG.

```
// file: config.json
{
    "domain":"localhost",
    "env1":"$ENV(ENV1)",
    "customSettings":{
        "value1":"$config(domain)"
    }
}
```

```
let value1 = CONFIG.get("customSettings").value1;
// value1 = "localhost";

let env1 = CONFIG.get("env1");
//env1 = (environment variable ENV1)
```

```
// sets the key "api_key" of the CONFIG settings to a dynamic processor $ENV that
recovers the value of API_KEY from the environment variables
CONFIG.set("api_key","$ENV(API_KEY)");

let api_key = CONFIG.get("api_key");
// api_key will contain the value of the system API_KEY environment var
// ($ENV processor returns a valid value only on Node.js , QCObjects CLI and QCObjects
Collab engine)
```

Processor

La clase estatica que usa para establecer el procesador personalizado para CONFIG.

Uso:

```
Processor.setProcessor(processor)
```

Donde **processor** es el nombre de la función que recibe el argumento del procesador

Ejemplo:

Tienes que usar ese valor en tus ajustes de configuración en el valor **serviceURL** pero también necesitas configurar el **host** y los ajustes de **port** usando el valor analizado de esa url. Para analizar el valor de el ambiente SERVICE_URL la variable bajo demanda y rellenarlo con los ajustes de configuración en tu config.json, tu config.json se vera algo así:

```
// file: config.json
{
   "serviceURL":"$ENV(SERVICE_URL)",
   "host":"$SERVICE_HOST(SERVICE_URL)",
   "port":"$SERVICE_PORT(SERVICE_URL)"
}
```

El **$SERVICE_HOST** y el **$SERVICE_PORT** procesadores no existen. Para definirlos tienes que usar:

```
// execute the next code in your init.js file or before to load the CONFIG settings
```

```
let SERVICE_HOST = function (arg){
    var processorHandler = this; // to make this always works, do not use arrow
functions to define your
    let serviceURL = new URL(processorHandler.processors.ENV(arg));
    return serviceURL.host;
}
let SERVICE_PORT = function (arg){
    var processorHandler = this; // to make this always works, do not use arrow
functions to define your
    let serviceURL = new URL(processorHandler.processors.ENV(arg));
    return serviceURL.port;
}

Processor.setProcessor(SERVICE_HOST);
Processor.setProcessor(SERVICE_PORT);
```

Entonces solo necesita establecer su variable de entorno SERVICE_URL en su shell

Lo siguiente es para los sistemas operativos de Unix/Linux :

```
export SERVICE_URL="https://example.com:443/path-to-a-resource/"
```

Y su configuración se cargará dinámicamente de esta manera:

```
{
    "serviceURL":"https://example.com:443/path-to-a-resource/",
    "host":"example.com",
    "port":"443"
}
```

Y consigue tus valores correspondientes en **CONFIG.get(value)**

waitUntil

WaitUntil es un ayudante solo en caso de que estés en problemas tratando de correr el código antes de que la condición sea real. El código dentro del waitUntil sera ejecutado una vez.

NOTA: Esto es útil en algunos casos pero no es recomendado para uso excesivo.

Uso:

```
waitUntil(()=>{
  // the code that will be executed after the condition is true
},()=>{return condition;});
// where condition is what I want to wait for
```

Ejemplo:

```
let someVar = 0;
waitUntil(()=>{
  console.log('someVar is present');
},()=>{return typeof someVar != 'undefined';});
// where condition is what I want to wait for
```

Package

Define el paquete de QCObjects y regrésalo.

Uso:

```
Package('packageName',[packageContent]);
```

Donde packageContent es una gama de clases de QCObjects. Si solo pasas el packageName param conseguirás el contenido declarado anteriormente.

Ejemplo (1):

```
'use strict';
Package('org.quickcorp.main',[
  Class('Main',InheritClass,{
    propertyName1:'propertyValue1',
  }),
  Class('MyCustomClass',InheritClass,{
    propertyName2:'propertyValue2',
  }),
]);
```

Ejemplo (2):

```
let mainPackage = Package('org.quickcorp.main'); // this will return the previously
declared content of package 'org.quickcorp.main'
// mainPackage[0] will be the Main class definition.
// This is useful for code introspection
```

La técnica de carga de paquetes presente en QCObjects esta basada en una promesa y orientada al alcance. Puedes preguntar si un paquete fue cargado simplemente llamando la función Packege() pasando el nombre del paquete a un argumento.

Import

Importa un paquete desde otro archivo JS.

Uso:

```
Import (packagename,[ready],[external]);
```

Donde el packagename es el nombre del paquete, listo es una función que podrá ser ejecutada después de que el paquete es cargado y el externo es un valor bolean que indica si el archivo JS esta en el mismo origen o esta desde otro recurso externo.

Ejemplo (1):

```
Import('org.quickcorp.main');
```

El código anterior intentara importar un archivo JS llamado 'org.quickcorp.main.js' desde un camino especifico en el valor de ajuste **relativeImportPath** presente en tu **CONFIG**. Dentro del archivo tienes que definir el paquete mediante el uso del paquete ('org.quickcorp.main',[Class1, Class2...])

Ejemplo (2):

```
Import('org.quickcorp.main',function (){
  console.log('remote import is loaded');
},true);
```

El código anterior esta vez esta tratando de cargar en el mismo paquete usando el camino externo mediante el **remoteImportsPath** ajustes presentes en tu **CONFIG**

NOTA: En los dos ejemplos anteriores no necesitas usar o especificar la extensión ".js". Esto esta usado por defecto y no puede ser cambiado por razones de seguridad.

Export

Pon un símbolo(barra o función) en el alcance global.

Uso:

```
Export('name of symbol');
```

Ejemplo:

```
(()=>{
  // this is local scope
  let someFunction = (someLocalParam)=>{
    console.log(someLocalParam);
  };
  Export(someFunction); // now, someFunction is in the top level scope.
})();


// this is the top level scope
someFunction('this works');
```

Cast

Usa el método Cast de cualquier elemento DOM para obtener las propiedades de otro tipo de objeto. Esto es útil para trasformar un tipo objeto a otro otorgándole mas flexibilidad a tu código.

Uso:

```
let resultObject = [element or QCObjects type].Cast(objectToCastFrom);
```

Donde objectToCastFrom es un objeto para obtener propiedades desde y poner el objeto resultante regresado por el cast.

Ejemplo:

```
Class('MyOwnClass',{
  prop1:'1',
  prop2:2
});

let obj = document.createElement('div').Cast(MyOwnClass);
```

El código anterior creara un objeto DOM y lo emitirá a MyOwnClass. Gracias a que MyOwnClass es un tipo clase de QCObject el objeto ahora tendrá propiedades prop1 y prop2 y ahora sera una instancia de objeto QCObject con una propiedad del cuerpo que es un elemento div.

Tag

Etiquetar es una función útil para seleccionar cualquier elemento DOM usando selectores. Etiquetar siempre regresa a la lista de elementos que puedas mapear, ordenar y filtrar como cualquier otra lista.

Uso:

```
var listOfElements = Tag(selector);
```

Donde el selector es un DOM selector de respuestas.

Ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Demo</title>
        <script type="text/javascript" src="https://qcobjects.dev/QCObjects.js">
</script>
    </head>
    <body>
    <div class="myselector">
    <p>Hello world</p>
    </div>
    <script>
    Ready(()=>{
      Tag('.myselector > p').map((element)=>{
        element.innerHTML = 'Hello world! How are you?';
      });
    });
    </script>
    </body>
</html>
```

En el código anterior el elemento párrafo fue creado dentro de un div con una clase css llamada myselector mediante html y luego es modificada dinámicamente usando la función etiqueta de QCObject. Si estas familiarizado con un framework selector query te encantara este.

Ready

Asigna una función para correr después de que todo esta hecho mediante QCObject y después del evento window.onload. Úsalo para prevenir un error de objeto DOM 'indefinido'.

Uso:

```
Ready(()=>{
  // My init code here!
});
```

Tenga en cuenta que si define los componentes dinámicos mediante el uso de una etiqueta "componente" en HTML, el contenido dinámico no activara eventos listos. Para atrapar el código cada vez que se carga un componente dinámico usa un método decontrolado hecho en su lugar.

Usaras implementación lista principalmente cuando quieras implementar QCObjects en conjunto con otro framework que lo necesite.

Component Class

Un tipo clase de QCObject por componentes.

Propiedades

[Component].domain Regresa una cadena con el dominio de tu aplicación. Se establece automáticamente por QCObjects al momento de cargar.

[Component].basePath Regresa una cadena con un camino url base de tu aplicación. Se establece automáticamente por QCObjects al momento de cargar.

NOTA: Si quieres cambiar los componentes con base en un camino, tienes que usar *CONFIG.set('componentsBasePath','new path relative to the domain')* en tu unidad de código.

[Component].templateURI Es una cadena representando un componente plantilla URI relativo al dominio. Cuando este listo, el componente sera cargado a la plantilla y lo agregara al contenido interno dentro del cuerpo childs como parte del DOM. Para establecer esta propiedad es recomendado usar la función ayudante ComponentURI.

[Component].tplsource Es la cadena representando a la fuente donde el template sera cargado. Puede ser "default" o "none". El valor "default" le dirá a QCOBject que cargue el template desde el contenido templateURI. El valor "none" le dirá a QCObject que no cargue el template desde ningún lado.

[Component].url Es una cadena representando una url completa de un componente. Es automáticamente establecido por QCObjects cuando un componente es instanciado.

[Component].name Es una cadena que representa el nombre de un componente. El nombre de un componente puede ser cualquier valor alphanumerico que identifique un tipo de componente. Sera internamente utilizado mediante ComponentURI para construir un tempalte URI normalizado.

[Component].method Es una cadena que representa a un método HTTP o HTTPS. Por defecto cada componente esta configurad para usar el método "GET". En la mayoría de los casos no necesitas cambiar esta propiedad.

[Component].data Es un objeto que representa los datos del componente. Cuando QCObject carga un template este conseguirá cada propiedad de los datos y la atará al nivel de template representado por la misma propiedad dentro del contenido del template entre los brackets dobles, ejemplo: (example: {{prop1}} in the template content will represent data.prop1 in the component instance). NOTA: Para refrescar los enlaces de datos para reconstruir el component (mira el uso de [Component].rebuild() para mas detalles).

[Component].reload Es un valor boolean el que dice cuando qcobjects es obligado a recargar el contenido de un componente en el template o no. Si el valor es cierto, el contenido del template sera remplazado por los actuales hijos DOM del elemento cuerpo. Si este valor es falso, el contenido del template sera añadido después de los hijos del componente cuerpo.

[Component].cached Es un valor booleado el que le dice a QCObject si el componente necesita ser atrapado o no. Cuando el componente es atrapado, el contenido del template es cargado desde templateURI sera cargado una sola vez. Puedes configurar esta propiedad incluso como una propiedad estática de un componente de clase para configurarlo como un valor predeterminado para cada siguiente componente instancia de objeto o configurarlo de manera individual el valor de la propiedad en cada definición de

componente. En un mundo donde el desempeño cuenta, para darle mas flexibilidad al comportamiento del cache es necesitado mas que nunca.

[Component].routingWay Regresa una cadena representando la forma de enrutamiento. Un valor puede ser "hash", "pathname" o "search". NOTA: Para cambiar el routingWay de cada componente es recomendado usar CONFIG.set('routingWay','value of a valid routing way')en tu unidad de código.

[Component].validRoutingWays Regresa a la lista que representa la forma de enrutamiento. QCObject usa esto internamente para validar el routingWay el cual usa para construir los enrutamientos de componente.

[Component].routingNodes Regresa al objeto NodeList representando la lista de nodes que fueron cargados por el creador de enrutamientos.

[Component].routings Regresa a la lista con los componentes complicados cuando el componente fue instanciado.

[Component].routingPath Regresa una cadena que representa el camino del enrutamiento actual.

[Component].routingSelected Regresa un objeto que representa el componente de enrutamiento actual.

[Component].subcomponents Regresa a la lista de componentes que son childs de las instancias de componentes.

[Component].body Es un elemento DOM que representa el cuerpo del componente NOTA:Cada vez que un cuerpo es configurado, sera activado el generador de rutas para este componente.

Métodos

[Component].set('prop',value) Establece un valor para una propiedad de componente.

[Component].get('prop') Devuelve el valor de una propiedad componente

[Component].rebuild() Reconstruye un componente. Forzará una llamada para el cargador de componentes con este componente cuando sea necesario.

[Component].Cast(ClassName or ComponentClassName) Devuelve el reparto de una definición de componente en otra. Esto es útil para combinar dinámicamente definiciones de componentes.

[Component].route() Fuerza al generador de rutas de componentes a recargar las rutas del componente. Esto resultara en una reconstrucción de llamada cuando sea necesario.

[Component].fullscreen() Pone el componente en modo de pantalla completa.

[Component].closefullscreen() Cierra el modo de pantalla completa.

[Component].css(css object) Establece las propiedades del css para el componente.

[Component].append(component or QCObjects object) Agrega un componente como hijo del cuerpo del componente actual

[Component].attachIn(selector) Adjunta un cuerpo de componente actual a cualquier elemento en el selector dado.

Tag HTML Component

Una etiqueta HTML es una representación de una instancia de componente. Cada declaración de una etiqueta `<component></component>` generara una instancia relacionada con un componente QCObjects. Mientras una etiqueta de componente no es una instancia por si misma, incluso puedes definir algunas propiedades de instancia configurando el atributo de etiqueta relacionado cuando esté disponible.

Atributos disponibles

A continuación se muestra una lista de los atributos disponibles para una etiqueta de componente

El atributo name

`<component name>` Establece el nombre de la instancia de componente relacionada creada por QCObjects.

USO:

```
<component name="name_of_component"></component>
```

EJEMPLO:

```
<!-- index.html -->
<!DOCTYPE html>
<html>
    <head>
        <title>Demo</title>
        <script type="text/javascript" src="https://qcobjects.dev/QCObjects.js">
</script>
    </head>
    <body>
      <!-- this will load the contents of ./templates/main[.tplextension] file -->
      <component name="main"></component>
    </body>
</html>
```

El atributo cached

`<component cached>` Establece la propiedad en caché si la instancia relacionada de un componente.

NOTA: Solo se puede establecer un valor de "verdadero" para indicar a QCObjects que el contenido de la plantilla del componente debe almacenarse en caché. Cualquier otro valor sera interpretado como falso.

USO:

```
<component name="name_of_component" cached="true"></component>
```

La declaración de la propiedad data

`<component data-property1 data-property2 ...>` Configura un valor estatico de las propiedades de los datos en una instancia de componente.

NOTA: La declaración de la etiqueta de propiedad de datos se pensó con el propósito de dar una forma simple de hacer mocking de un componente dinámico con asignaciones de Template. No lo uses pensando que es una forma bidireccional de datos. Mientras puedas tener un comportamiento de forma bidireccional accediendo a los objetos de datos desde una instancia de componente, no es lo mismo que la etiqueta de componente. La declaración de propiedad de datos en una etiqueta de componente es solo una forma de unión de datos debido a los componentes de arquitectura de árbol.

El atributo controllerClass

`<component controllerClass>` Define un controlador de clase personalizado desde una instancia de componente.

USO:

```
<component name="name_of_component" controllerClass="ControllerClassName"></component>
```

El atributo viewClass

`<component viewClass>` Define una vista de clase personalizada para la instancia del componente

USO:

```
<component name="name_of_component" viewClass="ViewClassName"></component>
```

El atributo componentClass

`<component componentClass>` Define un componente de clase personalizado para la instancia de un componente.

USO:

```
<component name="name_of_component" componentClass="ComponentClassName"></component>
```

El atributo effecClass

`<component effectClass>` Define un efecto de clase personalizado para la instancia del componente

USO:

```
<component name="name_of_component" effectClass="EffectClassName"></component>
```

El atributo template-source

`<component template-source>` Establece el tplsource de una propiedad relacionada con una instancia de un componente Posiblemente los valores son "none" o"default".

USO:

```
<component name="name_of_component" template-source="none"></component>
```

El atributo tplextension

`<component tplextension>` Establece la propiedad de tplextension relacionada con una instancia de un componente. Posibles valores son cualquier archivo de extensión. El valor por defecto es "html"

USO:

```
<component name="name_of_component" tplextension="tpl.html"></component>
```

ComponentURI

Es una función ayudante la que te deja definir el templateURI por un componente en una forma normalizada

Ejemplo:

```
var templateURI = ComponentURI({
  'COMPONENTS_BASE_PATH':CONFIG.get('componentsBasePath'),
  'COMPONENT_NAME':'main',
  'TPLEXTENSION':"tpl.html",
  'TPL_SOURCE':"default"
});

console.log(templateURI); // this will show something like
"templates/components/main.tpl.html" depending on your CONFIG settings
```

componentLoader

Carga una instancia de componente en un nivel bajo y agrega el tempalte del componente a el cuerpo del componente. En la mayoría de los casos no necesitara llamar al componentLoader con el fin de cargar un componente. Esto es automáticamente llamado por QCObjects cuando sea necesario. ComponentLoader regresa una promesa que es resuelta cuando el componente se ha cargado y rechazado cuando el componente fallo.

Uso:

```
[Promise] componentLoader(componentInstance,load_async)
```

Donde componentInstance es una instancia de componente creada por `New(ComponentDefinitionClass)`

Ejemplo:

```
componentLoader(componentInstance,load_async).then(
  (successStandardResponse)=>{
    // component load successful
    var request = successStandardResponse.request;
    var component = successStandardResponse.component;
  },(failStandardResponse)=>{
    // component load failed
    var component = failStandardResponse.component;
  });
```

buildComponents

Reconstruye cada componente que sea un elemento child del elemento DOM, quien posee el método. En la mayoría de los casos no necesitaras llamar a builcomponents con la intención de construir o reconstruir todos los componentes del DOM. Esto es automáticamente llamado por QCObjects cuando es necesario.

Uso:

```
[element].buildComponents()
```

Ejemplo:

```
document.buildComponents()
```

Controller

Una clase QCObjects built-in para definir un contorlador.

View

Una vista QCObjects built in para definir una vista.

VO

Una clase QCObjects built-in para definir un valor de objeto.

Service

Un tipo clase QCObjects para servicio.

Propiedades

[Service].domain Regresa a una cadena que domina tu aplicación.Es automáticamente configurado por QCObjects a la hora de carga.

[Service].basePath Regresa a la cadena con un camino base url para tu aplicación. Es automáticamente configurado por QCObjects a la hora de carga.

[Service].url Una cadena representa el url completo del servicio. Puede ser absoluto o relativo para el basepath cuando es aplicado. Puede ser también un url externo.

NOTA: Para cargar un servicio de un recurso externo necesitaras expecificar el parámetro al verdadero usando serviceLoader.

[Service].name Una cadena representando el nombre de un componente. El nombre de un servicio puede ser de cualquier valor alphanumerico que identifique la instancia del servicio. No es una identificación única, sino solo un nombre descriptivo.

[Service].method Una cadena representando a un metodo HTTP o a HTTPS. Los posibles valores son :"GET", "POST", "PUT", ... cualquier otro sera aceptado mediante servicios de llamadas REST.

[Service].data Es un objeto representando al servicio de datos. Cuando QCObjects carga el servicio recibe una respuesta y lo interpreta como un template. Así que una vez la respuesta del servicio es obtenida, Tomara cualquier propiedad de un objeto de datos y lo atara a una etiqueta template representando la misma propiedad dentro del contenido entre los brackets dobles(Ejemplo: {{prop1}}en el contenido de la plantilla se representará data.prop1 en la instancia de servicio).

[Service].cached Es un servicio boolean el que le dice a QCObjects si la respuesta necesita ser cacheada o no. Cuando el servicio es cacheado el contenido plantilla cargara desde el servicio url que sera cargado de una vez. Necesitas configurar el valor falso para cada instancia de servicio defines para asegurar la carga del servicio desde el recurso pero, no del almacenamiento caché.

Métodos

[Service].set('prop',value) Establece un valor para una propiedad de servicio.

[Service].get('prop') Devuelve el valor de una propiedad de servicio

serviceLoader

Carga una instancia de servicio y regresa a la promesa que es resuelta cuando los servicios han respondido exitosamente a la carga y rechazados cuando falla la respuesta de carga.

Uso:

```
[Promise] serviceLoader(serviceInstance)
```

Ejemplo:

```
Class('MyTestService',Service,{
    name:'myservice',
    external:true,
    cached:false,
    method:'GET',
    headers:{'Content-Type':'application/json'},
    url:'https://api.github.com/orgs/QuickCorp/repos',
    withCredentials:false,
    _new_:()=>{
```

```
      // service instantiated
    },
    done:()=>{
      // service loaded
    }
});
var service = serviceLoader(New(MyTestService,{
  data:{param1:1}
})).then(
  (successfulResponse)=>{
    // This will show the service response as a plain text
    console.log(successfulResponse.service.template);
  },
  (failedResponse)=>{

  });
```

JSONService

Es la definición Buil-in para un servicio de clase JSON

Propiedades

[JSONService].domain Regresa una cadena con la que domina tu aplicación. Es automáticamente configurada por QCObjects a la hora de carga.

[JSONService].basePath Regresa una cadena con camino url base de tu aplicación. Es automáticamente configurada por QCObjects a la hora de carga.

[JSONService].url Una cadena completa representando a todo el servicio url. Puede ser absoluto o relativo al basePath cuando es aplicado. Puede ser tambien un url externo.

NOTA: Para cargar un srvicio de un recurso interno necesitas especificar el parametro externo para verdaderamente usar el serviceLoader.

[JSONService].name Es una cadena representando un componente. El nombre del servicio puede ser cualquier valor alphanumerico que identifique el servicio de instancia. No es una identificación única, sino solo un nombre descriptivo.

[JSONService].method Es una cadena representando el metodo HTTP o HTTPS. Los posible valores son:"GET", "POST", "PUT", ... Cualquier otro sera aceptado mediante servicios de llamadas REST.

[JSONService].data Es un objeto representando al servicio de datos. Cuando QCObjects carga el servicio recibe una respuesta y lo interpreta como un template. Así que una vez la respuesta del servicio es obtenida, Tomara cualquier propiedad de un objeto de datos y lo atara a una etiqueta template representando la misma propiedad dentro del contenido entre los brackets dobles(Ejemplo: {{prop1}}en el contenido de la plantilla se representará data.prop1 en la instancia de servicio).

[JSONService].cached Es un servicio boolean el que le dice a QCObjects si la respuesta necesita ser cacheada o no. Cuando el servicio es cacheado el contenido plantilla cargara desde el servicio url que sera cargado solo una vez. Necesitas configurar el valor falso para cada instancia de servicio defines para asegurar la carga del servicio desde el recurso pero, no del almacenamiento caché.

Métodos

[JSONService].set('prop',value) Configura el Valor para una propiedad de servicio.

[JSONService].get('prop') Regresa el valor de una propiedad de servicio.

Ejemplo:

```
Class('MyTestJSONService',JSONService,{
    name:'myJSONservice',
    external:true,
    cached:false,
    method:'GET',
    withCredentials:false,
    url:'https://api.github.com/orgs/QuickCorp/repos',
    _new_:function (){
      // service instantiated
      delete this.headers.charset; // do not send the charset header
    },
    done:function (result){
      _super_('JSONService','done').call(this,result);
    }
});
var service = serviceLoader(New(MyTestJSONService,{
  data:{param1:1}
})).then(
  (successfulResponse)=>{
    // This will show the service response as a JSON object
    console.log(successfulResponse.service.JSONresponse);
  },
  (failedResponse)=>{

  });
```

ConfigService

Es una definicion de clase Buil-in que carga los ajustes CONFIG desde un archivo config.json

Ejemplo:

```
// To set the config.json file relative url
ConfigService.configFileName='config.json'; // it is done by default
CONFIG.set('useConfigService',true); // using config.json for custom settings config
```

SourceJS

Usa SourseJS como una clase estática que esta ayudándote a cargar dependencias JS externas. Esto es comúnmente usado para cargar librerías externas y que no siga el paquete sintaxis de QCObjet.

Ejemplo:

```
Class("MyNewController",Controller,{
    _new_:function (){
        var controller = this;
        controller.dependencies.push(
            New(SourceJS,{
                external:false,
```

```
                url:'doc/js/my-js-resource.js',
                done:function(){
                    logger.debug("Dependency loaded")
                }})
            );
    }
})
```

SourceCSS

Una clase estatica que es usada para cargar recursos CSS externos.

```
Class("MyNewController",Controller,{
    dependencies:[],
    done (){
        this.dependencies.push(New(SourceCSS,{
            external:false,
            url:CONFIG.get('basePath')+'css/my-css-resource.css'
        }));
    }
});
```

Effect

Effect es una super clase para definir los efectos.

Ejemplo:

```
Class('CustomFade',Effect,{
    duration:500, // milliseconds of duration
    apply: function (){
        // Necesitas la siguiente línea para crear un efecto Fade en runtime
        _super_('Fade','apply').apply(this,arguments);
    }
})
```

Timer

Timer es una implementación especial de **requestAnimationFrame** para emular la creación de instancias de hilo, así puedes manejar runtime paralell processing en una manera un poquito más eficiente.

NOTA: Como depende de la dispoinibilidad de requestAnimationFrame solo funciona en browsers modernos.

Ejemplo:

```
Timer.thread({
        duration:300, // duración en milisegundos
        timing(timeFraction,elapsed){
            return timeFraction; // puedes cambiar esta línea para devolver una función
customizada para timing
        },
        intervalInterceptor(progress){
            if (progress>=100){
                // haz lo que quieras aquí
```

```
            }
        }
});
```

Funciones de lista y matemáticas

ArrayList

Una definición de clase usada para manejar listas

```
let myvar = New(ArrayList,[1,2,3]);
```

ArrayCollection

Una definición extendida para manejo avanzado de colecciones

```
let collection = New(ArrayCollection, {source:[0,1,2]});
```

[ArrayList or Array].unique

Filtra un objeto Array o ArrayList para obtener solo elementos únicos. NOTA: Solo filtra una secuencia de valor único.

```
let my_unique_list = [0,1,2,1,2].unique()
// will result in: my_unique_list = [ 0, 1, 2 ]
```

[ArrayList or Array].table

Esto esta destinado para solo el uso de shell script. Muestra una tabla de valores en una lista.

```
["a","b","c","d"].table()
// it will show
┌─────────┬────────┐
│ (index) │ Values │
├─────────┼────────┤
│    0    │  'a'   │
│    1    │  'b'   │
│    2    │  'c'   │
│    3    │  'd'   │
└─────────┴────────┘
```

[ArrayList or Array].sort

Ordena los elementos del array o lista.

```
let my_sorted_array = [3,3,4,0,2,1].sort()
// my_sorted_array = [ 0, 1, 2, 3, 3, 4 ]
```

```
let my_sorted_list = New(ArrayList,{source:[3,3,4,0,2,1]}).source.sort()
// my_sorted_list = [ 0, 1, 2, 3, 3, 4 ]
```

[ArrayList or Array].sortBy

Ordena una lista de objetos por un valor de propiedad.

```
let my_ordered_list = [
                                                {"b":1,"a":2},
                                                {"b":2,"a":1},
                                                {"b":3,"a":3},
                                              ].sortBy("a")
// it will result in
[
    { b: 2, a: 1 },
    { b: 1, a: 2 },
    { b: 3, a: 3 }
]
```

[ArrayList or Array].matrix

Genera una matriz en una dimensión.

Uso

[].matrix (length, [value]) Donde **length** es un numero de elementos y el **value** opcional es un valor en cada elemento, puede ser cualquier valor de cualquier tipo.

```
let matrix = Array.matrix(10);
// matrix = [
  0, 0, 0, 0, 0,
  0, 0, 0, 0, 0
]
```

```
let matrix = ArrayList.matrix(10,1);
// matrix = [
  1, 1, 1, 1, 1,
  1, 1, 1, 1, 1
]
```

```
let a = 1, b = 2;
let c = ArrayList.matrix(10,{a,b})
// c = [
 { a: 1, b: 2 },
 { a: 1, b: 2 },
 { a: 1, b: 2 },
 { a: 1, b: 2 },
 { a: 1, b: 2 },
 { a: 1, b: 2 },
 { a: 1, b: 2 },
 { a: 1, b: 2 },
 { a: 1, b: 2 },
 { a: 1, b: 2 }
]
```

[ArrayList or Array].matrix2d

Crea una matriz 2D.

```
let matrix2d = ArrayList.matrix2d(2,1);
// [ [ 1, 1 ], [ 1, 1 ] ]
```

[ArrayList or Array].matrix3d

Crea una matriz 3D

```
let matrix3d = ArrayList.matrix3d(3,"a");
// it will result in a 3x3 matrix with the value "a" on every element
[
  [ [ 'a', 'a', 'a' ], [ 'a', 'a', 'a' ], [ 'a', 'a', 'a' ] ],
  [ [ 'a', 'a', 'a' ], [ 'a', 'a', 'a' ], [ 'a', 'a', 'a' ] ],
  [ [ 'a', 'a', 'a' ], [ 'a', 'a', 'a' ], [ 'a', 'a', 'a' ] ]
]
```

range

A Python le gusta la función de crear una lista de rango. Puedes usarla en conjunto con ArrayList.matrix, ArrayList.matrix2d y ArrayList.matrix3d para generar rangos complejos de matriz.

Uso

rango (longitud) o rango (initialIndex, finalIndex) range () sin ningún parámetro devuelve una lista vacía range (0) devuelve una lista con un elemento con valor 0

```
logger.debugEnabled=true;

for (var i in range(10)){
    (!isNaN(i) && logger.debug(i))
}

// the above code will show
[DEBUG] 0
[DEBUG] 1
[DEBUG] 2
[DEBUG] 3
[DEBUG] 4
[DEBUG] 5
[DEBUG] 6
[DEBUG] 7
[DEBUG] 8
[DEBUG] 9
[DEBUG] 10
```

```
logger.debugEnabled=true;

// same result will be obtained iterating the range first
for (var i in {...range(10)}){
    logger.debug(i)
}

// the above code will show
[DEBUG] 0
```

```
[DEBUG] 1
[DEBUG] 2
[DEBUG] 3
[DEBUG] 4
[DEBUG] 5
[DEBUG] 6
[DEBUG] 7
[DEBUG] 8
[DEBUG] 9
[DEBUG] 10
```

```
// a bit shorter syntax for the same result
range(10).map(n=>logger.debug(n))
```

```
let normalizedMatrix = ArrayList.matrix(3,range(2));
// normalizedMatrix = [ [ 0, 1, 2 ], [ 0, 1, 2 ], [ 0, 1, 2 ] ]
```

```
let my3dmatrix = ArrayList.matrix3d(3,range(0,1));
// my3dmatrix will be
[
  [
    [ [0, 1], [0, 1], [0, 1] ],
    [ [0, 1], [0, 1], [0, 1] ],
    [ [0, 1], [0, 1], [0, 1] ]
  ],
  [
    [ [0, 1], [0, 1], [0, 1] ],
    [ [0, 1], [0, 1], [0, 1] ],
    [ [0, 1], [0, 1], [0, 1] ]
  ],
  [
    [ [0, 1], [0, 1], [0, 1] ],
    [ [0, 1], [0, 1], [0, 1] ],
    [ [0, 1], [0, 1], [0, 1] ]
  ]
]
```

Array.sum

Suma los elementos de una matriz.

```
let s = [1,2,3].sum()
// s = 6
```

Array.avg

Calcula el valor promedio de los elementos en el Array

```
let average = [10,5].avg()
// average = 7.5
```

Array.min

Devuelve el valor mínimo de los elementos de un Array.

```
let minValue = [1,2,3].min()
// minValue = 1
```

Array.max

Devuelve el valor máximo de los elementos de unArray

```
let maxValue = [1,2,3].max()
// maxValue = 3
```

SDK

SDK Components

org.quickcorp.components.ShadowedComponent

La Clase **ShadowedComponent** es un componente personalizado diseñado para permitirte crear un componente usando el Shadow DOM de un buscador. Lee mas sobre los componentes Shadow en [Este articulo en Hackernoon] (https://www.hackernoon.com/shadowed-components-and-qcobjects-kd703yld).

Uso:

```
<component componentClass="ShadowedComponent"></component>
```

org.quickcorp.components.FormField

FormField es una clase par los componentes personalizados**QCObjects** que te permiten que le permite inyectar un comportamiento genérico de campo de formulario a sus componentes. Tiene una característica inversa data-binding para detectar valores de los campos DOM dentro de tu formulario y asignarlos a los valores de datos de tu componente. De esta forma no perderás el desempeño creando un data binig bidireccional a la antigua basado en observables. Para implementar este comportamiento avanzado, necesitas hacer lo siguiente:

1.- Asigna un atributo **data-field** a la etiqueta DOM dentro del cuerpo del componente con el nombre del campo correspondiente en tu objeto de datos.

2.- A tu **component tag**, asígnale **FormField** en el atributo **componentClass**.

3.- Para recuperar los datos del formulario dentro de tu componente solo tienen que usar el componentInstance.data . Cada propiedad del objeto componentInstance.data sera unido por los eventos bindig con ls propiedades del valor en cada objeto DOM del formulario que ha sido asignado al data-field.

Uso:

```
<!-- Where you place the component -->
<component name="myform" componentClass="FormField"></component>
```

```
<!-- template: myform.tpl.html -->
<label for="email"><b>Email</b></label>
<input data-field="email" type="email" placeholder="Enter Email" name="email" required>

<label for="psw"><b>Password</b></label>
<input data-field="name" type="text" placeholder="Enter Your Name" name="name"
required>
```

data-field="name" será emparejado con **this.data.name** dentro de la clase de componente y se actualizará cada vez que se active un evento de enlace de datos. Lo mismo pasa con **data-field="email"** y así.

FormField.executeBindings():

El método **executeBindings** del componente FormField donde encontraras los valores de atributo **data-field** y emparejaras con ellos los correspondientes archivos **data** en una instancia de componente.

Data Binding Event Change:

Dentro del cuerpo de su componente, cuando es un componente **FormField** cada vez que el DOM envía un evento de "cambio", activará el método executeBindings de su componente.

Data Binding Event Blur:

Dentro del cuerpo de su componente, cuando es un componente **FormField** cada vez que el DOM envía un evento "Blur", activará el método executeBindings de su componente.

Data Binding Event Focus:

Dentro del cuerpo de su componente, cuando es un componente**FormField** cada vez que el DOM envía un evento "Focus", activará el método executeBindings de su componente.

Data Binding Event Keydown:

Dentro del cuerpo de su componente, cuando es un componente **FormField** cada vez que el DOM envía un evento "Keydown", activará el método executeBindings de su componente.

org.quickcorp.components.ButtonField

ButtonField es una subdefinición de **FormField** que es comunmete usando para casi el mismo proposito del FormField.La principal diferencia entre ButtonField y FormField es que ButtonField tiene un elemento DOM ** `<button>` ** como el cuerpo del componente por defecto. Y FormField no tiene un cuerpo predefinido.

Uso:

```
<component name="name_of_component" componentClass="ButtonField"></component>
```

org.quickcorp.components.InputField

InputField es una subdefinición de **FormField**, que es comunmete usando para casi el mismo proposito del FormField. La principal diferencia entre InputField y FormField es que InputField tiene un elemento DOM **`<input>`** como el cuerpo del componente por defecto. Y FormField no tiene un cuerpo predefinido.

Uso:

```
<component name="name_of_component" componentClass="InputField"></component>
```

org.quickcorp.components.TextField

ButtonField es una subdefinición de **FormField**, que es comunmete usando para casi el mismo proposito del FormField. La principal diferencia entre InputField, ButtonField y FormField es que ButtonField tiene un elemento DOM **`<textarea>`** como el cuerpo del componente por defecto. Y FormField no tiene un cuerpo predefinido.

Uso:

```
<component name="name_of_component" componentClass="TextField"></component>
```

org.quickcorp.components.EmailField

EmailField es una subdefinición de **FormField**, que es comunmete usando para casi el mismo proposito del FormField. La principal diferencia entre ButtonField y FormField es que ButtonField tiene un elemento DOM `<input>` como el cuerpo del componente por defecto. Y FormField no tiene un cuerpo predefinido.

Uso:

```
<component name="name_of_component" componentClass="EmailField"></component>
```

org.quickcorp.components.GridComponent

GridComponent tiene un nombre predefinido asignado al valor "grid", así que tenlo en cuenta cuando uses esta clase de componente. Tambien GridComponent está diseñado para usarse junto con GridController para expandir su comportamiento a una cuadrícula CSS.

Uso:

```
<component componentClass="GridComponent" ...></component>
```

Ejemplo:

```
<component rows="2" cols="2" componentClass="GridComponent"
controllerClass="GridController">
  <!-- It is recommended to use subcomponents as the Grid elements-->
    <component name="name_of_subcomponent1"></component>
    <component name="name_of_subcomponent2"></component>
    <component name="name_of_subcomponent3"></component>
    <component name="name_of_subcomponent4"></component>
</controller>
```

El ejemplo anterior dibujará una cuadrícula css de dos columnas y dos filas y colocará los subcomponentes en ella.

No olvides este archivo:

```
<!-- file: grid.tpl.html, you can use the grid template either to draw the grid itself
or to draw a loading information -->
<p>Loading grid...</p>
```

org.quickcorp.components.ModalEnclosureComponent

org.quickcorp.components.ModalComponent

org.quickcorp.components.SwaggerUIComponent

Se usa para inyectar un DOM swagger-ui necesario para la Swagger UI API. Obtenga más información en este artículo de QCObjects DevBlog llamado Working with Swagger UI as a QCObjects Component

Uso:

```
<component componentClass="SwaggerUIComponent" controllerClass="SwaggerUIController" >
</component>
```

org.quickcorp.components.splashscreen.VideoSplashScreenComponent

Una potente definición de componente que le permite crear una impresionante pantalla de bienvenida de video para su aplicación web progresiva.

Ejemplo:

```
<!-- Add the splash screen component tag as the first component in your HTML document -
->
<component componentClass="VideoSplashScreenComponent"
    data-background="./img/splash/splashscreen-aqua.png"
    data-video_mp4="./img/splash/splashscreen-aqua.mp4"
    data-video_webm="./img/splash/splashscreen-aqua.webm"
    data-video_ogg="./img/splash/splashscreen-aqua.ogv" duration="5000">
    <img slot="logo" alt="logo" src="./img/logo-qcobjects-white.svg"></div>
</component>
<!-- Then you can put your main component as always -->
<component name="main" cached=true controllerClass="MainController">
</component>
```

SDK Controllers

org.quickcorp.controllers.GridController

GridController está destinado a ser utilizado junto con **GridComponent** para permitir crear una cuadricula CSS para colocar los subcompnentes. Más información org.quickcorp.components.GridComponent

org.quickcorp.controllers.DataGridController

DataGridController tomará los datos de su componente y asignará un conjunto de subcomponentes para completarlo. Esto se usa comúnmente para renderizar una lista dinámica de componentes. Es usando un valor de atributo **subcomponentClass** en la etiqueta **component** para determinar que definición de componente usar para construir y renderizar cada subcomponente. Los datos en cada subcomponente seran completados con el valor de un elemento mapeado a un subcomponente.

Uso:

```
<component name="name_of_component" controllerClass="DataGridController"
subcomponentClass="SubComponentClass">
</component>
```

Ejemplo:

Supón que tienes que representar una lista de perfiles. Cada perfil tiene una foto de perfil, un nombre y un correo electrónico. Deseas usar una tarjeta(card) para representar cada perfil de la lista.

Así que defines un CardComponent para renderizar la foto, el nombre y el email en un elemento en la lista.

```
Class("CardComponent",Component,{
    name:"card", // this will point to templates/components/card.tpl.html
    data:{ // the value of this object will be overriden by DataGridController
        name:"<name of contact>",
        email:"email@example.com",
        profilePicture:"img/photo.png"
    }
})
```

```
<!-- template: card.tpl.html -->
<img src="{{profilePicture}}"/>
<h3>{{name}}</h3>
<p>{{email}}</p>
```

Luego, debes colocar un componente para generar la lista de tarjetas(cards).

```
<component name="loading_list" componentClass="MyListComponent"
controllerClass="DataGridController"
subcomponentClass="CardComponent">
</component>
```

```
<!-- template: loading_list.tpl.html -->
<p>Loading list...</p>
```

Por último, debes definir MyListComponent para asignar los datos dinámicos de la lista.

```
Class("MyListComponent",Component,{
    data:[
        { // the value of this object will be mapped to a subcomponent by
DataGridController
            name:"<name of contact>",
            email:"email@example.com",
            profilePicture:"img/photo.png"
        },
        { // the value of this object will be mapped to a subcomponent by
DataGridController
            name:"<name of contact>",
            email:"email@example.com",
            profilePicture:"img/photo.png"
        },
        { // the value of this object will be mapped to a subcomponent by
DataGridController
            name:"<name of contact>",
            email:"email@example.com",
            profilePicture:"img/photo.png"
        }
    ]
})
```

El componente resultante sera una lista de **CardComponent** con los datos de cada perfil dentro de ellos mediante **DataGridController**.

org.quickcorp.controllers.ModalController

org.quickcorp.controllers.FormValidations

FormValidations se utiliza para manejar validaciones predeterminadas para **FormController**

Uso:

```
FormValidations.getDefault(validationName)
```

Donde **validationName** es el nombre de la validación presente en la propiedad **validations** del **FormController**

org.quickcorp.controllers.FormController

La definición FormController esta destinado a ayudarte a manejar formas dinámicas. Mediante el uso de la sintaxis normalizada de la definición FormController, no tienes que codificar la complejidad de la lógica de un formulario de envío, ya que se atomiza aquí en tres pasos.

- Asigna una serviceClass
- define los formSettings
- Asignar código qr las validaciones de campo

[FormController].serviceClass

Es el nombre de cadena de la definición de clase. FormController cargará esta clase usando la función auxiliar ClassFactory, por lo que el nombre puede ser un nombre completo del paquete más una definición.

[FormController].formSettings

Es un objeto con las propiedades especiales del formulario. La configuración predeterminada es: backRouting:'#', loadingRouting:'#loading', nextRouting:'#signupsuccessful'. Estas configuraciones están destinadas a manejar el comportamiento de flujo del formulario.

loadingRouting es el nombre de la ruta que se activará mientras **serviceClass** comience a llamar hasta ue el servicio de recarga regrese la respuesta.

backRouting es el nombre del enrutamiento que se activará si falla la llamada a **serviceClass**.

nextRouting es el nombre del enrutamiento que se activará cuando la llamada para **serviceClass** finalice bien.

[FormController].validations

Es un objeto con las funciones de ayuda que quieres para definir validando cada campo de formulario. Cuando defines una función de validación para un campo, FormController te preguntara si la ejecución de esa función regresa verdadera antes de enviar el formulario.

Para definir validaciones para los campos del formulario, solo necesitas agregarlas como parte de la propiedad de validaciones.

Uso:

```
    // Let's say you have a form field called "name"
    validations: {
        name (){
            return (fieldName, dataValue, element)=> {
                return [true ... or false condition];
            }
        }, //... add a validation for every form field that you want to be validated
    }
```

Tambien puede usar **FormValidations.getDefault** por simplicidad.

```
    // Let's say you have a form field called "name"
    validations: {
        name (){
            return FormValidations.getDefault('name')
        }, //... add a validation for every form field that you want to be validated
    }
```

[FormController].formSaveTouchHandler

Este metodo esta internamente usado mediante FormController para llamar al serviceClass como una acción de envío. Se vinculará a cualquier evento de clic o toque de cualquier elemento dentro del formulario que tenga asignada una clase CSS ".submit".

Ejemplo:

```
<button class="submit"><p>Send</p></button>
```

Cuando el clic o el toque del anterior botón se active, FormController llamara al servicio definido en **serviceClass**, esta acción se hará por **formSaveTouchHandler**. Si desea cambiar este comportamiento, simplemente anule este método en su controlador personalizado.

A complete example of FormController

A continuación se muestra un ejemplo completo de una definición para un Formulario de registro utilizando FormController.

```
// First, define a service class that will be called on submit.
Class('SignupClientService',JSONService,{
    name:'signup',
    external:true,
    cached:false,
    method:'POST',
    url:Service.basePath+'createaccount',
    withCredentials:false,
    _new_:()=>{
        // service instantiated
    },
    done:(successfulResponse)=>{
        // service loaded

_super_('JSONService','done').call(successfulResponse.service,successfulResponse);
            console.log(successfulResponse.service.JSONresponse);
    }
})
```

```
// To safe extend FormController, we extend first from Controller, then
// we use a defaultController to instance a new FormController
Class('SignupFormController',Controller,{
    serviceClass: 'SignupClientService',
    formSettings:{ /* routings that will be triggered once the serviceClass is called*/
        backRouting:'#',
        loadingRouting:'#loading',
        nextRouting:'#signupsuccessful'
    },
    validations: { /* validation definitions for every field in the form to be
validated before submit */
        name (){
            return FormValidations.getDefault('name')
        },
        email (){
            return FormValidations.getDefault('email')
        },
        comment (){
            return function (fieldName, dataValue, element){
                return (dataValue !== '')?(true):(false);
            }
        }
    },
```

```
    defaulController:null,
    _new_:function (o){
        o.serviceClass = this.serviceClass;
        o.formSettings = this.formSettings;
        o.validations = this.validations;
        // here we instance a defaultController with a New FormController
        // passing the o object declaration coming from the components stack building
process.
        this.defaulController = New(FormController,o);
    },
    done: function (){
        // we define a custom done callback function to inject a custom behavior as
well as calling the defaultController behavior
        logger.debugEnabled=true;
        var controller = this.defaulController;
        try {
            controller.done.call(controller);
        }catch (e){
            logger.debug('Unable to execute default behavior');
        }
    }
})
```

```
<!-- A Shadowed Component to render the signup forms -->
<!-- template: signup-form.tpl.html -->

<form action="#" style="border:1px solid #ccc;border-radius:20px">
    <div class="container">
        <slot name="title">
            <h1>Signup Form</h1>
        </slot>
        <slot name="subtitle">
            <p>Please fill up this form to ask for a quote.</p>
        </slot>
        <hr />
        <slot id="field_control" name="field-control">
        </slot>
        <hr />
        <slot name="submit_button"></slot>
    </div>
</form>
```

```
<!-- A signup form using the shadowed component signup-form -->
<!-- template: signup.tpl.html -->
<component name="signup-form" shadowed="true" controllerClass="SignupFormController">
  <h1 slot="title">Signup Form</h1>
  <p slot="subtitle">Please fill up this form to ask for a quote.</p>
  <label slot="field-control" id="name_label" for="name"><b>&#x1F9D1; Full Name</b>
</label>
  <input slot="field-control" type="text" pattern="^[a-zA-Z]+((['. -][a-zA-Z ])?[a-zA-
Z]*)*$" title="Please write your full name" placeholder="Full Name" name="name" data-
field="name" aria-labelledby="name_label" required>
```

```
  <label slot="field-control" id="email_label" for="email"><b>&#x1F4E7; Email</b>
</label>
  <input slot="field-control" type="email" pattern="^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$"
title="Please write a right email address" placeholder="Enter Email" name="email" data-
field="email" aria-labelledby="email_label" required>
  <label slot="field-control" for="comment" id="comment_label"><b>&#x1F4DD; Comment</b>
</label>
  <textarea slot="field-control" name="comment" title="Please write a comment"
rows="10" cols="100" data-field="comment" aria-labelledby="comment_label"></textarea>
  <p slot="field-control">By submitting this form you agree to our <a href="#"
style="color:dodgerblue">Terms & Privacy</a>.</p>
    <div class="clearfix">
        <button aria-label="Cancel" onclick="location.href='/#'" role="button"
tabindex="-1" type="button" class="cancelbtn"><p>Cancel</p></button>
        <button  aria-label="Send" role="button" tabindex="-1" type="button"
class="signupbtn submit"><p>Send</p></button>
    </div>
</component>
```

org.quickcorp.controllers.SwaggerUIController

Es usado para inyectar un swagger-ui DOM necesario para el Swagger UI API. Aprende mas en este articulo de QCObjects DevBlog llamado Working with Swagger UI as a QCObjects Component

Uso:

```
<component componentClass="SwaggerUIComponent" controllerClass="SwaggerUIController" >
</component>
```

SDK Effects

QCObjects tiene una definicion **Effect** para manejar y producir nuevos efectos y transiciones para los componentes. A continuación hay algunas definiciones de efectos personalizadas que te ayudaran a construir sorprendentes características visuales para tus componentes. Para mejorar el rendimiento, los efectos están cambiando CSS internamente para aplicar el efecto de manera inteligente. Y todo el motor de efectos se basa en la definición **requestAnimationFrame**, lee mas sobre esto aquí

org.quickcorp.tools.effects.Move

Mueve el objeto DOM de una posicion a otra.

Uso:

```
Move.apply(element, xfrom, yfrom, xto, yto)
```

Ejemplo:

```
// The next line will move all the images from (0,0) to (10,10)
Tag('img').map(img => Move.apply(img,0,0,10,10))
```

org.quickcorp.tools.effects.MoveXInFromRight

Mueve un elemento desde el lado derecho en el eje X a la posición original del objeto.

Uso:

```
MoveXInFromRight.apply(element)
```

Ejemplo:

```
// the next line will move every canvas on the document from right side to its original position
Tag('canvas').map(canvas => MoveXInFromRight.apply(canvas));
```

org.quickcorp.tools.effects.MoveXInFromLeft

Mueve un elemento desde el lado izquierdo en el eje X a la posición original del objeto.

Uso:

```
MoveXInFromLeft.apply(element)
```

Ejemplo:

```
// the next line will move every canvas on the document from left side to its original position
Tag('canvas').map(canvas => MoveXInFromLeft.apply(canvas));
```

org.quickcorp.tools.effects.MoveYInFromBottom

Mueve un objeto del DOM desde abajo a su posición original usando el eje Y.

Uso:

```
MoveYInFromBottom.apply(element)
```

Ejemplo:

```
// the next line will move the body of every component named "comp1" from bottom to its original position
Tag('component[name=comp1]').map(componentBody =>
MoveYInFromBottom.apply(componentBody));
```

org.quickcorp.tools.effects.MoveYInFromTop

Mueve un objeto del DOM de arriba a su posición original usando el eje Y.

Uso:

```
MoveYInFromTop.apply(element)
```

Ejemplo:

```
// the next line will move the body of every component named "comp1" from top to its original position
Tag('component[name=comp1]').map(componentBody => MoveYInFromTop.apply(componentBody));
```

org.quickcorp.tools.effects.RotateX

Rota un objeto en el eje X.

Uso:

```
RotateX.apply(element, angleFrom, angleTo)
```

angleFrom y **angleTo** representan un valor de ángulo expresado en grados, comenzando desde 0 (cero) a 360.

Ejemplo:

```
// the next line will rotate in X axis the div called #id from 180 degrees to 240 degrees
Tag('div#id').map(div => RotateX.apply(div, 180, 240));
```

org.quickcorp.tools.effects.RotateY

Rota un objeto en el eje Y.

Uso:

```
RotateY.apply(element, angleFrom, angleTo)
```

angleFrom y **angleTo** representan un valor de ángulo expresado en grados, comenzando desde 0 (cero) a 360.

Ejemplo:

```
// the next line will rotate in Y axis the div called #id from 0 degrees to 270 degrees
Tag('div#id').map(div => RotateY.apply(div, 0, 270));
```

org.quickcorp.tools.effects.RotateZ

Rota un objeto en el eje Z.

Uso:

```
RotateZ.apply(element, angleFrom, angleTo)
```

angleFrom y **angleTo** representan un valor de ángulo expresado en grados, comenzando desde 0 (cero) a 360.

Ejemplo:

```
// the next line will rotate in Z axis the div called #id from 0 degrees to 60 degrees
Tag('div#id').map(div => RotateZ.apply(div, 0, 60));
```

org.quickcorp.tools.effects.Rotate

Rota un objeto en los ejes X, Y, Z. Todos los ejes rotarán en paralelo al mismo tiempo produciendo una percepción de efectos visuales en 3D.

Uso:

```
Rotate.apply(element, angleFrom, angleTo)
```

angleFrom y **angleTo** representan un valor de ángulo expresado en grados, comenzando desde 0 (cero) a 360.

Ejemplo:

```
// the next line will rotate in X, Y and Z axes the div called #id form 0 to 270 degrees
```

```
Tag('div#id').map(div => Rotate.apply(div, 0, 270));
```

org.quickcorp.tools.effects.Fade

Produce un efecto de desvanecimiento al reducir la opacidad del elemento.

Uso:

```
Fade.apply(element, alphaFrom, alphaTo)
```

alphaFrom y **alphaTo** son numeros entre 0 (cero) y 1.

```
// the following line will fade a <b class="header"> element from 0.5 (mid visibility)
to 1 (full visibility)
Tag('b.header').map(header=>Fade.apply(header, 0.5, 1))
```

org.quickcorp.tools.effects.Radius

Redondea la esquina de un elemento.

Uso:

```
Radius.apply(element, radiusFrom, radiusTo)
```

radiusFrom y **radiusTo** son valores numéricos.

Ejemplo:

```
// the next line will round the corners of every image in the document
Tag('img').map(element => Radius.apply(element, 0, 100))
```

org.quickcorp.tools.effects.Resize

Uso:

```
Resize.apply(element, scaleFrom, scaleTo)
```

scaleFrom y **scaleTo** son valores numericos Un valor de 1 es tamaño normal, un valor de 2 es tamaño doble, un valor entre 0 y 1 es una escala pequeña.

Ejemplo:

```
// the next line will make a zoom-out effect on every image in the document
Tag('img').map(element => Resize.apply(element, 2,0))

// the next line will make a zoom-in effect on every image in the document
Tag('img').map(element => Resize.apply(element, 0,1))

// the next line will make a zoom-in-out effect on every image in the document
Tag('img').map(element => Resize.apply(element, 2,1))
```

org.quickcorp.tools.effects.WipeLeft

Hace un efecto de limpieza desde el lado izquierdo al origen del elemento.

Uso:

```
WipeLeft.apply(element, scaleFrom, scaleTo)
```

scaleFrom y **scaleTo** son valores numericos. Un valor de 1 es tamaño normal, un valor de 2 es tamaño doble, un valor entre 0 y 1 es una escala pequeña.

Ejemplo

```
Tag('img').map(element => WipeLeft.apply(element,0,1))
```

org.quickcorp.tools.effects.WipeRight

Hace un efecto de limpieza desde el lado derecho hasta el origen del elemento.

Uso:

```
WipeRight.apply(element, scaleFrom, scaleTo)
```

scaleFrom y **scaleTo** son valores numericos Un valor de 1 es tamaño normal, un valor de 2 es tamaño doble, un valor entre 0 y 1 es una escala pequeña.

Ejemplo

```
Tag('img').map(element => WipeRight.apply(element,0,1))
```

org.quickcorp.tools.effects.WipeUp

Realiza un efecto de limpieza de abajo hacia arriba en el origen del elemento.

Uso:

```
WipeUp.apply(element, scaleFrom, scaleTo)
```

scaleFrom y **scaleTo** son valores numericos. Un valor de 1 es tamaño normal, un valor de 2 es tamaño doble, un valor entre 0 y 1 es una escala pequeña.

Ejemplo

```
Tag('img').map(element => WipeUp.apply(element,0,1))
```

org.quickcorp.tools.effects.WipeDown

Realiza un efecto de limpieza de arriba a abajo en el origen del elemento.

Uso:

```
WipeDown.apply(element, scaleFrom, scaleTo)
```

scaleFrom y **scaleTo** son valores numéricos. Un valor de 1 es tamaño normal, un valor de 2 es tamaño doble, un valor entre 0 y 1 es una escala pequeña.

Ejemplo

```
Tag('img').map(element => WipeDown.apply(element,0,1))
```

SDK Misc Tools

org.quickcorp.tools.canvas.CanvasTool

org.quickcorp.tools.layouts.BasicLayout

SDK Views

A continuación hay un conjunto de vistas predefinidas para uso común.

org.quickcorp.views.GridView

Una definion generica GridView para usar con cuadriculas.

SDK i18n messages

El motor QCObjects i18n le permite definir mensajes personalizados. Obtenga más información en este artículo en el DevBlog llamado [i18n Internationalisation for your Progressive Web Apps]

org.quickcorp.i18n_messages.i18n_messages

Utlizado para llamar al motor i18n.

Uso:

```
  Class('i18n_messages_<custom lang>', i18n_messages,{
        ...
    })
```

Ejemplo

```
'use strict';
// file: js/packages/org.quickcorp.i18n_messages.es.js
Package('org.quickcorp.i18n_messages.es', [
  Class('i18n_messages_es', i18n_messages, {
    messages: [
       // ... your custom language dictionary is here
      {
         "en":"This is a paragraph",
         "es":"Esto es un párrafo"
      },
      {
         "en":"Welcome to my new app",
         "es":"Bienvenido a mi nueva app"
      }
    ]
  }),
  {
        // the next line generates an instance of the i18n engine and attaches it
automatically in the package
    _i18n_messages_es: New(i18n_messages_es)
  }
]);
```

The QCObjects HTTP2 Built-In Server

El servidor HTTP2 Built-In de QCObjects te ayudara a probar tu aplicacion en un entorno local antes de implementarlo en un entorno de producción. Para el entorno de producción, se recomienda utilizar el servidor QCObjects HTTP2 Built-In en Ubuntu 18.x o el mas nuevo y NodeJS 12.x o mas actual.

Start serving your files with QCObjects

Para comenzar a utilizar el servior de QCObjects HTTP2 Built-In,solo necesitas ir a la ruta del proyecto y ejecutar la ssiguiente liniea de comandos en tu bash shell:

```
> qcobjects launch mynewapp
```

o

```
> qcobjects-server
```

Comenzara a servir los archivos dentro de la carpeta donde estas, usando el servidor built-in HTTP2 Server cin la configuracion predeterminada.

Principals of an N-Tier or Multitier architecture

QCObjects fue diseñado para trabajar en un entorno profesional. Hay muchas formas y paradigmas para usar cuando define su arquitectura de software. Se recomienda es usar una arquitectura Multitier o N-Tier.

Los beneficios de la arquitectura Multitier o N-Tier son la escalabilidad y confiabilidad de los sistemas que exigen un mayor impacto y rendimiento. Para profundizar en estos conceptos sería innecesario ampliar este documento de referencia, pero puede leer más sobre estos conceptos en los siguientes enlaces externos(solo para referencia y estudio):

- Arquitectura Miltitier
- Arquitectura 3 Tier
- Aplicacion Milti Tier
- Conceptos del sistema de arquitectura N Tier y Tips

Micro-services Principals

El objetivo principal de un microservicio es que puede compactar un fragmento de funcionalidad de back-end en un código que se puede llamar de forma remota desde otro terminal de back-end o frontend. Básicamente, puedes dividir un servicio de back-end de alto nivel en múltiples microservicios pequeños que pueden completar la tarea. Hay miles de buenos ejemplos de este tipo de adopción de patrones. Puede leer más sobre este concepto en los siguientes enlaces externos (solo para referencia y estudio):

- Microservice Patterns
- Microservices on Wikipedia

Con QCObjects puedes codificar sus microservicios de una manera más elegante, limpia y rápida.

Backend settings in config.json

También puedes usar config.json en el lado del backend para hacer algunos ajustes y configuraciones personalizadas para el backend, especialmente útil para hacer las rutas de microservicio.

A continuación se muestra un ejemplo de un archivo config.json personalizado que incluye la configuración del back-end:

```
{
  "documentRoot":"/home/qcobjects/projects/mynewapp/",
  "relativeImportPath":"js/packages/",
  "basePath":"/home/qcobjects/projects/mynewapp/",
  "projectPath":"/home/qcobjects/projects/mynewapp/",
  "domain":"mynewapp.qcobjects.com",
  "dataPath":"/etc/qcobjects/data/",
  "private-cert-pem":"/etc/letsencrypt/live/mynewapp.qcobjects.com/fullchain.pem",
  "private-key-pem":"/etc/letsencrypt/live/mynewapp.qcobjects.com/privkey.pem",
```

```
  "backend":{
    "routes":[
      {
        "path":"/createaccount",
        "microservice":"org.quickcorp.backend.signup",
        "responseHeaders":{}
      }
    ]
  }
}
```

Backend routing

Dentro de su archivo config.json puede establecer las rutas de back-end para sus microservicios. Para cada ruta de microservicio, se requiere una ruta y una cadena de paquete de microservicio.

```
{
  "backend":{
    "routes":[
      {
        "path":"/createaccount",
        "microservice":"org.quickcorp.backend.signup"
      }
    ]
  }
}
```

Cuando configuras las rutas de back-end, el servidor QCObjects HTTP2 Built-In sabra como manejar las llaadas de cada ruta definida. Para cada otra ruta que este indefinida mediante los ajustes del Back-end, el servidor se manejara para llamar al comportamiento por defecto que está usando un archivo estático como respuesta si existe. Con eso en mente puedes usar QCObjects tanto para administar y servir archivos estaticos o servicios servidos dinamicamente cuando lo necesites.

The QCObjects Microservice Class and Package

cuando configuras la definicion de ruta bakend, necesitas especificar un paquete de microdervicio.Este paquete de microservicio es una definicion de QCObjects de un paquete con una clase microservicio extendida desde la clase BackendMicroservice ya definida por QCObjects.

A continuación se muestra un ejemplo de una definición de paquete de microservicio, escrita en el archivo org.quickcorp.backend.signup.js

```
'use strict';
const fs = require('fs');

Package('cl.quickcorp.backend.signup',[
  Class('Microservice',BackendMicroservice,{
    body:{
      "jsonrpc": "2.0",
      "result": "",
      "id": 1
    },
    saveToFile: function (filename,data){
      logger.debug('Writing file: '+filename);
```

```
      fs.writeFile(filename, data, (err) => {
        if (err) throw err;
        console.log('The file has been saved!');
      });
    },
    post:function (data){
      let submittedDataPath = CONFIG.get('dataPath'); // this is filled out from
qcobjects-server
      let filename = submittedDataPath+'signup/signup'+Date.now().toString()+'.json';
      console.log('DATA RECEIVED: '+data);
      this.saveToFile(filename,data);
    }
  })
]);
```

El microservicio anterior está guardando un archivo con los datos recibidos de una solicitud posterior y respondiendo a una salida estándar jsonrpc 2.0. Lee sobre estas especificaciones JSON RPC 2.0 Aquí

El servidor QCObjects HTTP2 Built-In hara una llamada al metodo post() de la definicion de clase Microservice solo cuadose realiza una solicitud posterior en la ruta correcta definida en config.json que hace referencia al nombre del paquete como el punto de referencia de indexación inicial.

Para permitir que QCObjects entienda y ejecute sus microservicios de la manera correcta dentro de un paquete de microservicios, una definicion de clase Microservice es requerida y tambine la definicion de clase Microservice tiene que extender la clase BackendMicroservice que es parte de las clases built-in de QCObjects.

Generating a Self-Signed Certificate with QCObjects

```
> qcobjects-createcert
```

Working with a Letsencrypt HTTPS certificate, Certbot and QCObjects

Quick Start Guide

Quick Start your PWA (Progressive Web App)

```
> qcobjects create --pwa mynewapp
```

Quick Start your AMP (Accelerated Mobile Page)

```
> qcobjects create --amp mynewapp
```

Start Coding

Step 1: Start creating a main import file and name it like: cl.quickcorp.js. Put it into packages/js/ file directory

```
"use strict";
/*
* QuickCorp/QCObjects is licensed under the
* GNU Lesser General Public License v3.0
* [LICENSE] (https://github.com/QuickCorp/QCObjects/blob/master/LICENSE.txt)
*
* Permissions of this copyleft license are conditioned on making available
* complete source code of licensed works and modifications under the same
* license or the GNU GPLv3. Copyright and license notices must be preserved.
* Contributors provide an express grant of patent rights. However, a larger
* work using the licensed work through interfaces provided by the licensed
* work may be distributed under different terms and without source code for
* the larger work.
*
* Copyright (C) 2015 Jean Machuca,<correojean@gmail.com>
*
* Everyone is permitted to copy and distribute verbatim copies of this
* license document, but changing it is not allowed.
*/


Import ('external/libs');
Import ('cl.quickcorp.model');
Import ('cl.quickcorp.components');
Import ('cl.quickcorp.controller');
Import ('cl.quickcorp.view');

Package('cl.quickcorp',[
  Class('FormValidator',Object,{

  }),
]);
```

Step 2: Then create some services inhereting classes into the file js/packages/cl.quickcorp.services.js :

```
"use strict";
/*
* QuickCorp/QCObjects is licensed under the
* GNU Lesser General Public License v3.0
* [LICENSE] (https://github.com/QuickCorp/QCObjects/blob/master/LICENSE.txt)
*
* Permissions of this copyleft license are conditioned on making available
* complete source code of licensed works and modifications under the same
* license or the GNU GPLv3. Copyright and license notices must be preserved.
* Contributors provide an express grant of patent rights. However, a larger
* work using the licensed work through interfaces provided by the licensed
* work may be distributed under different terms and without source code for
* the larger work.
*
* Copyright (C) 2015 Jean Machuca,<correojean@gmail.com>
```

```
*
* Everyone is permitted to copy and distribute verbatim copies of this
* license document, but changing it is not allowed.
*/


Package('cl.quickcorp.service',[
    Class('FormSubmitService',JSONService,{
        name:'mySubmitService',
        external:true,
        cached:false,
        method:'POST',
        withCredentials:false,
        url:'https://api.github.com/orgs/QuickCorp/repos',
        _new_:function (){
          // service instantiated
          delete this.headers.charset; // do not send the charset header
        },
        done:function (result){
          _super_('JSONService','done').call(this,result);
        },
            fail: function(result) {
          //TODO negative case
          console.log("***** ERROR");
        }
  })
])
```

Step 3: Now it's time to create the components (cl.quickcorp.components.js)

```
"use strict";
/*
* QuickCorp/QCObjects is licensed under the
* GNU Lesser General Public License v3.0
* [LICENSE] (https://github.com/QuickCorp/QCObjects/blob/master/LICENSE.txt)
*
* Permissions of this copyleft license are conditioned on making available
* complete source code of licensed works and modifications under the same
* license or the GNU GPLv3. Copyright and license notices must be preserved.
* Contributors provide an express grant of patent rights. However, a larger
* work using the licensed work through interfaces provided by the licensed
* work may be distributed under different terms and without source code for
* the larger work.
*
* Copyright (C) 2015 Jean Machuca,<correojean@gmail.com>
*
* Everyone is permitted to copy and distribute verbatim copies of this
* license document, but changing it is not allowed.
*/
Package('cl.quickcorp.components',[
```

```
    Class('MyCustomComponent',Component,{
      name:'mycustomcomponent',
      cached:false,
      controller:null,
      view:null,
        templateURI:ComponentURI({
            'COMPONENTS_BASE_PATH':Component.basePath,
            'COMPONENT_NAME':'mycustomcomponent',
            'TPLEXTENSION':'tpl.html',
            'TPL_SOURCE':'default'
        })
    })
]);
```

Step 4: Once you have done the above components declaration, you will now want to code your controllers (cl.quickcorp.controller.js)

```
"use strict";
/*
* QuickCorp/QCObjects is licensed under the
* GNU Lesser General Public License v3.0
* [LICENSE] (https://github.com/QuickCorp/QCObjects/blob/master/LICENSE.txt)
*
* Permissions of this copyleft license are conditioned on making available
* complete source code of licensed works and modifications under the same
* license or the GNU GPLv3. Copyright and license notices must be preserved.
* Contributors provide an express grant of patent rights. However, a larger
* work using the licensed work through interfaces provided by the licensed
* work may be distributed under different terms and without source code for
* the larger work.
*
* Copyright (C) 2015 Jean Machuca,<correojean@gmail.com>
*
* Everyone is permitted to copy and distribute verbatim copies of this
* license document, but changing it is not allowed.
*/
"use strict";
Package('cl.quickcorp.controller',[
    Class('MainController',Controller,{
        _new_:function (){
            //TODO: Implement
            logger.debug('MainController Element Initialized');
        }
    }),
    Class('MyAccountController',Controller,{
        component: null,
        done:function (){
            var controller = this;
            logger.debug('MyAccountController Element Initialized');
            this.component.body.setAttribute('loaded',true);
        },
```

```
        _new_:function (o){
            //TODO: Implement
            this.component = o.component;
        }
    }),
]);
```

Step 5: To use into the HTML5 code you only need to do some settings between script tags

```
<script>
CONFIG.set('relativeImportPath','js/packages/');
CONFIG.set('componentsBasePath','templates/components/');
CONFIG.set('delayForReady',1); // delay to wait before executing the first ready event, it includes imports
CONFIG.set('preserveComponentBodyTag',false); // don't use ```<componentBody></componentBody>``` tag

Import('cl.quickcorp'); // this will import your main file: cl.quickcorp.js into js/packages/ file path
</script>
```

QCObjects CLI Tool

Usage

qcobjects [options] [command]

Options

-V, --version output the version number -h, --help output usage information

Commands

create [options] `<appname>` Crea una aplicación con `<appname>` **publish** [options] `<appname>` Publica una aplicación con `<appname>` **generate-sw** `<appname>` Genera el trabajador de servicio `<appname>` **launch** `<appname>` Inicia la aplicación.

Use:

$ qcobjects-cli [command] --help For detailed information of a command