

# TribalFX - JavaScript Framework

## Allgemein

Bei FX handelt es sich um ein Framework, das dem Benutzer nicht eine reichhaltige Anzahl an Widgets bieten soll, sondern die Funktionalität und Effektivität des JavaScripts erweitern soll. D.h. es soll weitestgehend die objektorientierte Entwicklung unterstützt werden aber auch die funktionale Erweiterbarkeit des DOM im Fokus stehen.

Die Schwerpunkte bei der Entwicklung des Frameworks lagen auf folgenden Punkten:

- strikte Trennung von HTML, JavaScript und CSS
- Erweiterung der Funktionsbibliotheken von HTML-Elementen (global / lokal)
- Erweiterung der Funktionsbibliotheken von Objekten (global / lokal)
- AJAX-Schnittstelle mit Berechnung einer prozentualen Fortschrittsanzeige
- Bearbeitung von CSS-Eigenschaften direkt in der Stylesheet-Datei
- Adaption von Funktionen zwischen IE und Firefox

Folgender Programmierstil kommt zum Einsatz:

```
< ClassName > = ( function() {  
    function < Constructor >() {  
        this.< public MethodName > = < private MethodName >  
        ...  
    }  
  
    function < private MethodName >() { ... }  
    ...  
  
    return {  
        < ConstructorName > : < Constructor >  
    }  
} ) ();  
  
var < PackageName > = new < ClassName >.< ConstructorName >();  
< PackageName >.< public MethodName >();
```

**Konstanten**

---

**::ROOT****Beschreibung:** Pfadangabe zum Home-Verzeichnis

---

**::ROOT\_JS****Beschreibung:** Pfadangabe zum individuellen JavaScript-Verzeichnis des Benutzers ( jsFX )

---

**::ROOT\_FX****Beschreibung:** Pfadangabe zum Tribal-Framework-Verzeichnis ( TribalFX )

---

**::ROOT\_FX\_MODULES****Beschreibung:** Pfadangabe zum Modul-Verzeichnis des Tribal-Frameworks ( TribalFX/modules )

---

**::DOMAIN****Beschreibung:** Domain-Adresse

---

## Shortcuts

::\$

---

**Syntax:** \$( < Element-ID > )  
**Return:** < HTMLElement > Element  
**Beispiel:** var foo = \$( 'fooElement' );

**Beschreibung:** Kurzschreibweise für den Befehl document.getElementById().

::\$A

---

**Syntax:** \$A( < Collection > )  
**Return:** < Array > Array  
**Beispiel:** var foo = \$A( collection );

**Beschreibung:** Castet eine Collection ( Array-Object z.B. eine Node-List, HTML-Collection ) zu einem Array

---

## Tribal

### ::include

---

**Syntax:** Tribal.include( < FilePackage > )

**Return:** null

**Beispiel:** Tribal.include( 'scripts/foo' );

**Beschreibung:** Mit Hilfe des "include" kann man im Laufzeitbetrieb JavaScript-Dateien synchron nachladen. Somit wird verhindert, dass ein Funktionsaufruf, aufgrund noch nicht verfügbarer Funktionen, eine Exception wirft.

---

## Tribal.extend

### ::extend.DOMObject

---

**Syntax:** Tribal.extend.DOMObject ( < Object/PackageName >, < Interface > )

**Return:** < Object > Erweitertes Objekt

**Beispiel:** Tribal.extend.DOMObject( Tribal, new Foo.init() );

**Beschreibung:** Über diese Funktion lassen sich (DOM-)Objekte um weitere Methoden erweitern.

Zu den verfügbaren Objekten gehören unter anderem:

- Object
- Array
- Number
- String
- Date
- Math
- Function
- RegExp
- Boolean

**Achtung!!!:** In Verbindung mit "prototype" stehen die Methoden bei einigen Objekt-Typen erst nach einer neuen Instanzierung des Objektes zur Verfügung.

---

### ::extend.DOMinterface

---

**Syntax:** Tribal.extend.DOMinterface( < Interface > )

**Return:** null

**Beispiel:** Tribal.extend.DOMinterface( Tribal, new Foo.init() );

**Beschreibung:** Über diese Funktion lässt sich das DOM-Interface "HTMLInputElement" um weitere Methoden erweitern. Die Methoden stehen, auch bereits erzeugten Elementen nach der Initialisierung zur Verfügung.

---

## Tribal.browser

### ::browser

---

**Syntax:** Tribal.browser

**Return:** < String > Browsername

**Beispiel:** alert( Tribal.browser );

**Beschreibung:** Rückgabe des Browsernamens, mit dem das Dokument geöffnet wurde.

### ::browser.version

---

**Syntax:** Tribal.browser.version

**Return:** < String > Browserversion

**Beispiel:** alert( Tribal.browser.version );

**Beschreibung:** Rückgabe der Browserversion des Browsers, mit dem das Dokument geöffnet wurde.

### ::browser.OS

---

**Syntax:** Tribal.browser.OS

**Return:** < String > Betriebssystem

**Beispiel:** alert( Tribal.browser.OS );

**Beschreibung:** Rückgabe des Betriebssystems auf dem der Browser läuft.

---

## Tribal.ajax

### ::ajax.sendNewRequest

<b>Syntax:</b>	<pre>Tribal.ajax.sendNewRequest (     &lt; String &gt; transmission,     &lt; String &gt; inputTarget,     &lt; String    Array &gt; attachement,     &lt; String &gt; type,     &lt; String &gt; outputTarget,     [ &lt; String &gt; structData, ]     [ &lt; Boolean &gt; newGroup ] )</pre>														
<b>Parameter:</b>	<table><tr><td>transmission</td><td>Übertragungsmethode ( POST    GET )</td></tr><tr><td>inputTarget</td><td>Dokument, auf das der Request erfolgen soll</td></tr><tr><td>attachement</td><td>Anhang der dem Request mitgeschickt werden soll</td></tr><tr><td>type</td><td>Responsesdatentyp ( XSL    DATA ) XSL: Die Responsesdaten werden mit Hilfe des Parameter "structData" zu einem Dokument transformiert und an die Zielfunktion weitergeleitet DATA: Die Daten werden direkt zur Zielfunktion durchgereicht</td></tr><tr><td>outputTarget</td><td>Zielfunktion, an die die Responsesdaten weitergeleitet werden sollen</td></tr><tr><td>structData</td><td>Angabe einer gültigen XSL-Datei</td></tr><tr><td>newGroup</td><td>Angabe ob eine neue Requestgruppe erstellt werden soll</td></tr></table>	transmission	Übertragungsmethode ( POST    GET )	inputTarget	Dokument, auf das der Request erfolgen soll	attachement	Anhang der dem Request mitgeschickt werden soll	type	Responsesdatentyp ( XSL    DATA ) XSL: Die Responsesdaten werden mit Hilfe des Parameter "structData" zu einem Dokument transformiert und an die Zielfunktion weitergeleitet DATA: Die Daten werden direkt zur Zielfunktion durchgereicht	outputTarget	Zielfunktion, an die die Responsesdaten weitergeleitet werden sollen	structData	Angabe einer gültigen XSL-Datei	newGroup	Angabe ob eine neue Requestgruppe erstellt werden soll
transmission	Übertragungsmethode ( POST    GET )														
inputTarget	Dokument, auf das der Request erfolgen soll														
attachement	Anhang der dem Request mitgeschickt werden soll														
type	Responsesdatentyp ( XSL    DATA ) XSL: Die Responsesdaten werden mit Hilfe des Parameter "structData" zu einem Dokument transformiert und an die Zielfunktion weitergeleitet DATA: Die Daten werden direkt zur Zielfunktion durchgereicht														
outputTarget	Zielfunktion, an die die Responsesdaten weitergeleitet werden sollen														
structData	Angabe einer gültigen XSL-Datei														
newGroup	Angabe ob eine neue Requestgruppe erstellt werden soll														
<b>Return:</b>	null														
<b>Beispiel:</b>	<pre>Tribal.ajax.sendNewRequest (     'POST',     'foo.xml',     'text=helloWorld!',     'XSL',     'fooResponse',     'foo.xsl',     true ); Tribal.ajax.sendNewRequest (     'GET',     'foo.php',     'text=helloWorld!',     'DATA',     'fooResponse' );</pre>														
<b>Beschreibung:</b>	Erstellung eines asynchronen Requests. Die angeforderten Daten werden nach Eingang transformiert oder roh an die entsprechende Zielfunktion weitergeleitet. Zudem lassen sich mehrere Requests zu Gruppen zusammenfügen, die zur Berechnung der Fortschrittsanzeige dienen.														

---

## ::ajax.enableGrouping

---

**Syntax:** Tribal.ajax.enableGrouping()

**Return:** null

**Beispiel:** -

**Beschreibung:** Gruppierungen von Requests aktivieren.

---

## ::ajax.disableGrouping

---

**Syntax:** Tribal.ajax.disableGrouping()

**Return:** null

**Beispiel:** -

**Beschreibung:** Gruppierungen von Requests deaktivieren.

---

## ::ajax.nextGroup

---

**Syntax:** Tribal.ajax.nextGroup()

**Return:** null

**Beispiel:** -

**Beschreibung:** Neue Gruppe für kommende Requests erstellen.

---

## ::Handler::ajax.notInitialized

---

**Syntax:** Tribal.ajax.notInitialized()

**Return:** null

**Beispiel:** Tribal.ajax.notInitialized = function() {  
    alert( 'not initialized' );  
};

**Beschreibung:** Ajax-Handler, falls keine Verbindung zum Server aufgebaut werden kann.

---

## ::Handler::ajax.openSuccessful

---

**Syntax:** Tribal.ajax.openSuccessful()

**Return:** null

**Beispiel:** Tribal.ajax.openSuccessful = function() {  
    alert( 'open successful' );  
};

**Beschreibung:** Ajax-Handler, wenn eine Verbindung zum Server erstellt werden kann.

---

## ::Handler::ajax.sentRequest

---

**Syntax:** Tribal.ajax.sentRequest()

**Return:** null

**Beispiel:** Tribal.ajax.sentRequest = function() {  
    alert( 'sent Request' );  
};

**Beschreibung:** Ajax-Handler, wenn eine Anfrage an den Server gestartet werden konnte.

---

## ::Handler::ajax.headerReceived

---

**Syntax:** Tribal.ajax.headerReceived()

**Return:** null

**Beispiel:** Tribal.ajax.headerReceived = function() {  
    alert( 'header received' );  
};

**Beschreibung:** Ajax-Handler, wenn der Header eines Response-Datenpaketes empfangen wurde.

---

## ::Handler::ajax.progress

---

**Syntax:** Tribal.ajax.headerReceived()

**Return:** null

**Beispiel:** Tribal.ajax.progress = function() {  
    alert( 'progress' );  
};

**Beschreibung:** Ajax-Handler, wenn ein Response-Datenpaket aus einer Request-Gruppierung empfangen wurde.

---

## ::Handler::ajax.complete

---

**Syntax:** Tribal.ajax.headerReceived()

**Return:** null

**Beispiel:** Tribal.ajax.complete = function() {  
    alert( 'complete' );  
};

**Beschreibung:** Ajax-Handler, wenn alle Response-Datenpakete aus einer Request-Gruppierung empfangen wurden.

---

## Tribal.css

---

### ::css.wantCSSRule

---

**Syntax:** Tribal.css.wantCSSRule( < String > RegelName )

**Return:** < Object > CSS-Rule-Object

**Beispiel:** var fooRule = Tribal.css.wantCSSRule( '#foo' );

**Beschreibung:** Anfordern eines CSS-Rule-Objektes zur Speicherung von Style-Angaben. Falls die CSS-Regel noch nicht existiert wird diese neu angelegt.

---

### ::css.getCSSRule

---

**Syntax:** Tribal.css.getCSSRule( < String > RegelName )

**Return:** < Object > CSS-Rule-Object

**Beispiel:** var fooRule = Tribal.css.getCSSRule( '#foo' );

**Beschreibung:** Anfordern eines CSS-Rule-Objektes zur Speicherung von Style-Angaben. Das CSS-Rule-Objekt muss bereits existieren.

---

### ::css.addCSSRule

---

**Syntax:** Tribal.css.addCSSRule( < String > RegelName )

**Return:** < Object > CSS-Rule-Object

**Beispiel:** var fooRule = Tribal.css.addCSSRule( '#foo' );

**Beschreibung:** CSS-Rule-Objekt für eine neue Regel wird erstellt und zur Befüllung zurückgegeben.

---

### ::css.killCSSRule

---

**Syntax:** Tribal.css.killCSSRule( < String > RegelName )

**Return:** < Object > CSS-Rule-Object

**Beispiel:** var fooRule = Tribal.css.killCSSRule( '#foo' );

**Beschreibung:** Löschen eines CSS-Rule-Objektes

---

## JSON

### ::toJSON

---

**Syntax:** < Object | Array | String | Collection >.toJSON()

**Return:** < String > serialisiertes Objekt

**Beispiel:** alert( \$( 'eltern' ).toJSON() );

**Beschreibung:** Wandelt ein Objekt in einen String um.

### ::evalJSON

---

**Syntax:** < String >.toJSON()

**Return:** < Objekt || Array || String >

**Beispiel:** var foo = '{  
    1 : "hello", 2 : "world", 3 : "!"  
}'.evalJSON();

**Beschreibung:** Wandelt einen String in ein Objekt/Array/String um.

---

## Element

### ::addElement

---

**Syntax:** < Element >.addElement( < String > Type )  
**Return:** < Element > erstelltes Element  
**Beispiel:** var kind = \$( 'eltern' ).addElement( 'span' );  
kind.id = 'kind';

**Beschreibung:** Dem Elternelement ein Kindelement anhängen

### ::getElementsByClassName

---

**Syntax:** < Element >.getElementsByClassName( < String > TagName, < String > ClassName )  
**Parameter:** TagName == < String > | nur Elemente mit dem angegebenen Tag  
TagName == "\*" | alle Elemente durchsuchen  
**Return:** < Array > Elemente  
**Beispiel:** var kinder = \$( 'eltern' ).getElementsByClassName( '\*', 'kinder' );  
for( var kind in kinder )  
alert( kinder[ kind ].tagName );

**Beschreibung:** Durchsucht unter einem Element alle Kindelemente nach der angegebenen Class. Per "\*" als TagName werden alle Kinder durchsucht. Bei einer anderen Angabe werden nur die Kindelemente mit dem entsprechenden TagName durchsucht.

### ::cssKey

---

**Syntax:** < Element >.cssKey()  
**Return:** < String > absolute CSS-Pfandangabe  
**Beispiel:** var cssPath = \$( 'kind' ).cssKey();

**Beschreibung:** Ausgabe des absoluten CSS-Pfades zum jeweiligen HTML-Element.

## ::getOpacity

---

**Syntax:** < Element >.getOpacity()

**Return:** < Number > Transparenz-Wert

**Beispiel:** alert( \$( 'kind' ).getOpacity() );

**Beschreibung:** Ausgabe des Transparenz-Werte des jeweiligen Objektes.

## ::setOpacity

---

**Syntax:** < Element >.setOpacity( < Number > )

**Return:** null

**Beispiel:** \$( 'kind' ).setOpacity( 50 );

**Beschreibung:** Zuweisung eines Transparenz-Werte dem jeweiligen Objekt.

---

## Function

### ::bind

---

<b>Syntax:</b>	<code>&lt; Function &gt;.bind(     &lt; Object &gt; Object [, &lt; Argument &gt; Argument, ... ] )</code>
<b>Parameter:</b>	Object   Object, das per "this" in der aufzurufenden Funktion verfügbar sein soll
<b>Return:</b>	< Object > dynamisch erzeugtes Funktionsobjekt, das die eigentliche Funktion umhüllt
<b>Beispiel:</b>	<code>this.fooFunction.bind( this, 'hello', 'world' );</code>
<b>Beschreibung:</b>	Ruft eine Funktion im Kontext des Objektes auf und findet Verwendung bei Timeouts und Intervallen.

### ::bindAsEventListener

---

<b>Syntax:</b>	<code>&lt; Function &gt;.bindAsEventListener(     &lt; Object &gt; Object [, &lt; Argument &gt; Argument, ... ] )</code>
<b>Parameter:</b>	Object   Object, das per "this" in der aufzurufenden Funktion verfügbar sein soll
<b>Return:</b>	< Object > dynamisch erzeugtes Funktionsobjekt, das die eigentliche Funktion umhüllt
<b>Beispiel:</b>	<code>this.fooFunction.bindAsEventListener( this, 'hello', 'world' );</code>
<b>Beschreibung:</b>	Ruft eine Funktion im Kontext des Objektes auf und findet Verwendung bei Event-Handlern.

---

## Math

### ::sq

---

**Syntax:** Math.sq( < Number > Zahl )

**Return:** < Number > Zahl im Quadrat

**Beispiel:** alert( Math.sq( 2.1 ) );

**Beschreibung:** Berechnung einer Zahl im Quadrat.

### ::pythagoras

---

**Syntax:** Math.pythagoras (   
 < Number > Kathete1, < Number > Kathete2   
 )

**Return:** < Number > Kathete3

**Beispiel:** alert( Math.pythagoras( 2, 2 ) );

**Beschreibung:** Berechnung der 3. Kathete eines rechtwinkligen Dreiecks.

### ::randInt

---

**Syntax:** Math.randInt( < Integer > Max )

**Return:** < Integer > Integer-Zufallszahl

**Beispiel:** alert( Math.randInt( 10 ) );

**Beschreibung:** Berechnung einer Zufallszahl im Bereich von 0 bis Max.

---

## **IE-DOM vs. Firefox-DOM**

Bei der Entwicklung von CrossBrowser-Anwendungen entstehen die meisten Probleme in der Anpassung von eigenen Funktionen an die unterschiedlichen Browsertypen. Es kommen Weichen zum Einsatz, die den Quellcode nur sinnlos aufblähen, aber auch die Übersichtlichkeit beeinträchtigen. Aufgrund dessen wurden einige Grundfunktionen des IE in den Firefox adaptiert:

```
< Object >.attachEvent()  
< Object >.detachEvent()  
< Event >.keyCode  
< Event >.returnValue  
< Event >.offsetX  
< Event >.offsetY  
< Event >.clientX  
< Event >.clientY  
< Event >.shiftKey  
< Event >.ctrlKey  
< Event >.altKey  
< Element >.outerHTML  
< Element >.innerText  
< Element >.outerText  
< Element >.currentStyle  
< Element >.style.getAttribute()  
< Element >.style.setAttribute()  
< Element >.style.removeAttribute()
```