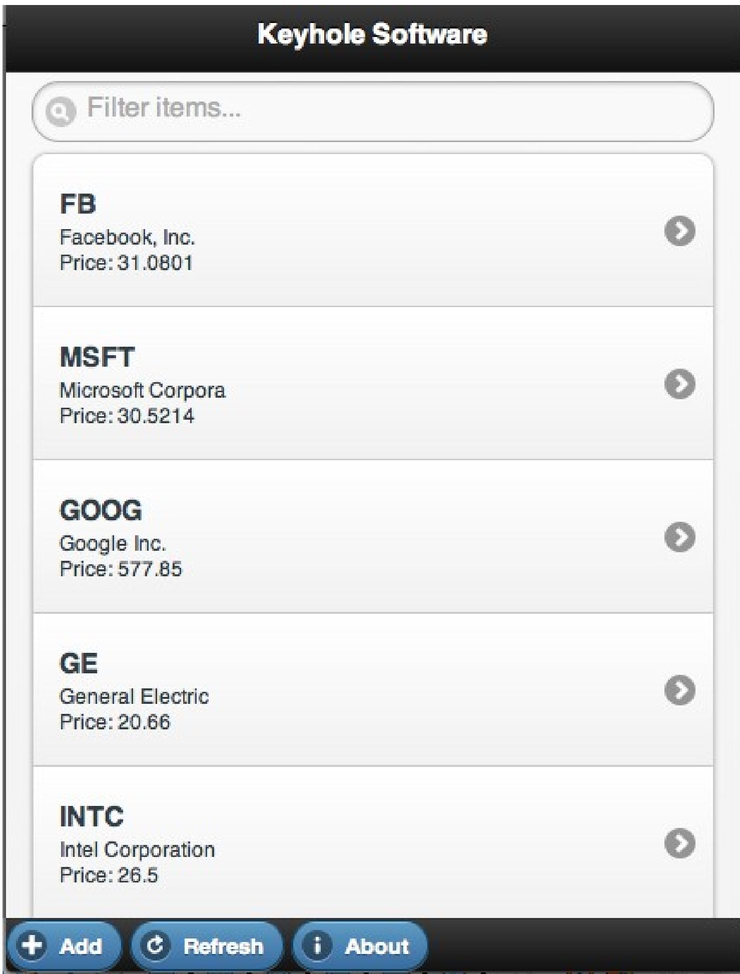
Mobile Application Architecture With HTML5 / Javascript

a Keyhole Software white paper

Introduction

This white paper walks through the frameworks and structures used to implement a rich client side Javascript/HTML5-based mobile application. The user interface and navigation elements are all browser resident components, while the application servers only role is to provide JSON data access for the user interface.

Since the intent is to provide a reference of framework and application structure, the example implements basic functionality. A screen shot of the mobile browser based application described in this paper is shown here:

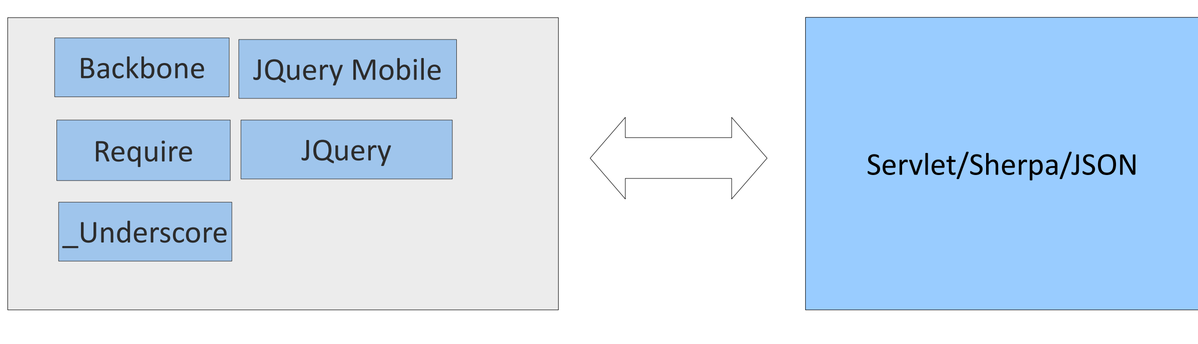


Keyhole Software, LLC. 8900 State Line Road, Suite 455 Leawood, KS 66206 Tel 877-521-7769 www.keyholesoftware.com
Copyright © 2012 Keyhole Software, LCC. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws.

HTML5 Application Architecture

JavaScript was never intended to be a general purpose application development language. Its original intent was to enhance the user experience in the browser by allowing dynamic HTML to be rendered and changed without having to access a remote server. This provides a perceived and real performance improvement. Mobile devices don't have the processing horsepower or the bandwidth access that desktop/laptop resident browsers do, so rich user interfaces will minimize round-trips to servers by implementing as much as possible in Javascript and HTML5 elements on the client.

This is a departure from current server side web applications (JSP/ASP/PHP) where dynamic HTML elements are rendered on the server. In this new topology, server side elements support authentication and data access requirements, and user interaction and most application logic will reside in the client browser. This can be seen in the graphic shown below:



For maintainability and stability, the following Javascript frameworks are used to help enforce a modular, layered object oriented Javascript application architecture.

Backbone.js – http://backbonejs.org

Backbone allows an object oriented way to separate and apply the model view controller (MVC) pattern to Javascript applications. HTML5 user interfaces are separated from controller and object model implementations. Additionally, a standard navigation mechanism between user interface features is provided.

Require.js – http://requirejs.org

Javascript file and module loader framework. This framework allows dependent java scripts to be loaded and validated when “required” by a java script module/function. This communicates dependency information to the developer and asserts an error if the Javascript module/library is not loaded.

_Underscore.js – http://underscorejs.org/

Underscore.js is a library that provides utility methods that allow functional programming mechanisms to be applied to collections of objects. It also provides a HTML template-ing feature.

Keyhole Software, LLC. 8900 State Line Road, Suite 455 Leawood, KS 66206 Tel 877-521-7769 www.keyholesoftware.com
Copyright © 2012 Keyhole Software, LCC. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws.

JQuery – http://jquery.com/

Library used to access and manipulate HTML DOM elements.

jQuery Mobile – http://jquerymobile.com/

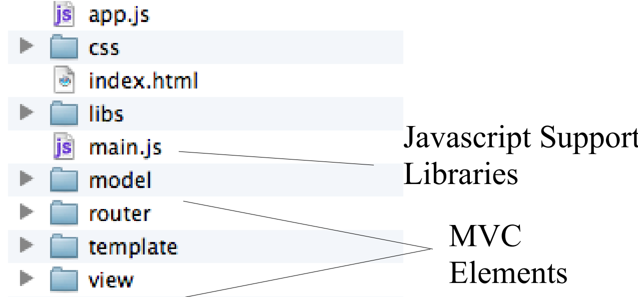
HTML5 User interface component library. Provides suite of UI controls rendered in HTML5. Provides mechanisms for event handling, as well as look and feel.

khsSherpa – https://github.com/in-the-keyhole/khs-sherpa/

Java application server framework that allows remote Java objects to be accessed via HTTP requests. It provides token-based authentication support along with automatic marshaling of java types to JSON objects. Optional: JSONP cross-domain support is enabled.

Javascript Folder Structure

Javascript does not provide a standard way to organize programming elements, which are just text based .js files. Other languages have organizing mechanisms, such as packages in Java or Namespaces in C#. Trying to define all Javascript functions and objects in one large file is cumbersome to maintain, especially when a large part of the application will be defined in Javascript. Therefore, file system folders under a root folder can be defined to help partition Javascript source into areas of responsibility.



Folders will contain MVC Javascript elements of the application. An example folder structure for the example application is shown below. Assume these folders are located at the document root of a web server/application server.

Modularity Support

Javascript does not have a built-in mechanism to partition source code elements. Other languages do have these mechanisms, such as Java with its packages, and C# with Namespaces. However, applications can import or include directives to bring in dependent modules. This allows applications and frameworks to modularize code improving maintainability and reuse.

The **require.js** framework provides a mechanism to efficiently break up and treat Javascript files as modules and provides the ability to define, import, and access dependent modules.

The Require framework uses the Asynchronous Module Definition (AMD) framework to accomplish defining and loading dependencies. The require/AMD function to load modules is shown here:

Copyright © 2012 Keyhole Software, LCC. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws.

```
define([modules], factory function);
```

Each module is an individual Javascript file that defines a Javascript object. When invoked, the modules are loaded and an object instance is created and passed into the factory function for the developer to use. The example below shows a Javascript module using the require **define()** function load utility dependency.

```
define(["util"], function (util) {
        return {
         date: function(){
                var date = new Date();
        return util.format(date);
           }
    };
});
```

The Require framework also has optimization features in order to help minimize file loading and increase performance.

Backbone MVC

This framework allows the popular MVC design pattern to be implemented using Javascript. Typical web applications implement this pattern with a general purpose language on the server side using dynamic HTML generation technology, such as JSP/ASP or some kind of HTML template engine. The framework provides components or abstractions for processing user input and applying application object models to the dynamic HTML mechanism. The Backbone framework provides a way to apply these mechanisms in Javascript, instead of generating HTML tags on the server.

HTML Template

A Backbone view is simply a static HTML file with HTML5 roles applied. Backbone provides a template mechanism, so that model attributes can be applied to the HTML when the view is rendered by a view/controller. The example application defines a JQuery Mobile list view which is defined using HTML5 role attributes. The **stock-list.html** HTML5 view is shown below:

```
<div id="stockListContainer" data-role='content'>
        <ul data-role="listview" id='tcStockList' data-inset="true" data-filter="true"></ul>
</div>
```

Stock Items in the list view are defined in **stock-list-item.html**, it applies the backbone template mechanism for stock model JSON object detail display. HTML with template elements are shown below.

```
<a href='#<%=ticker%>'>
        <h4><%= ticker %></h4>
        <p><%= name %></p>
        <p>Price: <%= price %></p>
</a>
```

Notice that the template expressions above are similar to JSP/ASP pages, where <%= %> are used to mark locations to be replaced with attribute values from a model JSON object. HTML templates are located in the template folder.

Keyhole Software, LLC. 8900 State Line Road, Suite 455 Leawood, KS 66206 Tel 877-521-7769 www.keyholesoftware.com
Copyright © 2012 Keyhole Software, LCC. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws.

View/Controller

An HTML view is rendered by navigating to a Backbone controller implementation. Controllers bind Javascript object(s) to an HTML view and tells the framework to render the view along with defining and handling events. In Backbone terminology, views are controllers, and creating a Backbone view is done by extending from the framework supplied **Backone.View** object.

The partial example for the **stockListPage.js** view controller below, where first loads required javascript .js files using the **require.js** framework. Which invokes the **define([modules,...], controller())** Javascript function that returns a Backbone view controller function. Notice how this function extends a **Backbone.View** object. The nice thing about the Require framework's define function is that it loads dependent modules required by the view controller implementation. Notice how a model and html template modules are also provided to the view/controller module object:

```
define([
        'jquery',
        'backbone',
        'underscore',
        'model/stockListCollection',
        'view/stockListView',
        'text!template/stock-list.html'],
        function($, Backbone, _, StockListCollection, StockListView, stockListTemplate) {var list = {};
                return Backbone.View.extend({
                        id : 'stock-list-page',
```

When a view/controller instance is created, the initialize: function is invoked and provides a way to define events and initialize the controller's model, which can be an individual object or collection of objects.

Notice in the example **stockListPage.js** view/controllers initialize function, a **StockListCollection** object is created. Collections are also a Backbone supplied object that provides a way to manage “collections” of Javascript object models for the view. When this controller is invoked, the initialize() method is executed. When an instance is created, it uses Jquery selectors to apply Backbone event handlers to buttons. The initialize function snippet is shown below:

```
var list = {};
return Backbone.View.extend({
        id : 'stock-list-page',
        initialize : function() {
                this.list = new StockListCollection();

                $("#about").on("click", function(e){
                        navigate(e);
                        e.preventDefault();
        e.stopPropagation();
                        return false;
                });


                $("#add").on("click", function(e){
                        navigate(e);
                        e.preventDefault();
        e.stopPropagation();
                        return false;
                });

        },
```

Events are associated with a view/controller method by an associating form event and a jQuery selector with a

Keyhole Software, LLC. 8900 State Line Road, Suite 455 Leawood, KS 66206 Tel 877-521-7769 www.keyholesoftware.com
Copyright © 2012 Keyhole Software, LCC. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws.

method name. The example below shows event and handling methods for the example apps and add buttons on the stock list page. Notice the navigation commands, as they will be discussed in the next section.

```
events: {
  "click #about" : "about",
  "click #add" : "add",
 },
 about : function(e) {

window.stock.routers.workspaceRouter.navigate("#about",true);
               return false;
 },

 add : function(e) {
                    window.stock.routers.workspaceRouter.navigate("#add",true);
               return false;
 },
```

A View/Controllers HTML template is rendered and displayed when the render() method is sent to an instance. The render function for the stockListPage.js is shown below. You can see how it compiles a template, then displays an HTML template which is applied to the controllers el attribute. The this.el attribute is the controllers location in the DOM when HTML will be inserted. Next notice how another view/controller is instantiated and rendered. The StockListView controller renders the JQueryMobile list view of of stocks.

```
        render : function(eventName) {
                var compiled_template = _.template(stockListTemplate);
                var $el = $(this.el);
                $el.html(compiled_template());
                this.listView = new StockListView({
                        el : $('ul', this.el),
                        collection : this.list
                });
                this.listView.render();
                return this;
                },
        });

});
```

Navigation

Navigating between controller views is another mechanism provided by Backbone. Backbone refers to this “routing” and as such, provides Backbone.Router object that can be extended to define navigation routes. The example application router is shown below:

```
define(['jquery', 'backbone', 'jquerymobile' ], function($, Backbone) {
        var transition = $.mobile.defaultPageTransition;
        var WorkspaceRouter = Backbone.Router.extend({
                // bookmarkMode : false,
                id : 'workspaceRouter',
                routes : {
                        "index" : "stockList",
                        "stockDetail" : "stockDetail"
                },
                initialize : function() {
                        $('.back').on('click', function(event) {
                                window.history.back();
                                return false;
```


Keyhole Software, LLC. 8900 State Line Road, Suite 455 Leawood, KS 66206 Tel 877-521-7769 www.keyholesoftware.com
Copyright © 2012 Keyhole Software, LCC. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws.

```
                        });
                        this.firstPage = true;
                },
                defaultRoute: function() {
                        console.log('default route');
                        this.runScript("script","stockList");
        },
        stockDetail: function() {
                require(['view/stockDetailView'], function (ThisView) {
                                var page = new ThisView();
                                $(page.el).attr({
                                        'data-role' : 'page',
                                        'data-add-back-btn' : "false"
                                });

                                page.render();

                                $(page.el).prependTo($('body'));

                                $.mobile.changePage($(page.el), {
                                        transition : 'slide'
                                });

                 });
        },

        stockList : function() {

                        require(['view/stockListPage'], function (ThisView) {
                                var page = new ThisView();
                                $(page.el).attr({
                                        'data-role' : 'page',
                                        'data-add-back-btn' : "false"
                                });

                                page.render();

                                $(page.el).prependTo($('body'));

                                $.mobile.changePage($(page.el), {
                                        transition : 'flip'
                                });

                 });
                },
        });
        return new WorkspaceRouter();
});
```

The require define function is used to provide instances of JQuery, Backbone, and JQuery Mobile instances to the overridden router function/methods. When the router instance is created, routes are initialized with an ID and function name to execute when the “route” is navigated to. In the example above, there are two routes: **#index** and **#stockDetail.** Notice the functions defined for these routes.

The router object can be invoked to navigate to a defined view/controller with the following expression:

```
<aRouter>.navigate("#index");
```

A routing function creates an instance of a **Backbone.View** and invokes the render function. The snippet below is from the example extended **BackBone.Router** function that renders the stock list jQuery Mobile list view. Notice in the source below how the Require framework creates an instance of the view/stockListPage Backbone view controller, then uses JQuery to adorn page attributes and render it.

Copyright © 2012 Keyhole Software, LCC. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws.

```
// Router navigation funtion
stockList : function() {
        require(['view/stockListPage'], function (ThisView) {
            var page = new ThisView();
            $(page.el).attr({
                'data-role' : 'page',
                'data-add-back-btn' : "false"
            });
            page.render();

            $(page.el).prependTo($('body'));

            $.mobile.changePage($(page.el), {
                transition : 'flip'
            });
    });
    },
```

Collection/Model

Backbone provides a collection object that manages lists of **Backbone.Model** objects. View/Controller objects have attributes that reference a list or single Javascript object. A **Backbone.Collection** object for the **StockListItem** model objects displayed by the view/controller is shown below:

```
define(['jquery', 'backbone', 'underscore', 'model/stockItemModel'],
function($, Backbone, _, StockListItem) {
    return Backbone.Collection.extend({
        model : StockListItem,
        url : 'http://localhost:8080/khs-sherpa-jquery/sherpa?endpoint=StockService&action=quotes&callback=?',
        initialize : function() {
            $.mobile.showPageLoadingMsg();
            console.log('findScripts url:' + this.url);
            var data = this.localGet();
            if (data == null) {
                this.loadStocks();
            } else {
                console.log('local data present..');
                this.reset(data);
            }
        },
        loadStocks : function() {
            var self = this;
            $.getJSON(this.url, {
                }).success(function(data, textStatus, xhr) {
                    console.log('script list get json success');
                    console.log(JSON.stringify(data.scripts));
                    self.reset(data);
                    self.localSave(data);
                }).error(function(data, textStatus, xhr) {
                    console.log('error');
                    console.log("data - " + JSON.stringify(data));
                    console.log("textStatus - " + textStatus);
                    console.log("xhr - " + JSON.stringify(xhr));
                }).complete(function() {
                    console.log('json request complete');
                    $.mobile.hidePageLoadingMsg();
                });
        },
        localSave : function(data) {
            var d = JSON.stringify(data);
            localStorage.setItem('STOCKS', d);
        },
        localGet : function() {
```


Keyhole Software, LLC. 8900 State Line Road, Suite 455 Leawood, KS 66206 Tel 877-521-7769 www.keyholesoftware.com
Copyright © 2012 Keyhole Software, LCC. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws.

```
                var d = localStorage.getItem('STOCKS');
            data = JSON.parse(d);
            return data;
    }

    });
});
```

When the collection object is initialized, (in the example application this happens when a view/controller creates an instance) the specified URL attribute is invoked using the jQuery AJAX mechanism to invoke a server side JSONP endpoint. The endpoint returns JSON stock objects, which are automatically mapped to the collections **stockListItem** model. The backbone.model definition for the StockItemModel is shown below:

```
define(['jquery',
        'backbone',
        'underscore'],
        function($, Backbone, _) {
            return Backbone.Model.extend({
                initialize : function() {

                }
            });
});
```

For readers familiar with strongly typed languages, Javascript's ability to turn JSON-formatted strings into Javascript model objects seems likes magic.

Backbone.Model objects have a number of methods to help save and to synchronize with a server. Likewise Backbone.Collection objects have methods for synchronizing and saving to a server as well as to perform functional programming type operations. You can check out these capabilities at http://backbonejs.org.

Local Storage

Other methods added to the example **StockListCollection** extended **Backbone.Collection** provide the ability to save and restore objects from the HTML5 local storage mechanism. Defined in the **localSave()** and **localGet()** methods in the above collection. Once a collection of Stock objects are obtained from the server, the HTML5 application can operate without connectivity. This example utilizes the key/value local session storage mechanism. HTML5 also provides a local relational storage mechanism referred to as webSql. However, work towards this spec has stalled, and it is not fully supported, so relying upon its existence could be risky. The key/value session storage is well supported. Check out this http://www.w3.org/TR/webdatabase/ for more information regarding the webSql spec.

Application Bootstrap/Startup

The standard **index.html** starts things off, when loaded style sheets are defined along with the following tag:

```
<script data-main=”main.js src=”libs/require/require.js”/>
```

This invokes the **main.js** function that configures and loads supporting the Javascript libraries. The Require framework provides a nice mechanism to key a Javascript library to simple name and a base URL location path. Since the lib folder is off of the doc root,no base Url path is required. An example is shown in the next code

Keyhole Software, LLC. 8900 State Line Road, Suite 455 Leawood, KS 66206 Tel 877-521-7769 www.keyholesoftware.com
Copyright © 2012 Keyhole Software, LCC. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws.

snippet:

```
require.config({
        paths : {
            'backbone' : 'libs/AMDbackbone-0.5.3',
            'underscore' : 'libs/underscore-1.2.2',
            'text' : 'libs/require/text',
            'jquery' : 'libs/jquery-1.7.2',
            'jquerymobile' : 'libs/jquery.mobile-1.1.0-rc.2'
        },
        baseUrl : ''
    });
```

This startup function also uses the require function to configure jQuery mobile properties and loads the **app.js** script with navigates to the **#index** route displaying the stock-list-item.html.

The app.js script is shown below, and it initializes the workspace router instance, starts backbone, then navigates to the **#index** page. Source is shown below:

```
define(['backbone', 'router/workspaceRouter'], function(Backbone, WorkspaceRouter) {

    "use strict";

     $(function(){
        window.tc = {
            routers : {
                workspaceRouter : WorkspaceRouter
            },
            views : {},
            models : {},
            ticker: null
        };

        var started = Backbone.history.start({pushState:false, root:'/HTML5BackboneJQMRequireJS/'});
        window.tc.routers.workspaceRouter.navigate("#index", {trigger:true});
    });
});
```

Server Side JSON Endpoints

An application server configured to use the khsSherpa JSON framework provides URLs to endpoints that provide create, read, update, and delete methods for Lists and individual Stock objects. The framework marshals HTTP request parameters to Java Endpoint method calls and serializes Java objects to and from JSON strings.

This example project is defined and intended to be deployed as a JEE WAR component. This WAR contains both static HTML/Java Script that is initially delivered and made resident on the clients browser and the Sherpa JSON java application server endpoints that drive the HTML5 interface.

Here's the definition of the Java server endpoint that serves up stock quote JSON objects in JSON format. Endpoints are invoked using a HTTP URL get.

The example application only requires an endpoint to retrieve a collection of Stock objects. However, more realistic applications would require authentication and more endpoints to support CRUD operations. This framework supports these requirements. For more feature descriptions, check the framework out on GITHUB at https://github.com/in-the-keyhole/khs-sherpa/ .

Keyhole Software, LLC. 8900 State Line Road, Suite 455 Leawood, KS 66206 Tel 877-521-7769 www.keyholesoftware.com
Copyright © 2012 Keyhole Software, LCC. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws.

Conclusion

Even though this example is simplistic in functionality, the goal was to introduce an MVC application architecture for browser-resident applications. Eliminating the need for application servers having to render dynamic HTML for application interfaces is a key feature of the application architecture presented in this paper. Javascript is not a natural general purpose programming language, however the explosion of mobile devices and the large adoption of HTML5 and the resistance and non-support of browser plug-in technologies, is making Javascript with HTML5 a viable way to deliver rich browser-based applications to mobile devices.

Complete source for the example application can be found on GitHub at:
https://github.com/in-the-keyhole/khs-backbone-example

About The Author

David Pitt is a Sr. Solutions Architect with nearly 25 years IT experience. For the last fifteen years, David has helped corporate IT departments adopt object technology. Since 1999, he has been leading and mentoring development teams with software development utilizing Java (JEE) and .NET (C#) based technologies. He is an author of numerous technical articles, and a co-author of a popular IBM WebSphere book that documents some of his architecture design patterns.

About Keyhole Software

Keyhole Software is a “boutique” software development and consulting firm with a remarkable team. Specializing in Java and .NET technologies, we are experts in application solutions and services such as Custom Development (full SDLC), Application Maintenance, Legacy Systems Integration and Technical Mentoring. Agile development is our strong suit, and we excel in helping companies adopt the best software practices and techniques for success.

Our HTML5 / Javascript Services

- Outsourced Development – A Keyhole team provided to perform analysis, design, construction, and testing of Mobile/Web-based applications
- Development Support – Specialized members of our team participate as project team member and perform development activities
- HTML5 / Javascript Education - 3-day custom course on mobile and web development
- Mentoring / Player Coaching - Consultant provided to help with best practices and knowledge transfer

For More Information

Keyhole Corporate Kansas City
8900 State Line Road, Suite 455
Leawood, KS 66206
Tel: (877) 521-7769

Keyhole St. Louis
Phone: (314) 329-1699

Keyhole Chicago
200 E Randolph St
Chicago, IL 60601
Phone: (630) 460-8317

Published: July 11, 2012

Revised: July 23, 2012

Copyright © 2012 Keyhole Software, LCC. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws.