Kiss Developer Manual

Kiss Developer Manual abc

Introduction

The *KISS* Framework} is an application development framework for developing web-based business applications, portable desktop applications, and command-line utilities. The main home for *Kiss* is https://github.com/blakemcbride/Kiss

Kiss' focus is on simplicity and development speed. By being simple to develop in, development and support of the application can occur more rapidly. Simplicity is achieved by abstracting away as much common functionality as possible so that developer lines of code are maximally applicable to the application solution rather than infrastructure and support of the framework. Throughout the framework, business-normal defaults have been employed in order to minimize commonly needed functionality.

Another goal of the *Kiss* framework is to be a complete web-based application development solution. *Kiss* isn't a browser solution alone, nor is it a back-end solution. *Kiss* includes solutions for both ends — although the two sides may largely be used independently.

Kiss attempts to create a consistent interface. This can greatly simply code even in simple cases. For example, in terms of an input text control, why would you disable/enable it with:

```
$('#id').prop('disable', false);
$('#id').prop('disable', true);
```

and then hide/show it with:

```
$('#id').hide();
$('#id').show();
```

Kiss provides a consistent interface. With *Kiss*, you would do:

```
$$('id').disable();
$$('id').enable();
$$('id').hide();
$$('id').show();
```

Kiss is designed to be simple to get started with, simple to learn, and simple to use. *Kiss* does this while supporting important technologies such as micro-services, front-end components, and SQL.

The term *single page application* has several, subtly different, meanings. One meaning is that the entire application code is bundled into a single file or HTTP GET request. In that sense, *Kiss* is not a

single page application. This makes no sense for a business application that could have hundreds of screens.

Another meaning of the term *single page application* is that there is only a single `html` tag and all of the remaining pages are modifications of the original `html` tag contents. In this sense, *Kiss* is a *single page application*. *Kiss* applications lazy-load as needed. Browser cache is leveraged to minimize Internet traffic.

Kiss is used in a production environment and built by someone with more than 30 years experience as a framework designer and a business application software engineer. So *Kiss* is not a proof of concept.

Kiss was built as a solution to the challenges faced by the author when developing web-based business applications. As such, *Kiss* is more a solution for business application development than for the development of public facing company presentation web sites.

Another goal of *Kiss* is to keep the front-end and back-end as independent of each other as possible. To this end, communications between the front-end and back-end occur via REST services and JSON. This accomplished two things. First, it allows your organization to be best prepared for the ever-evolving software environment. Pieces can be changed and enhanced without causing massive re-writes of the entire system. The second advantage is that by pushing as much processing to the front-end as possible (rendering the display on the front-end), the system can better scale.

Kiss Highlights

Some highlights of the *Kiss* system include:

Back-end Highlights

1. Micro services - add, change, or delete a web service on a running system.
2. Each web-method is in a single file and are very simple. No configuration files or setup code needed.
3. Easy access to common SQL databases with support for nested queries without cursor interference.
4. All REST services are stateless. However, the system fully authenticates each request.
5. Changes to web services occur immediately, on a running system, without the need to reboot the application.
6. A growing class library to handle common business application needs.
7. Back-end framework is written in Java, and the system is portable to Linux or Windows servers.
8. Web services may be written in Groovy, Java or Common Lisp. Python, JavaScript, Ruby, and Scala are expected to follow soon.
9. User authentication
10. Asynchronous back-end REST services (via a queue and thread pool) provide support for heavy loads and high throughput.

11. A powerful and convenient class library for dealing with SQL persistence that supports PostgreSQL, Microsoft SQL Server, MySQL, Oracle, and SQLite.

Front-end Highlights

1. Build your own HTML components thus encapsulating any amount of code into a simple, custom HTML tag.
2. Browser cache control. Never ask your users to clear their browser cache again.
3. All code written in JavaScript/HTML/CSS. No need for a complex build and debug process, nor any need to learn yet another language.
4. Growing list of included business oriented components designed to provide simple access to fully functional business components.
5. Straight forward means of designing your own components without a lot of hidden and unpredictable magic.
6. System is small and concise, rather then hundreds of megabytes other systems take up.
7. Consistent and simple API.

Back-end Web Service Example

The following example depicts a complete back-end web service. The path to the file is its URL. The class name is the web method name.

The file is a text file, but compiled code gets executed. Authentication occurs before `main` is called.

Simply drop the file in place and the web service and method become immediately available on a running system. Changes to the service take effect immediately (no need to reboot the server app). There are no configuration files or other code that needs to be changed.

For example, the following file is located in the `services` directory.

```
class MyWebService {
    void myWebMethod(JSONObject injson, JSONObject outjson) {
        int num1 = injson.getInt("num1");
        int num2 = injson.getInt("num2");
        outjson.put("result", num1 + num2);
    }
}
```

Front-end Web Service Usage Example

The following front-end example utilizes the web service defined in the previous sub-section.

```
let data = {
    num1:  22,
```

```
    num2:  11
};
let res = await Server.call("services/MyWebService", "myWebMethod", data);
if (res._Success) {
    let result = res.result;
    //...
}
```

Supported Environments

Development Environment

The following development platforms are supported by the *Kiss* framework:

- Linux
- Mac
- Windows
- WSL under Windows

Production Environment

The following production platforms are supported by the *Kiss* framework:

- Linux
- Windows Server

Databases Supported

The following database servers are supported by the *Kiss* framework:

- PostgreSQL
- Microsoft SQL Server
- MySQL
- Oracle
- SQLite

Java

The system is tested with Java verion 8 and 17. Any Java version above 8 is expected to work.

HTML component usage

To use a component add to HTML:

```
<my-component></my-component>
```

Add to JavaScript:

```
Utils.useComponent('MyComponent');
```

The component can put any HTML in the component location, have any functionality, have its own modal windows, and use other components. The component can have custom and non-custom attributes (like style). Non-custom attributes do what you's expect them to do.

The system also supports tag-less components. This provides an easy way to package arbitrary blocks of code (that can have screens too).

System Maturity And Stability

The Kiss system has been used in production environments for a several years. Additionally, several commercial applications utilize *Kiss*. In spite of this, however, *Kiss* is constantly being adjusted in response to additional needs, evolving environments, and bug fixes.

We use *Kiss* daily in a Linux and PostgreSQL environment. Therefore, it is best tested there. While we support all of the listed environments, they receive a bit less testing. If you encounter a problem, please reach out to us. It is probably easy for us to fix, and we are happy to do so.

Getting All Source Code

Source code for all of *Kiss* and its dependencies is freely available. The builder program located at `src/main/core/org/kissweb/builder/Tasks.java` contains the paths to all of the external dependencies (those not included in the *Kiss* distribution). The following lists the paths to the internal dependencies (those included with *Kiss*):

abcl.jar	https://common-lisp.net/project/armedbear
json.jar	https://github.com/blakemcbride/JSON-java
SimpleWebServer.jar (only used during development)	https://github.com/blakemcbride/SimpleWebServer

Support, Contact, And Links

The *Kiss* main web site is at https://kissweb.org

Source code is at https://github.com/blakemcbride/Kiss

Public discussion and support is available at https://groups.google.com/forum/#!forum/kissweb

Issue tracking is at https://github.com/blakemcbride/Kiss/issues

Commercial support is available. Contact us via email at kissweb.org@gmail.com

License

Acknowledgments

The Kiss design, code, documentation, and web site were written by Blake McBride. The author gratefully acknowledges and appreciates, among others, the following:

Apache Groovy located at https://groovy-lang.org

Dynamic Loader located at https://github.com/dvare/dynamic-loader

JSON-Java located at https://github.com/stleary/JSON-java (although I am using a modified version available at https://github.com/blakemcbride/JSON-java)

C3P0 located at https://www.mchange.com/projects/c3p0

ABCL project located at https://common-lisp.net/project/armedbear

Melaine Sarbey (melswildart@gmail.com) for creating the Kiss logo.