

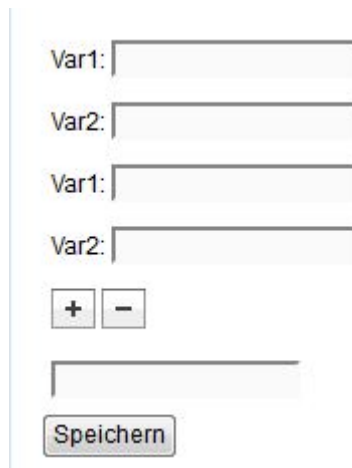
Extended DomRepeater

In order to to repeat elements in a web or smartform dynamically one needs to use the so called DomRepeater. Should the basic version be insufficient, e.g. because it can only delete the last row or one simply needs a more precise tool, the extended DomRepeater is an excellent choice: It consists of one javascript file and can be used in Smart- and Webforms.

Attention: jQuery is needed!!

Sourcen

TODO



Var1:

Var2:

Var1:

Var2:

Example Form

| | | | |
|------------|----------------------|----------------------------------|----------------------------------|
| Kriterium: | <input type="text"/> | <input type="button" value="+"/> | <input type="button" value="-"/> |
|------------|----------------------|----------------------------------|----------------------------------|

| | | | |
|------------|--|----------------------------------|----------------------------------|
| Kriterium: | <input type="text" value="Arbeitszeiten"/> | <input type="button" value="+"/> | <input type="button" value="-"/> |
| Kriterium: | <input type="text" value="Bezahlung"/> | <input type="button" value="+"/> | <input type="button" value="-"/> |

| | | | |
|------------|--|----------------------------------|----------------------------------|
| Kriterium: | <input type="text" value="Arbeitszeiten"/> | <input type="button" value="+"/> | <input type="button" value="-"/> |
| Kriterium: | <input type="text"/> | <input type="button" value="+"/> | <input type="button" value="-"/> |
| Kriterium: | <input type="text" value="Bezahlung"/> | <input type="button" value="+"/> | <input type="button" value="-"/> |

Example HTML

```
<div class="repeat row" data-repeat="kriterium">
  <div class="col-md-2 form-group">
    <label for="kriterium">
      Kriterium:
    </label>
  </div>
  <div class="col-md-4 form-group">
    <input id="kriterium" name="kriterium" class="form-control"/>
  </div>
  <div class="col-md-1 form-group">
    <button type="button" value="Einfügen" id="repeatKriterium"
name="repeatKriterium" data-callback="gadget.functions.addCallbackkriterium"
start="1" class="kriterium_add btn btn-default btn-lg">
      <span class="glyphicon glyphicon-plus" aria-
hidden="true"></span>
    </button>
    <button type="button" value="Entfernen"
name="repeatKriteriumRemover" id="repeatKriteriumRemover" data-
callback="gadget.functions.removeCallbackkriterium" class="kriterium_rem btn
btn-default btn-lg">
      <span class="glyphicon glyphicon-minus" aria-
hidden="true"></span>
    </button>
  </div>
</div>
```

The following points need special attention:

- The repeated element needs the class "repeat"
- The repeated element needs the attribute "data_repeat". The value must be unique
- The "+"-button has to have a callback-method which is named as follows: "gadget.functions.addCallbackCHANGE1". CHANGE1 must be switched to the value of the "data-repeat"-attribute of the repeated element
- The "+"-button has to have a class which is named as follows: "CHANGE1_add". CHANGE1 must be switched to the value of the "data-repeat"-attribute of the repeated element
- The "-"-button has to have a callback-method which is named as follows: "gadget.functions.removeCallbackCHANGE1". CHANGE1 must be switched to the value of the "data-repeat"-attribute of the repeated element
- The "-"-button has to have a class which is named as follows: "CHANGE1_rem". CHANGE1 must be switched to the value of the "data-repeat"-attribute of the repeated element
- If an option in a SELECT has the default="default" attribute, this option will be selected for cloned elements as long as no value is found. This is also important for mandatory fields during the start of an instance!

The "+"-button can receive a "start"-attribute. The element will be repeated n times as soon as the Smartform is opened.

Example Custom

```
var head = document.getElementsByTagName("head")[0];
var customUrl = "/loom-portal/custom/BELIEBIGERORDNER/";
scriptTag = document.createElement('script');
scriptTag.setAttribute("type", "text/javascript");
scriptTag.setAttribute("src", customUrl + "ExtendedDomRepeater.js");
head.appendChild(scriptTag);

gadget.functions.initFunction=function(){
    var callbackfunction = function(){
        // Wird ausgeführt nachdem der DomRepeater initialisiert ist
    }
    DOMRepeater(this.form, this.entity, callbackfunction);
}
```

Important details to consider:

- The first argument is the form
- The second argument is the entity
- The third argument is optional and contains a callback-method which will be triggered after all rows are created

DomRepeater Code

domRepeater.js

```
/**
 * @author fta on 03.03.16.
 */

(function() {
    "use strict";

    // jQuery
    var $ = jQuery;

    // Tim entity
    var ENTITY = null;

    // Repeater count template
    var COUNT = '<input type="hidden" name="ID" value="1">';

    /**
     * Creates a Object that stores a clean template of the repeatable node
     * Parses that node, applies add/rem functionality and attaches it to
the DOM
     * Stores references to its parent/child Repeater objects
     */
}
```

```
* @param $html {Node}
* @param parent {Repeater}
* @param idx {Number}
* @constructor
*/
function Repeater($html, parent, idx) {

    // Get the repeater id
    // Clone the target html twice
    // $html gets attached to the DOM
    // Clone gets passed on to the next sibling
    // and store references to the parent Repeater
    // and all child Repeater objects
    this.id      = $html.data('repeat');
    this.$html   = $html.clone(true, true);
    this.clone   = $html.clone(true, true);

    this.parent  = parent;
    this.children = [];

    this.init(idx);

}

/**
 * Adds add/rem functionality to the $html
 * and puts this Repeater into the children array of the parent
 * Looks for nested repeaters in $html
 *
 * @param idx {Number}
 */
Repeater.prototype.init = function(idx) {

    // Put this Repeater in its parent.children array
    this.parent.children.splice(idx, 0, this);

    // Run all init functions
    this.initSubRepeatables();
    this.initCountField();
    this.initClicks();
    this.initInputs();

};

/**
 * Finds all nested repeatables on the same level
 * Deep search is done by recursion
 */
Repeater.prototype.initSubRepeatables = function() {

    // Searches for repeatables (on the same level) in a given context
```

```
    $('.repeat:not(* .repeat .repeat)', this.$html).each(function(i, el)
{
    // Replace the original node with the cloned and parsed
Repeater.$html
    el.parentNode.replaceChild(new Repeater($(el), this,
0).$html.get(0), el);

    }.bind(this));

});

/**
 * Creates a hidden input field representing the count
 * of this Repeater branch. If a Model is provided it
 * will be updated
 */
Repeater.prototype.initCountField = function() {

    var self = this;

    // Do this only once
    if (!this.parent.$count) {

        // Create a hidden field and store it on the parent Repeater
        // This field will hold the current Repeater count
        this.parent.$count = $(COUNT.replace('ID',
this.id)).on('count.rep', function() {

            // Update count on the view
            this.value = self.parent.children.length;

            if (ENTITY) {

                // Update count on the model
                ENTITY.setValue(this.name, this.value);

            }

        });

        // Attach the hidden field to the parent node
        // so it wont be removed when this Repeater instance is removed
        $(this.$html.context()).parent().append(this.parent.$count);

    }

    // Trigger a model update
    this.parent.$count.trigger('count.rep');

};
```

```
/**
 * Searches for add/rem buttons and binds onclick function to them
 */
Repeater.prototype.initClicks = function() {

    var self = this;

    // Search for the add button and attach its click function
    var cleanId = this.id.replace("[", "[").replace("]", "]");
    this.$html.on('click', '.' + cleanId + '_add', function() {

        self.add(true);

        // Look for a callback
        var fn = self.addCallback
            || (self.addCallback = eval($(this).data('callback')));

        if (typeof(fn) === 'function') {

            // If provided invoke the callback
            fn.call(self.parent.children[self.index() + 1]);

        }

    });

    // Search for the rem button and attach its click function
    this.$html.on('click', '.' + cleanId + '_rem', function() {

        self.rem();

        // Look for a callback
        var fn = self.remCallback
            || (self.remCallback = eval($(this).data('callback')));

        if (typeof(fn) === 'function') {

            // If provided invoke the callback
            fn.call(self);

        }

    });

};

/**
 * Searches for all input fields of this Repeater
 * and binds all necessary events (alter/save/change/remove) to them
 */
Repeater.prototype.initInputs = function() {
```

```
var self = this;

// Find and cache all inputs of this Repeater
this.$inputs = $(':input[name],span[id],div[id]',
this.$html).not($('repeat :input', this.$html)).each(function(i, el) {

    // Store the original id/name
    var $el = $(el), id = el.id, name = el.name;

    $(el).on('alter', function() {

        // Get the current suffix
        var suffix = self.idSuffix();

        // Adjust the id/name attributes
        this.id    = id ? id + suffix : null;
        this.name = name ? name + suffix : null;

    }).on('change.rep', function() {

        if (ENTITY) {

            // Don't use true/false if this is a checkbox, use this
            fields value or ''
            ENTITY.setValue(this.name, $el.is(':checkbox') ?
this.checked ? this.value : '' : this.value);

        }

    }).on('save.rep', function() {

        if (ENTITY) {

            if ($el.is(':radio')) {

                if (this.checked) {

                    // Don't persist unchecked radio buttons
                    ENTITY.setValue(this.name, this.value);

                }

            }
            else {

                // Don't use true/false if this is a checkbox, use
                this fields value or ''
                ENTITY.setValue(this.name, $el.is(':checkbox') ?
this.checked ? this.value : '' : this.value);

            }

        }

    });

});
```

```
    }

    }).on('model2view.rep', function() {

        if (ENTITY) {

            if($el.prop("tagName") == "SPAN")
                $el.text([ENTITY[this.id] || '']);
            else if($el.prop("tagName") == "SELECT"){

                var val = ENTITY[this.name];
                if(!val){

$el.find("option[default]").attr("selected",true);

                }else{

                    $el.val([ENTITY[this.name] || '']);

                }

            }else if($el.prop("tagName")!= "DIV"){
                // Puts the model value on the view
                $el.val([ENTITY[this.name] || '']);
            }

        }

    }).on('destroy.rep', function() {

        if (ENTITY) {

            // Clears the model
            ENTITY.setValue(this.name, '');

        }

    });

});

/**
 * Adds a new instances directly after this Repeater
 *
 * @param click {?Boolean} Source of call is a click event
 */
Repeater.prototype.add = function(click) {

    // Create a new Repeater object and attach the resulting html to the
```


DOM

```
this.$html.after(new Repeater(this.clone, this.parent, this.index()
+ 1).$html);

    if (click) {

        // Recently added instance index
        var nextIndex = this.index() + 1;

        // Alter and resave all following instances
        this.broadcast(nextIndex, 'alter save', true);

        // Clean the model of the next instance
        this.parent.children[nextIndex].broadcast('this', 'destroy');

        if (ENTITY) {

            // Resize smartform
            gadget.onResize();

        }

    }

};

/**
 * Removes this instance from them DOM and Repeater tree
 */
Repeater.prototype.rem = function() {

    // There has to be at least 1 element
    if (this.parent.children.length > 1) {

        // Last known index
        var idx = this.index();

        // Get the last instance of this branch
        var lastInstance = this.parent.children.slice(-1)[0];

        if (this !== lastInstance) {

            // Broadcast a clear event on the last instance
            lastInstance.broadcast(lastInstance, 'destroy');

        }

        // Splice this Repeater from its parent.children array
        // and detach the html from the DOM
        this.parent.children.splice(idx, 1)[0].$html.remove();
    }
}
```

```

        // Trigger a model update
        this.parent.$count.trigger('count');

        // Removes this branch from the model
        this.broadcast('this', 'destroy');

        // Broadcast a alter event
        this.broadcast(idx, 'alter save');

    }

};

/**
 * Moves down this branch starting at a specified index
 * and triggers a event or calls a function on all instances
 *
 * @param index {Number}
 * @param event {Function|String}
 * @param reverse {?Boolean}
 */
Repeater.prototype.broadcast = function(index, event, reverse) {

    // Needed to reverse the array out of place
    var siblings = !isNaN(index) ? this.parent.children.slice(index) :
[this];

    (!reverse ? siblings : siblings.reverse()).forEach(function(rep) {

        // Needed to reverse the array out of place
        var children = $.extend([], rep.children);

        // Move deeper into the branch
        (!reverse ? children :
children.reverse()).forEach(function(child) {

            child.broadcast(0, event, reverse);

        });

        if (typeof(event) === 'string') {

            // If event is of type string
            // split it into partials and trigger them
            event.split(' ').forEach(function(e) {

                rep.$inputs.trigger(e);

            });

        }

    })
};

```

```
        else {

            // Callback
            event.call(rep);

        }

    });

};

/**
 * Moves up the Repeater tree and builds a string representation of
 * Repeater indexes eg. [0][1][2]
 *
 * @returns {String}
 */
Repeater.prototype.idSuffix = function() {

    // Store index strings
    // and the current Repeater
    var ids = [], curr = this;

    // Keep moving up until there is no parent
    while (curr.parent) {

        // Put the next id at the beginning of the array
        ids.unshift('[' + curr.index() + ']');

        // Get the parent
        curr = curr.parent;

    }

    return ids.join('');
};

/**
 * Returns the index of the current Repeater
 * in its parent.children array
 *
 * @returns {Number}
 */
Repeater.prototype.index = function() {

    // Return the index of the current Repeater
    return this.parent.children.indexOf(this);
};

/**
 * Looks for a positive count on the Tim entity
```

```
* and adds that many instances of the current Repeater
* When everything is added this function looks for child Repeater
* and recursively initializes them aswell
*
*/
function initialize(rep) {

    if (!rep) {
        return;
    }

    if (ENTITY) {

        // Get the provided count or -1
        var count =
        parseInt(ENTITY.getValue(rep.parent.$count.attr('name'))) ||
        rep.$html.find("." + rep.parent.$count.attr('name')+"_add").attr("start") ||
        0) - 1;

        // Add instances according to the count
        for (var i = 0; i <count; i++) {

            rep.add(false);

        }

    }

    // Recursively initialize all siblings/children
    rep.parent.children.forEach(function(child) {

        // Alter all inputs and update the view
        child.$inputs.trigger('alter').trigger('model2view');

        // At this point there is only one child
        initialize(child.children[0]);

    });

}

/**
 * Main entry point
 * Call this function to attach
 * domrepeater functionality to your document
 *
 * @param ctx {Document|jQuery}
 * @param entity {?Entity}
 * @constructor
 */
window.DOMRepeater = function(ctx, entity, callback) {
```

```
ENTITY = entity;

// Holds all top level Repeater instances
var root = window.Repeater = [];

// Start traversing the context
// use a plain object as the tree root
// Lookup only top level repeatables
$('.repeat:not(*.repeat.repeat)', ctx).each(function(i, el) {

    // Create a new top level parent instance
    // and eplace the original node with the cloned and parsed
    Repeater.$html
        el.parentNode.replaceChild(new Repeater$(el), root[root.push({
            children: []
        }) - 1], 0).$html.get(0), el);

});

// Loop through all top level objects
// and recursively initialize them
root.forEach(function(obj) {

    initialize(obj.children[0]);

});

if (ENTITY) {

    // Resize smartform
    gadget.onResize();

}

if(typeof callback == "function")
    callback();

};

})();
```

From:

<https://wiki.tim-solutions.de/> - **TIM Wiki** / [NEW TIM 6 Documentation](#)

Permanent link:

https://wiki.tim-solutions.de/doku.php?id=en:extended_domrepeater

Last update: **2021/07/01 09:52**

