WEBfactory®

WEBfactory 2010

Accessing web services using HTML 5 and JavaScript

WEBfactory Version 3.3.x

September 2012

This whitepaper is a product of the company WEBfactory GmbH.

WEBfactory GmbH
Hollergasse 15
74722 Buchen
Germany

Tel: (+49) 6281 5233 0
Fax: (+49) 6281 5233 333

http://www.webfactory-world.de

All information and descriptions were selected with special care, following careful investigations. However, application errors or changes leading to a series of changed commands or other results may not be entirely excluded. WEBfactory GmbH cannot undertake any legal or other liability for such changes or any errors deriving from them.

WEBfactory GmbH denies any liability for increased costs of phone calls as a consequence of using the respective communications modules, even in the case of accidental connections.

WEBfactory GmbH has drawn up this user manual according to the most advanced knowledge but it does not undetake any responsibility for the programs / systems generating the results expected by the user.

WEBfactory GmbH reserves the right to make changes to the user manual without undertaking the responsibility of informing third parties as to such changes.

We reserve the right to make technical changes!

1. Introduction

This whitepaper is intended for all the users of WEBfactory 2010 software version 3.3 and is designed to facilitate the understanding of WEBfactory 2010's approach on using HTML and JavaScript to access web services.

The document will present methods of accessing the alarming and logging web services using HTML and JavaScript.

For a correct understanding of the following information, WEBfactory 2010 software version 3.3 and additional modules must be installed on the operating machine.

For more information about installing WEBfactory 2010, system requirements, licensing and release notes, please visit the WEBfactory Knowledge Base at:

http://webfactory-support.de/assets/documentation/Default.htm .

2. Accessing web services using HTML and JavaScript

Accessing the web services using HTML and JavaScript is possible by using additional scripts. Follow this link to download the necessary scripts.

IMPORTANT: The jquery.ajaxdotnet.js script needs to be downloaded from our source. This version has an error handling bug fix which is not available in the original version of the script.

IMPORTANT: In order to work, the HTML file and the scripts need to be accessed using IIS and need to have the same protocol, host and port as the web services.

IMPORTANT: The data passed to the web services is case sensitive and must match exactly the function signature of the web services.

2.1. Accessing alarming web services using HTML and JavaScript (data binding)

In order to access the scripts, the head of the HTML file must contain the links to the external scripts:

```
<script src="Scripts/jquery-1.5.1.min.js"
type="text/javascript"/>
<script src="Scripts/json2.min.js" type="text/javascript"/>
<script src="Scripts/jquery.ajaxdotnet.3.js"
type="text/javascript"/>
<script src="Scripts/knockout-2.0.0.js"
type="text/javascript"/>
<script src="Scripts/helpers.js" type="text/javascript"/>
```

Besides the provided scripts, the HTML file must contain the actual code that will access the web service and retrieve the data. The web service call can be wrapped inside a function for easier usage:

```
function getAlarmGroups(arguments, successCallback,
errorCallback){

      $.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/AlarmsService.
svc/js/GetAlarmGroups",
                  {
                      data: arguments,
                      success: function (data) {
                                    successCallback(data.d);
                      },
                      error: function () {
                          errorCallback();
                      }
                  }
              );
      }
```

The above function calls the GetAlarmGroups method from the AlarmsService web service.

In order to display the results, a viewModel has to be defined. In this case, the viewModel will contain a single property – the alarmGroups array:

```
var viewModel = {
            alarmGroups : ko.observableArray(),
};
```

If more information needs to be pulled from the web service, it can be added to the viewModel. This implementation is based on the MVVM (Model-View-ViewModel) pattern which allows decoupling between the data (Model), the view (HTML) by using a view model that makes the data suitable for being displayed in the view using a declarative model.

More information on this can be found here:
http://knockoutjs.com/documentation/introduction.html

If the web service call is successful, the retrieved data will be pushed item by item in the viewModel .alarmGroups array. If an error occurs when calling the web service, an alert will be shown:

```
getAlarmGroups({
                  languageID: 7
            },function(result){
                  viewModel.alarmGroups.removeAll();
                  $.each(result, function(index,item){
                        viewModel.alarmGroups.push(item);
                  });
            },function(){
                  alert("Error");
            });
```

The head of the HTML page should look like this:

```
<head>
    <meta charset="utf-8" />
    <script src="Scripts/jquery-1.5.1.min.js"
type="text/javascript"/>
    <script src="Scripts/json2.min.js" type="text/javascript"/>
    <script src="Scripts/jquery.ajaxdotnet.3.js"
type="text/javascript"/>
    <script src="Scripts/knockout-2.0.0.js"
type="text/javascript"/>
    <script src="Scripts/helpers.js" type="text/javascript"/>
    <script type="text/javascript">
      function getAlarmGroups(arguments, successCallback,
errorCallback){
      $.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/AlarmsService.
svc/js/GetAlarmGroups",
                  {
                      data: arguments,
                      success: function (data) {
                                    successCallback(data.d);
                      },

                       error: function () {
                          errorCallback();
                      }
                  }
              );
      }
      $(document).ready(function(){

            var viewModel = {
                  alarmGroups : ko.observableArray(),
            };
            getAlarmGroups({
                        languageID: 7
                  },function(result){
                        viewModel.alarmGroups.removeAll();
                        $.each(result, function(index,item){
                              viewModel.alarmGroups.push(item);
                        });
                  },function(){
                        alert("Error");
                  });
            ko.applyBindings(viewModel);
      });
      </script>
</head>
```

To display the data in the HTML page, an HTML element from the will be bound to the configured view model (in our example below, the tbody element). In this example, an HTML table will host the data:

```
      <table>
            <thead>
                  <tr>
                        <th>ID</th>
                        <th>SymbolicTextName</th>
                        <th>SymbolicTextTranslation</th>
                  </tr>
            </thead>
            <tbody data-bind="foreach: alarmGroups">
                  <tr>
                        <td data-bind="text:ID"/>
                        <td data-bind="text:SymbolicTextName"/>
                        <td data-
bind="text:SymbolicTextTranslation"/>
                  <tr>
            </tbody>
      </table>
```

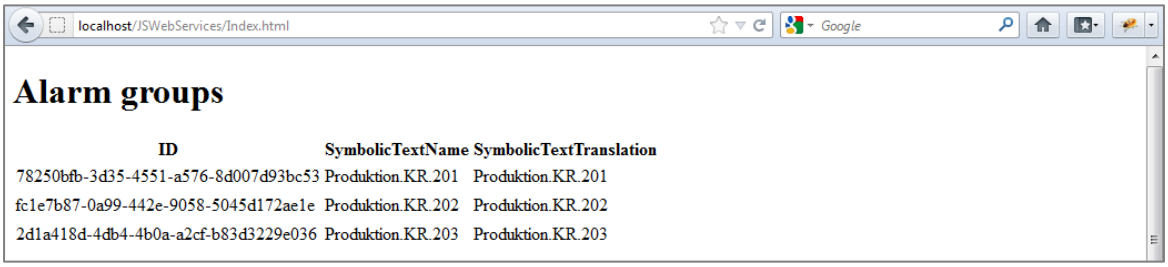
When the call to ko.applyBindings() is made, the data from the viewModel is connected to the defined views. This mechanism is known as “data binding”.

The above data-binding will create a row in the table with the corresponding data for each item in the alarmGroups array. The row will have three cells, displaying the corresponding properties of the alarm group item (ID, SymbolicTextName, SymbolicTextTranslation).

The body of the HTML page should look like this:

```
<body>
	<h1>Alarm groups</h1>
	<table>
		<thead>
			<tr>
				<th>ID</th>
				<th>SymbolicTextName</th>
				<th>SymbolicTextTranslation</th>
			</tr>
		</thead>
		<tbody data-bind="foreach: alarmGroups">
			<tr>
				<td data-bind="text:ID"/>
				<td data-bind="text:SymbolicTextName"/>
				<td data-
bind="text:SymbolicTextTranslation"/>
			<tr>
		</tbody>
	</table>
</body>
```

If the HTML file is run from the IIS and the WEBfactory server is online and has alarms, the browser will display the list of all defined alarm groups together with their translation in the specified language ID.



2.2. Accessing alarming web services using HTML and JavaScript

A more complex example is getting data for online alarms, using the filter items. The same procedure shall be followed.

```
function getOnlineAlarms(arguments, successCallback,
errorCallback){

      $.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/AlarmsService.
svc/js/GetOnlineAlarms",
                  {
                      data: arguments,
                      success: function (data) {
                                    successCallback(data.d);
                      },
                      error: function () {
                          errorCallback();
                      }
                  }
              );
      }
```

The viewModel will contain the onlineAlarms array definition:

```
var viewModel = {


              onlineAlarms: ko.observableArray(),
};
```

The getOnlineAlarms function defined above will require additional filters in order to get the data properly form the web service:

```
getOnlineAlarms({
      filter : {
            LanguageID : 7,
            AlarmGroups : [],
            AlarmTypes: [],
            MinimumPriority: 0,
            MaximumPriority: 1,
            SortOrder: 4,
            MaxRowCount: 10,
            AlarmStatusFilter : 0,
            StartTime : {
                  DateTime: new Date(1900,0,0).toMsJson(),
                  OffsetMinutes: 0
              },
            EndTime : {
                  DateTime: new Date().toMsJson(),
                  OffsetMinutes: 0
              },
            Column : 0,
            ColumnFilter : null,
            FilterAlarmGroupsByUser : false,
            UserName : null
          }

      },function(result){
            viewModel.onlineAlarms.removeAll();
            $.each(result, function(index,item){
                  viewModel.onlineAlarms.push(item);
            });
      },function(){
            alert("Error");
      });
```

IMPORTANT: because there is no standard DateTimeOffset format for JavaScript, whenever the web service requires data of this type the toMsJson() extension method must be used. The following example shows how to pass the current date and time as a DateTimeOffset value:

```
            DateTime: new Date().toMsJson()
```

Up to this point, the head of the HTML file should look like this:

```
<head>
    <meta charset="utf-8" />
    <script src="Scripts/jquery-1.5.1.min.js"
type="text/javascript"></script>
    <script src="Scripts/json2.min.js"
type="text/javascript"></script>
    <script src="Scripts/jquery.ajaxdotnet.3.js"
type="text/javascript"></script>
    <script src="Scripts/knockout-2.0.0.js"
type="text/javascript"></script>
    <script src="Scripts/helpers.js" type="text/javascript"></script>

      <script type="text/javascript">

      function getOnlineAlarms(arguments, successCallback,
errorCallback){

      $.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/AlarmsService.svc/js/
GetOnlineAlarms",
                  {
                      data: arguments,
                      success: function (data) {
                                    successCallback(data.d);
                      },
                      error: function () {
                          errorCallback();
                      }
                  }
            );
      }
      $(document).ready(function(){

            var viewModel = {
                  onlineAlarms: ko.observableArray()
            };


            getOnlineAlarms({
                        filter : {
                              LanguageID : 7,
                              AlarmGroups : [],
                              AlarmTypes: [],
                              MinimumPriority: 0,
                              MaximumPriority: 1,
                              SortOrder: 4,
                              MaxRowCount: 10,
                              AlarmStatusFilter : 0,
                              StartTime : {
                                    DateTime: new
Date(1900,0,0).toMsJson(),
                                    OffsetMinutes: 0
                              },
```

```
                              EndTime : {
                                    DateTime: new Date().toMsJson(),
                                    OffsetMinutes: 0
                              },
                              Column : 0,
                              ColumnFilter : null,
                              FilterAlarmGroupsByUser : false,
                              UserName : null
                        }

                        },function(result){
                              viewModel.onlineAlarms.removeAll();
                              $.each(result, function(index,item){
                                    viewModel.onlineAlarms.push(item);
                        });


                        },function(){
                        alert("Error");
                        });
            ko.applyBindings(viewModel);
      });

      </script>
</head>
```

Next, the data-binding needs to be made accordingly to the data that needs to be retrieved.

The bindings will be made in a HTML table as well:

```
      <table>
            <thead>
                  <tr>
                        <th>AlarmLogID</th>
                        <th>DateOn</th>
                        <th>DateOff</th>
                        <th>DateAck</th>
                        <th>SysTime</th>
                        <th>AlarmID</th>
                        <th>AlarmTag</th>
                        <th>SignalName</th>
                        <th>SignalAliasName</th>
                        <th>Priority</th>
                        <th>AckText</th>
                        <th>AlarmLinkURL</th>
                        <th>AlarmSymbolicText</th>
                        <th>AlarmSymbolicTextTranslation</th>
                        <th>AlarmGroupSymbolicText</th>
                        <th>AlarmGroupSymbolicTextTranslation</th>
                        <th>AlarmTypeSymbolicText</th>
```

```
                        <th>AlarmTypeSymbolicTextTranslation</th>
                        <th>AckUserName</th>
                        <th>AlarmComment</th>
                        <th>OccurrenceComment</th>
                        <th>Status</th>
                        <th>ExtendedProperty1</th>
                        <th>HelpCause</th>
                        <th>HelpEffect</th>
                        <th>HelpRepair</th>
                        <th>OccurrenceCount</th>
                        <th>NavigationSource</th>
                        <th>NavigationTarget</th>
                        <th>OPCQuality</th>
                    </tr>
            </thead>
            <tbody data-bind="foreach: onlineAlarms" >
            <tr>
                        <td data-bind="text:AlarmLogID"/>
                        <td data-bind="text:DateOn"/>
                        <td data-bind="text:DateOff"/>
                        <td data-bind="text:DateAck"/>
                        <td data-bind="text:SysTime"/>
                        <td data-bind="text:AlarmID"/>
                        <td data-bind="text:AlarmTag"/>
                        <td data-bind="text:SignalName"/>
                        <td data-bind="text:SignalAliasName"/>
                        <td data-bind="text:Priority"/>
                        <td data-bind="text:AckText"/>
                        <td data-bind="text:AlarmLinkURL"/>
                        <td data-bind="text:AlarmSymbolicText"/>
                        <td data-bind=
"text:AlarmSymbolicTextTranslation"/>
                        <td data-bind=
"text:AlarmGroupSymbolicText"/>
                        <td data-bind=
"text:AlarmGroupSymbolicTextTranslation"/>
                        <td data-bind="text:AlarmTypeSymbolicText"/>
                        <td data-bind=
"text:AlarmTypeSymbolicTextTranslation"/>
                        <td data-bind="text:AckUserName"/>
                        <td data-bind="text:AlarmComment"/>
                        <td data-bind="text:OccurrenceComment"/>
                        <td data-bind="text:Status"/>
                        <td data-bind="text:ExtendedProperty1"/>
                        <td data-bind="text:HelpCause"/>
                        <td data-bind="text:HelpEffect"/>
                        <td data-bind="text:HelpRepair"/>
                        <td data-bind="text:OccurrenceCount"/>
                        <td data-bind="text:NavigationSource"/>
                        <td data-bind="text:NavigationTarget"/>
                        <td data-bind="text:OPCQuality"/>
                    <tr>
            </tbody>
      </table>
```

2.3. Accessing the logging web services using HTML and JavaScript

In order to access the scripts, the head of the HTML file must contain the links to the external scripts:

```
<script src="Scripts/jquery-1.5.1.min.js" type="text/javascript"/>
<script src="Scripts/json2.min.js" type="text/javascript"/>
<script src="Scripts/jquery.ajaxdotnet.3.js" type="text/javascript"/>
<script src="Scripts/knockout-2.0.0.js" type="text/javascript"/>
<script src="Scripts/helpers.js" type="text/javascript"/>
```

Besides the provided scripts, the HTML file must contain the actual code that will access the web service and retrieve the data. The web service call can be wrapped inside a function for easier usage:

```
function GetLogValues(arguments, successCallback, errorCallback){
$.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/SignalsService.svc/js/GetLo
gValues",
                {
                    data: arguments,
                    success: function (data) {
                        successCallback(data.d);
                    },
                    error: function () {
                        errorCallback();
                    }
                }
            );
    }
```

The above function calls the GetLogValues method from the SignalService web service.

The GetLogValues function defined above will require additional filters in order to get the data properly from the web service:

```
GetLogValues({
            filter:{
                        LogIDs: ["299F590C-A740-4B7C-A762-802E3F208FA5"],
                        StartDate : {
                            DateTime: new Date(1900,0,0).toMsJson(),
                            OffsetMinutes: 0
                        },
                        EndDate : {
                            DateTime: new Date().toMsJson(),
                            OffsetMinutes: 0
                        },
                        MaxResults: 5,
                        SortOrder: 4
            }
        }
```

The values are written in the HTML table using JavaScript:

```
function(result){

                for(var i = 0; i<result.length;++i){
                    var item = result[i];
                    var cells = "";
                    cells += "<td>"+item.EntriesDate+"</td>";
                    cells += "<td>"+item.Values[0].EditedValue+"</td>";
                    cells += "<td>"+item.Values[0].EditedValue2+"</td>";
                    cells += "<td>"+item.Values[0].Value+"</td>";
                    cells += "<td>"+item.Values[0].Value2+"</td>";
                    $(".logValues tbody").append("<tr>"+cells+"</tr>");
                };

            }
```

If the GetLogValues method fails to retrieve the correct data, the following function executed:

```
function(){
                alert("Error");
            }
```

The Head of the HTML file should look like this:

```
<head>
    <meta charset="utf-8" />
    <script src="Scripts/jquery-1.5.1.min.js"
type="text/javascript"></script>
    <script src="Scripts/json2.min.js" type="text/javascript"></script>
    <script src="Scripts/jquery.ajaxdotnet.3.js"
type="text/javascript"></script>
    <script src="Scripts/knockout-2.0.0.js"
type="text/javascript"></script>
    <script src="Scripts/helpers.js" type="text/javascript"></script>

    <script type="text/javascript">
    function GetLogValues(arguments, successCallback, errorCallback){

$.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/SignalsService.svc/js/GetLogVal
ues",
                {
                    data: arguments,
                    success: function (data) {
                        successCallback(data.d);
                    },
                    error: function () {
                        errorCallback();
                    }
                }
            );
    }

    $(document).ready(function(){

        GetLogValues({
            filter:{
                        LogIDs: ["299F590C-A740-4B7C-A762-802E3F208FA5"],
                        StartDate : {
                            DateTime: new Date(1900,0,0).toMsJson(),
                            OffsetMinutes: 0
                        },
                        EndDate : {
                            DateTime: new Date().toMsJson(),
                            OffsetMinutes: 0
                        },
                        MaxResults: 5,
                        SortOrder: 4
            }
        },function(result){
                for(var i = 0; i<result.length;++i){
                    var item = result[i];
                    var cells = "";
                    cells += "<td>"+item.EntriesDate+"</td>";
                    cells += "<td>"+item.Values[0].EditedValue+"</td>";
                    cells += "<td>"+item.Values[0].EditedValue2+"</td>";
```

```
                    cells += "<td>"+item.Values[0].Value+"</td>";
                    cells += "<td>"+item.Values[0].Value2+"</td>";
                    $(".logValues tbody").append("<tr>"+cells+"</tr>");
                };

            },
            function(){
                alert("Error");
            });
    });
        /*ko.applyBindings(viewModel);*/

</script>
</head>
```

The body of the HTML file will contain the table:

```
<body>
    <h1>Data Table</h1>
    <table class="logValues">
        <thead>
            <tr>
                <th>EntriesDate</th>
                <th>EditedValue</th>
                <th>EditedValue2</th>
                <th>Value</th>
                <th>Value2</th>
            </tr>
        </thead>
        <tbody >
            <tr>
                <td />
                <td />
                <td />
                <td />
                <td />
            <tr>
        </tbody>
    </table>
</body>
```

3. Signal-related Javascript functions

3.1. wfsConnect

Description

The `wfsConnect` method connects to the signal service and establishes a WEBfactory session. The `onSuccess` callback returns a session ID that will be further used in other requests.

Function

```
$.wfsConnect = function (onSuccess, onError, state) {
}
```

Parameters

`onSuccess` – The success callback. The callback will be executed when the operation is successful. The callback should look like: `function onSuccess(session, state)`.

- `session` - will contain information about the WEBfactory session and license:
  - `session.sessionId` - the session ID (used in all subsequent requests the Signals service).
  - `session.isLicensed` - `true`, if the license is valid; `false`, otherwise - is the argument passed to the method and can contain any user defined data.
- `state` - is the argument passed to the method and can contain any user defined data.

`onError` - the error callback. This callback will be executed when the operation fails. The callback must look like: `function onError(status, error, state)`.

- `status` - will contain the error message.
- `error` - contains more details about the error.
- `state` - is the argument passed to the method and can contain any user defined data.

Implementation

```
$.wfsConnect = function (onSuccess, onError, state) {

$.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/SignalsService.svc/js/Connect", {
```

```
            success: function (data) {
                onSuccess(data.d, state);
            },
            error: function (jqXhr, status, error) {
                onError(status, error, state);
            }
        });
    };
```

3.2. wfsDisconnect

Description

Disconnects from the WEBfactory, ending the current session. The session ID is used for this request.

Function

```
$.wfsDisconnect = function (sessionId, onSuccess, onError, state) {
}
```

Parameters

`sessionId` - the current session ID, obtained from a call to connect.

`onSuccess` - the success callback. This callback will be executed when the operation is successful. The callback must look like: `function onSuccess(state)`.

- `state` - is the argument passed to the method and can contain any user defined data.

`onError` - the error callback. This callback will be executed when the operation. The callback must look like: function `onError(status, error, state)`.

- `status` - will contain the error message.
- `error` - contains more details about the error.
- `state` - is the argument passed to the method and can contain any user defined data.

`state` - an user-defined object (can be null) which is passed to the callbacks.

Implementation

```
$.wfsDisconnect = function (sessionId, onSuccess, onError, state) {

$.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/SignalsService.svc/js/Disc
onnect", {
            data: {
                sessionId: sessionId
            },
            success: function () {
                onSuccess(state);
            },
            error: function (jqXhr, status, error) {
                onError(status, error, state);
            }
        });
    };
```

3.3. wfsReadSignals

Description

Reads the current value of the specified signal names.

Function

```
$.wfsReadSignals = function (sessionId, signalNames, onSuccess,
onError, state) {
}
```

Parameters

`sessionId` - the current session ID, obtained from a call to connect.

`signalNames` - an array of signal names for which the values will be read.

`onSuccess` - the success callback. This callback will be executed when the operation is successful. The callback must look like: `function onSuccess(results, state)`.

- `results` - an array containing the values of the requested signals. The items in the array have the same order as the supplied signal names (e.g. results[1] is the result for signalNames[1]). Each result item contains two fields:
  - `result.Value` - the signal value.
  - `result.Result` - the status of the signal read (0 = no error occurred).
- `state` - is the argument passed to the method and can contain any user defined data.

`onError` - the error callback. This callback will be executed when the operation fails. The callback must look like: `function onError(status, error, state)`.

- `status` - will contain the error message.
- `error` - contains more details about the error.
- `state` - is the argument passed to the method and can contain any user defined data.

`state` - an user-defined object (can be null) which is passed to the callbacks.

Implementation

```
$.wfsReadSignals = function (sessionId, signalNames, onSuccess,
onError, state) {

$.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/SignalsService.svc/js/Read
Signals", {
            data: {
                sessionId: sessionId,
                signalNames: signalNames
            },
            success: function (data) {
                onSuccess(data.d, state);
            },
            error: function (jqXhr, status, error) {
                onError(status, error, state);
            }
        });
    };
```

3.4. wfsRegisterSignals

Description

Registers a list of signals for getting updates on them.

Function

```
$.wfsRegisterSignals = function (sessionId, signalNames, onSuccess,
onError, state) {
}
```

Parameters

`sessionId` - the current session ID, obtained from a call to connect.

`signalNames` - an array of signal names which will be registered.

`onSuccess` - the success callback. This callback will be executed when the operation is successful. The callback must look like: `function onSuccess(results, state)`.

- `results` - an array containing the error codes of for the signal registrations. The items in the array have the same order as the supplied signal names (e.g. results[1] is the result for signalNames[1]). A result of 0 means that the registration was successful.
- `state` - is the argument passed to the method and can contain any user defined data.

`onError` - the error callback. This callback will be executed when the operation fails. The callback must look like: function onError(status, error, state).

- `status` - will contain the error message.
- `error` - contains more details about the error.
- `state` - is the argument passed to the method and can contain any user defined data.

`state` - an user-defined object (can be null) which is passed to the callbacks.

Implementation

```
$.wfsRegisterSignals = function (sessionId, signalNames, onSuccess,
onError, state) {

$.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/SignalsService.svc/js/Regi
sterSignals", {
            data: {
                sessionId: sessionId,
                signalNames: signalNames
            },
            success: function (data) {
                onSuccess(data.d, state);
            },
            error: function (jqXhr, status, error) {
                onError(status, error, state);
            }
        });
    };
```

3.5. wfsUnregisterSignals

Description

Unregisters a list of signals from getting updates on them.

Function

```
$.wfsUnregisterSignals = function (sessionId, signalNames, onSuccess,
onError, state) {
}
```

Parameters

`sessionId` - the current session ID, obtained from a call to connect.

`signalNames` - an array of signal names which will be unregistered.

`onSuccess` - the success callback. This callback will be executed when the operation is successful. The callback must look like: `function onSuccess(results, state)`.

- `results` - an array containing the error codes of for the signal unregistrations. The items in the array have the same order as the supplied signal names (e.g. results[1] is the result for signalNames[1]). A result of 0 means that the unregistration was successful.
- `state` - is the argument passed to the method and can contain any user defined data.

`onError` - the error callback. This callback will be executed when the operation fails. The callback must look like: function onError(status, error, state).

- `status` - will contain the error message.
- `error` - contains more details about the error.
- `state` - is the argument passed to the method and can contain any user defined data.

`state` - an user-defined object (can be null) which is passed to the callbacks.

Implementation

```
$.wfsUnregisterSignals = function (sessionId, signalNames, onSuccess,
onError, state) {
```

```
$.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/SignalsService.svc/js/Unregi
sterSignals", {
            data: {
                sessionId: sessionId,
                signalNames: signalNames
            },
            success: function (data) {
                onSuccess(data.d, state);
            },
            error: function (jqXhr, status, error) {
                onError(status, error, state);
            }
        });
    };
```

3.6. wfsGetUpdates

Description

Requests updates of any previously registered signals. Initial value of `requestId` should be one, the next values being obtained from the `wfsGetNextRequestId` when function returning the `prevRequestId`. The `ResponseId` from the `onSuccess` result should be fed in the `prevResponseId` parameter of the `wfsGetNextRequestId` function, in order to obtain the subsequent `RequestId` parameters.

Function

```
$.wfsGetUpdates = function (sessionId, requestId, onSuccess, onError,
state) {
}
```

Parameters

`sessionId` - the current session ID, obtained from a call to connect.

`requestId` - initial value should be 1, subsequent values should be obtained using the `getNextRequestId` function.

`onSuccess` - the success callback. This callback will be executed when the operation is successful. The callback must look like: `function onSuccess(results, state)`.

- `result` - an object containing the signal updates:

- `result.ResponseId` - the server response id (will be used in the call to getNextRequestId).
- `result.Updates` - an array containing the updated signals. Each update has the following fields:
  - `update.key` - the signal name.
  - `update.value` - the signal value.

- `state` - is the argument passed to the method and can contain any user data.

`onError` - the error callback. This callback will be executed when the operation fails. The callback must look like: function onError(status, error, state)

- `status` - will contain the error message.
- `error` - contains more details about the error.
- `state` - is the argument passed to the method and can contain any user defined data.

`state` - an user-defined object (can be null) which is passed to the callbacks.

Implementation

```
$.wfsGetUpdates = function (sessionId, requestId, onSuccess, onError, state) {

$.ajaxDotNet("/_WEBFACTORY/WebServices/WCF/SignalsService.svc/js/GetUpdates", {
            data: {
                sessionId: sessionId,
                requestId: requestId
            },
            success: function (data) {
                onSuccess(data.d, state);
            },
            error: function (jqXhr, status, error) {
                onError(status, error, state);
            }
        });
    };
```

3.7. wfsGetNextRequestId

Description

Gets the next request ID for performing a call to `getUpdates`.

Function

```
$.wfsGetNextRequestId = function (prevRequestId, prevResponseId) {
}
```

Parameters

`prevRequestId` - the previous request ID

`prevResponseId` - the previous response ID, as a result to `getUpdates`

Implementation

```
$.wfsGetNextRequestId = function (prevRequestId, prevResponseId) {
        if (prevResponseId == 0) return 1;
        if (prevResponseId == prevRequestId) return prevRequestId %
1000 + 1;
        return 0;
    };
```