DOCUMENTS

GADGETS
Einrichtung / Entwicklerhandbuch

GADGETS 2.1 / DOCUMENTS 5

Inhaltsverzeichnis

1. Einleitung

Dieses Dokument beschreibt die Installation, Erstellung und Verwendung von *Gadgets*.

Gadgets sind eine Erweiterung für DOCUMENTS und bieten die Möglichkeit, individuelle Anwendungen an verschiedenen Stellen in DOCUMENTS zu integrieren. Außerdem ist es möglich mehrere Gadgets auf s.g. Dashboards zu platzieren. Dashboards sind Übersichtsseiten, die mehrere Gadgets beinhalten die vom Benutzer selbst hinzugefügt, sortiert und gelöscht werden können.

Gadgets basieren auf Portal-Skripten. Für die fachliche Funktionalität der Gadgets steht deshalb die gesamte Portal Skript API zur Verfügung.

Mit der Gadget API lassen sich Gadgets mit vordefinierten Funktionen als Formulare, Tabellen, Diagramme und als Teilbereich von Übersichtsseiten erstellen.

2. Installation / Einrichtung

2.1 Voraussetzungen

Für die Verwendung von *GADGETS* wird **DOCUMENTS 5.0a** oder eine aktuellere Version sowie eine Lizenz die die Verwendung der Gadget API beinhaltet benötigt. Zur Implementierung von Gadgets sind außerdem Kenntnisse der Portal-Script API nötig.

2.2 Import der Gadget – API

Die Gadget-API wird in einem Archiv mit dem Namen Gadgets_x.x.x.zip ausgeliefert, wobei „x.x.x“ für die jeweilige Versionsnummer steht. Der Inhalt des Archivs ist wie folgt aufgebaut:

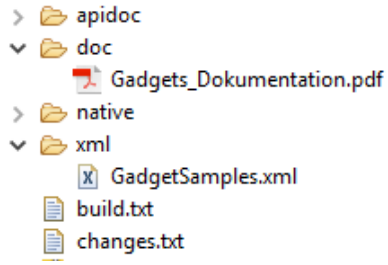

Abb. 1 Aufbau des Gadget-Pakets

2.2.1 Import der Gadget-Beispiele

Sollen die Gadget-Beispiele installiert werden, muss die folgende Datei importiert werden: GadgetSamples.xml. Die in dieser Dokumentation vorgestellten Skripte sind alle in diesem Beispiel-Paket enthalten.

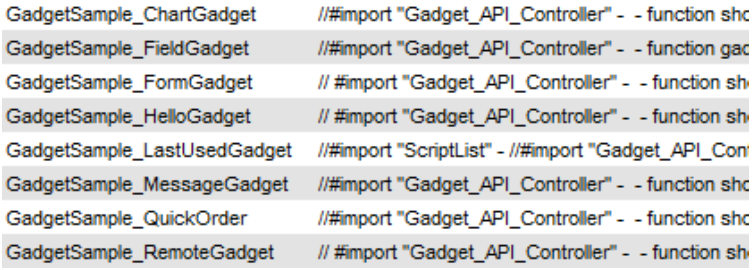

Abb. 2 Beispiel-Skripte im Documents Manager

3. „Getting started“

3.1 Was ist ein Gadget?

Ein Gadget erweitert individuell und mandantenabhängig die Funktionalität von DOCUMENTS. Ein Gadget kann an verschiedenen Stellen in DOCUMENTS integriert werden. Ein Gadget kann sowohl mit anderen Gadgets als auch über eine clientseitige API mit DOCUMENTS interagieren.

Da Gadgets in Portal Skripten implementiert werden, steht bei der Verwendung von Gadgets die Portal Skript API in vollem Umfang zur Verfügung. Außerdem bleiben die Gadgets bei einem Versionsupdate kompatibel.

3.2 Was ist ein Gadget nicht?

Ein Gadget ist kein Ersatz für bereits vorhandene Programmfunktionen von DOCUMENTS. Ein Gadget kann nicht verwendet werden um Berechtigungen in DOCUMENTS zu umgehen oder zu verändern.

Gadgets sind nicht für komplexe und umfangreiche Anwendungen gedacht, da diese in Portal Skripten implementiert werden und die Ausführung ggf. Performance-lastig sein kann.

3.3 „Hello Gadget“

In diesem Abschnitt geht es um die exemplarische Erstellung und Einbindung eines einfachen Gadgets.

3.4 Erstellung des Gadgets

Um ein Gadget anzulegen, muss zunächst der serverseitige Quelltext des Gadgets in einem Portal-Script abgelegt werden. Ein neues Portal-Script wird im **DOCUMENTS MANAGER** unter dem Baumeintrag „Documents -> Scripting“ angelegt. Der vergebene Skriptname wird später benötigt, um das Gadget einzubinden. Im Beispiel wird der Name „GadgetSample_HelloGadget“ verwendet siehe Abb. 2.

Wichtiger Hinweis:

Der Name eines Portal Skriptes das als Gadget ausgeführt werden soll muss immer mit „Gadget“ beginnen!

Abb. 2 Erstellen des Gadgets

Im Feld „Quellcode“ wird der Quellcode des Gadgets eingefügt. Im Beispiel „Hello Gadget“ wird folgender Quellcode verwendet:

```
#import "Gadget_API_Controller"

function showHelloMessage() {

   this.execute = function() {

        var htmlGad = new otris.gadget.gui.HTML("<h1>Hello Gadget!</h1>");

        return htmlGad.transfer();

       }

}
```

Dieses Gadget-Skript enthält, neben dem Import der Gadget-API, die Gadget-Aktion „showHelloMessage()“. Diese Aktion liefert als Ergebnis ein HTML-Element mit der Überschrift „Hello Gadget!“ zurück und zeigt dieses an. Das Gadget ist nach dem Speichern erfolgreich angelegt.

3.4.1 Bereitstellung des Gadgets

Ein Gadget, das in einem Portal-Skript abgelegt wurde, kann nun an verschiedenen Stellen im Portal eingebunden werden. Dazu muss der Name des Gadget-Skripts (im Beispiel: „GadgetSample_HelloGadget“) und der Name der Gadget-Aktion (im Beispiel: „showHelloMessage“) bekannt sein. Diese Angaben bilden zusammen die Gadget-Konfiguration (Siehe 4.2 Die Gadget-Konfiguration).

Um ein Gadget bspw. über einen Eintrag im Baum der DOCUMENTS-Oberfläche zugänglich zu machen, muss ein neuer öffentlicher Ordner angelegt werden. Dies geschieht im DOCUMENTS MANAGER über den Eintrag „Documents -> Öffentliche Ordner -> Neu“.

Um dem öffentlichen Ordner nun ein Gadget zuzuweisen, muss im Register „Eigenschaften“ des Ordners eine neue Eigenschaft „gadgetConfig“ mit dem Wert:

```
{gadgetScript:'GadgetSample_HelloGadget',gadgetAction:'showHelloMessage'}
```

angelegt werden (siehe Abb. 3).

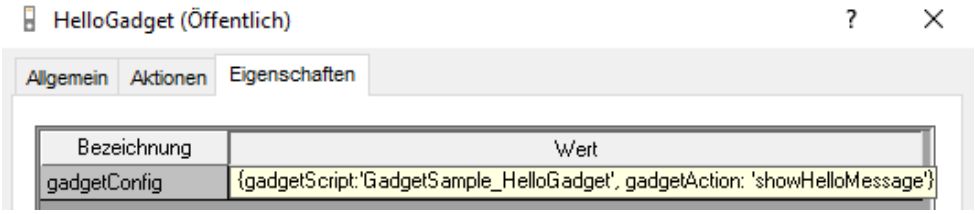

Abb. 3 Anlegen der Gadget-Konfiguration

Das angelegte Gadget kann nun über einen Eintrag im Baum angezeigt werden (siehe Abb. 5 und Abb. 4).

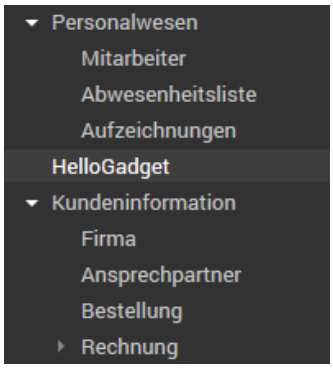

Abb. 5 "Hello Gadget" im Baum

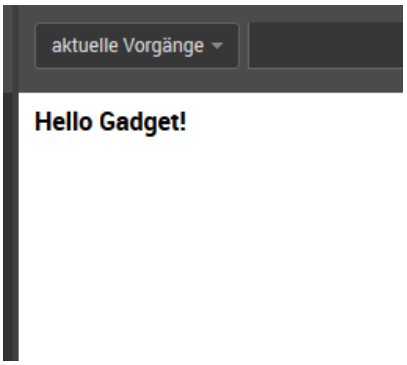

Abb. 4 Ergebnis des Gadgets

Gadgets können an folgenden Stellen in DOCUMENTS verwendet werden:

- An öffentlichen Ordnern (Eintrag im linken Menü-Baum)
- In der Seitenleiste von Mappen
- Als Register einer Mappe
- Als Mappen-Feld (Typ: Gadget)
- Als benutzerdefinierte Aktion (an Mappen, Registern und Ordnern)
- Als Kachel in einem Dashboard (siehe Dashboard-Dokumentation)
- Als „User-Exit“ an einem Mappen-Feld (in Vorbereitung)
- Als „Email-Exit“ für einen Mappentypen (in Vorbereitung)
- Weitere Integrationsmöglichkeiten sind geplant

Die verschiedenen Möglichkeiten der Integration sind im Kapitel 6 „Integration von Gadgets“ beschrieben.

4. Grundkonzepte

4.1 Aufbau eines Gadgets

Ein Gadget besteht immer aus einem Gadget-Skript und einer Gadget-Aktion die in diesem Gadget-Skript abgelegt wird. Im Folgenden wird der Aufbau eines Gadgets anhand des „Hello Gadget“ Beispiels aus Kapitel 3 „Getting started“ erläutert.

Das Gadget-Skript beginnt immer mit dem Import der Gadget-Controller API:

```
#import "Gadget_API_Controller"
```

Diese sorgt dafür, dass das Skript als Gadget-Skript ausführbar ist und stellt die Funktionen der Gadget-API bereit.

Die Gadget-Aktion ist eine Portal-Scripting Funktion, die einer bestimmten Struktur folgen muss, damit sie als Gadget-Aktion ausführbar ist:

```
function showHelloMessage() {
        this.execute = function() {
                //Gadget Code
        }
}
```

Ein Gadget-Skript kann beliebig viele Gadget-Aktionen sowie Hilfsfunktionen beinhalten, die von den Gadget-Aktionen genutzt werden können.

4.2 Die Gadget-Konfiguration

Im gesamten Gadget-Umfeld wird der Aufruf eines bestimmten Gadgets immer mit einer „Gadget-Konfiguration“ beschrieben. Diese Konfiguration wird verwendet um den Gadget-Aufruf eindeutig festzulegen. Eine Gadget-Konfiguration besteht aus:

gadgetId (optional):

Ein eindeutiger Name, der dem Gadget zugewiesen wird. Wird dieser Parameter weggelassen, wird hier ein Wert automatisch generiert. Wenn später allerdings auf das Gadget zugegriffen werden soll (bspw. bei der Kommunikation mit anderen Gadgets), sollte hier ein Name vergeben werden um das Gadget adressieren zu können.

gadgetScript:

Der Name des Portal-Skripts, das die aufzurufende Gadget-Aktion (gadgetAction) beinhaltet.

gadgetAction:

Der Name der aufzurufenden Gadget-Aktion.

gadgetWidth:

Die bevorzugte Breite des Gadgets in Pixeln. (Die Angabe des Wertes darf nicht in Anführungszeichen erfolgen).

gadgetHeight:

Die bevorzugte Höhe des Gadgets in Pixeln. (Die Angabe des Wertes darf nicht in Anführungszeichen erfolgen).

Parameter:

Alle weiteren angegebenen Eigenschaften der Gadget-Konfiguration werden automatisch als Parameter an das Gadget weitergeleitet. Die Gadget-Konfiguration wird immer als Objekt im „JSON“-Format angelegt.

4.2.1 Beispiel einer Gadget-Konfiguration

```
{gadgetId: 'helloGadget', gadgetScript: 'GadgetSample_HelloGadget', gadgetAction:
'showHelloMessage', message: 'Hello Gadget! '}
```

In diesem Fall ist nun der Parameter „message“ im Gadget-Kontext zugänglich.

4.3 Lebenszyklus eines Gadgets

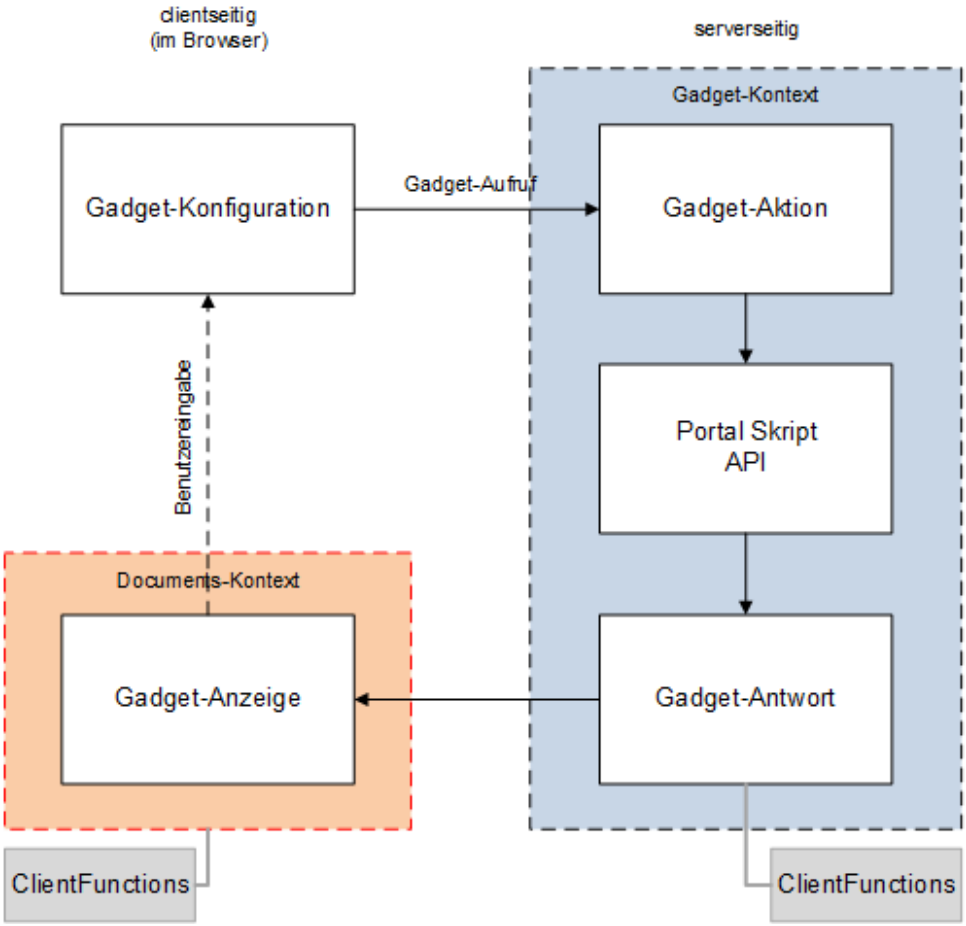

Abb. 6 Lebenszyklus eines Gadgets

- Die Ausführung eines Gadgets beginnt immer mit der Festlegung der *Gadget-Konfiguration.* (Siehe Abschnitt 4.2)
- Anhand der *Gadget-Konfiguration* erfolgt dann der eigentliche Aufruf des Gadgets durch das System.
- Während der Ausführung des Gadgets kann auf Informationen und Funktionen des *Gadget-Kontexts* zugegriffen werden. (Siehe Abschnitt 0)
- Wurde das Gadget ausgeführt und liefert eine Antwort zurück, wird dieses angezeigt. Während der Anzeige des Gadgets stehen nun die Informationen und Funktionen des *Documents-Kontexts* zur Verfügung (Siehe Abschnitt 4.6). Der *Gadget-Kontext* ist hier nicht mehr erreichbar.
- Das angezeigte Gadget kann nun bspw. mithilfe von GadgetActionButtons einen erneuten Gadget-Aufruf starten.

Abb. 6 gibt eine Übersicht über den Lebenszyklus eines Gadgets.

4.4 Kommunikation zwischen Gadgets (Gadget-ID)

Wie bereits im Abschnitt 4.2 beschrieben, kann in einer Gadget-Konfiguration eine Gadget-ID angegeben werden. Dieser eindeutige Name kann verwendet werden, um ein angezeigtes Gadget eindeutig zu identifizieren.

Wird ein Gadget mit einer Gadget-ID ausgeführt, die bereits von einem anderen Gadget auf der Seite verwendet wird, so wird die Anzeige des bisher angezeigten Gadgets durch die neue ersetzt.

Auf diese Art und Weise kann ein Gadget durch einen Aufruf auch den Inhalt eines anderen Gadgets ersetzen. Im folgenden Beispiel kann durch Betätigen der Schaltflächen in einem Gadget die Farbe des Quadrats in dem anderen Gadget geändert werden:

Abb. 7 Beispiel zur Kommunikation zwischen Gadgets

```
#import "Gadget_API_Controller"


function showRemote() {

       this.execute = function() {

              var form = new otris.gadget.gui.Form();

              var greenConfig = {"gadgetScript": 'GadgetSample_RemoteGadget',
                          "gadgetAction": 'showColorGadget',
                      "gadgetId": 'colorGadget',
           "color": '#00aa00'};

              var redConfig = {"gadgetScript": 'GadgetSample_RemoteGadget',
                        "gadgetAction": 'showColorGadget',
                    "gadgetId": 'colorGadget',
         "color": '#aa0000'};



              form.addGadgetActionButton("buttonGreen", "Grün", greenConfig);

              form.addGadgetActionButton("buttonRed", "Rot",
redConfig).setInLine(true);

      return form.transfer();

       }

}



function showColorGadget() {

       this.execute = function() {

              var color = gadgetContext.gadgetParams.color ?
                  gadgetContext.gadgetParams.color : '#00aa00';



              var htmlText = "<div style='width: 100px; height:\
100px; background-color: "+ color +";'>";



              var html = new otris.gadget.gui.HTML(htmlText);

              return html.transfer();

       }

}
```

Um die in Abb. 7 dargestellte Anzeige zu erreichen, wurden an einem Mappentypen folgende zwei Gadgets konfiguriert:

```
{gadgetScript: 'GadgetSample_RemoteGadget', gadgetAction: 'showRemote'}

{gadgetScript: 'GadgetSample_RemoteGadget', gadgetAction:'showColorGadget', gadgetId:
'colorGadget'}
```

Die Konfiguration von mehreren Gadgets wird in Kapitel 6 „Integration von Gadgets“ erläutert.

Wird ein Gadget auf einer Seite ausgeführt, die nur ein einziges Gadget beinhalten kann, so ersetzt jeder neue Gadget-Aufruf das aktuell angezeigte Gadget.

4.5 Der Gadget-Kontext (Serverseitig)

Der Gadget-Kontext ist ein implizites Objekt (gadgetContext), das **während der Ausführung** eines Gadgets zur Verfügung steht. Es beinhaltet alle wichtigen Informationen über das ausgeführte Gadget und die Parameter die beim Aufruf übergeben wurden.

4.5.1 Kontext-Informationen

Informationen zum Gadget selbst und dem Kontext in dem es ausgeführt wurde, sind direkt im gadgetContext abrufbar. Die wichtigsten Informationen sind:

- **gadgetContext.gadgetId** : Die Gadget-Id des aktuellen Gadgets
- **gadgetContext.gadgetScript**: Das Gadget-Skript des aktuellen Gadgets
- **gadgetContext.gadgetAction**: Die Gadget-Aktion des aktuellen Gadgets
- **gadgetContext.folderId**: Die ID des Ordners in dem das Gadget ausgeführt wird
- **gadgetContext.fileId**: Die ID der Mappe in dem das Gadget ausgeführt wird
- **gadgetContext.registerId**: Die ID des Registers in dem das Gadget ausgeführt wird (wenn das Gadget in einem anderen Register als dem Mappendeckel ausgeführt wird).
- **gadgetContext.fieldId**: Die ID des Feldes an dem das Gadget ausgeführt wird.
- **gadgetContext.documentId**: Die ID des Dokuments dass gerade angezeigt wird, wenn das Gadget auf einem Dokumentenregister ausgeführt wird.

4.5.2 Gadget Parameter

Gadget-Parameter sind Parameter, die direkt beim Aufruf des Gadgets über die Gadget-Konfiguration übergeben wurden. Diese sind über das Objekt *gadgetContext.gadgetParams* abrufbar. Wird bei der Gadget-Konfiguration bspw. ein Parameter „name“ übergeben, so kann dieser wie folgt abgerufen werden:

```
//Parameter „name“ abfragen und ausgeben

util.out(gadgetContext.gadgetParams.name);
```

4.5.3 Formular Parameter

Wenn ein Gadget ein Formular enthält, werden die Parameter des Formulars beim Betätigen eines Action-Buttons in diesem Formular an das aufgerufene Gadget weitergegeben. Enthält das Formular bspw. einen Parameter „firma“ so kann dieser wie folgt abgerufen werden:

```
//Parameter „firma“ abfragen und ausgeben

util.out(gadgetContext.formParams.firma);
```

4.6 Der Documents-Kontext (Clientseitig)

Der Documents-Kontext ist ein implizites Objekt (documentsContext), das clientseitig **während der Anzeige** eines Gadgets im Browser zur Verfügung steht. Es beinhaltet alle wichtigen Funktionen um Aktionen im Documents-Umfeld ausführen zu können wie z.B. das Anzeigen und Bearbeiten von Mappen.

Eine detaillierte Beschreibung der Funktionen des Documents-Kontext ist in der API-Dokumentation im Abschnitt „DocumentsContext“ zu finden.

5. API – Überblick

Dieses Kapitel gibt einen Überblick über die in der Gadget-API enthaltenen Elemente und deren Verwendung. Eine vollständige API-Dokumentation ist dem Gadget-Paket beigefügt.

5.1 Allgemeines

Grundsätzlich kann ein Gadget jedes Element aus der Gadget-API als Ergebnis zurückliefern, welches eine *transfer()* Methode besitzt. Das zurückgegebene Element ist das Ergebnis des Gadgets und wird in der DOCUMENTS Oberfläche dargestellt.

5.2 HTML Elemente

Mit dem Element „otris.gadget.gui.HTML“ kann beliebiger HTML-Text dargestellt werden.

```
var htmltext = '<h1>Hallo Willi Schreiber,</h1>\
                <p>Sie verwenden gerade das HTML Gadget.</p>\
                <p>Und hier noch ein zweiter Absatz.</p>';


var htmlerg = new otris.gadget.gui.HTML(htmltext);
return htmlerg.transfer();
```

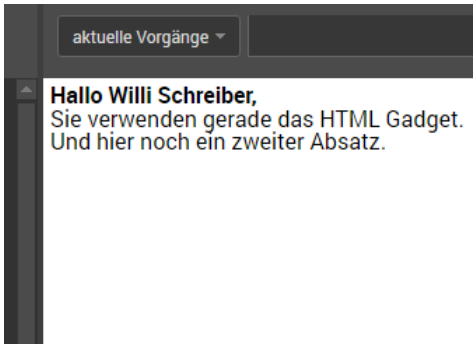

Abb. 8 Ergebnis des HTML Gadgets

Das HTML-Element bietet außerdem noch die Möglichkeit, Funktionen aus dem Gadget-Skript für die Ausführung des Gadgets bereitzustellen. Solche Funktionen werden dem HTML-Element mit der Methode „addClientFunction (clientFunction)“ bekannt gemacht.

```
//#import "Gadget_API_Controller"

function showHtmlText()

{

        this.execute = function()

        {

                var htmltext = '<h1>Hallo!,</h1>\

                <p>Sie verwendet gerade das HTML Gadget.</p> \

                <p><a href=\"#\"
onClick=\"documentsContext.getClientFunctions()['eineFunktion']()\">Ausführen!</a>';

            var htmlresult = new otris.gadget.gui.HTML(htmltext);

            htmlresult.addClientFunction(eineFunktion);

            return htmlresult.transfer();

        }

}

function eineFunktion(){

        alert("eine Funktion wurde ausgeführt");

}
```

Wenn man im Gadget nun auf „Ausführen!“ klickt wird die Funktion ausgeführt.

WICHTIGER HINWEIS:

*Funktionen die auf diese Art und Weise aufgerufen werden, werden clientseitig im DOCUMENTS-Kontext im Browser ausgeführt. Das bedeutet, dass das Objekt „gadgetContext“ und alle darin enthaltenen Informationen in solchen Funktionen **nicht** zur Verfügung stehen.*

5.3 Nachrichten und Fehlermeldungen

Gadgets können Nachrichten und Fehlermeldungen anzeigen. Hierzu wird die Klasse „otris.gadget.gui.Message“ als Ergebnis des Gadgets zurück gegeben. Folgende Nachrichtentypen werden unterstützt:

- **error**: Fehlermeldung
- **warning**: Warnung
- **info**: Beliebige Information

Je nach Nachrichtentyp und Browser werden verschiedene Symbole oder Hinweise angezeigt.

```
//#import "Gadget_API_Controller"


function showMessageGadget()

{

        this.execute = function()

        {

                if(gadgetContext.gadgetEvent == 'klickmich')

                {

                        return new otris.gadget.gui.Message('Die Schaltfläche\

 wurde geklickt!', 'info').transfer();

                }

                else

                {

                        myForm = new otris.gadget.gui.Form();

                        myForm.addHeadLine('Bitte klicken Sie auf die Schaltfläche:');

                        myForm.addGadgetActionButton('klickmich', 'Klick mich!');


                        return myForm.transfer();

                }

        }

}
```

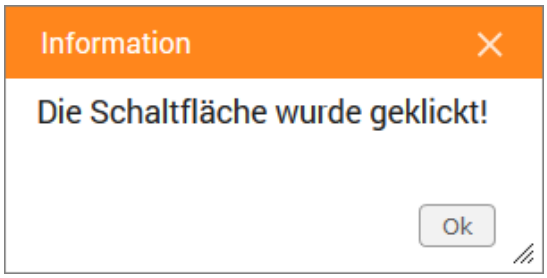

Abb. 9 Angezeigte Nachricht (Typ: Info)

5.4 Formulare und GadgetActionButtons

Gadgets können Formulare darstellen und die eingegebenen Daten mithilfe von GadgetActionButtons an ein beliebiges Gadget versenden. Zur Erstellung von Formularen wird die Klasse „otris.gadget.gui.Form“ verwendet. Diese Klasse enthält eine Vielzahl von Funktionen, mit denen Formularelemente hinzugefügt werden können.

GadgetActionButtons sind Schaltflächen, die in Formulare eingefügt werden können und beim Betätigen einen Gadget-Aufruf starten. Wird ein GadgetActionButton im Formular betätigt, werden die Inhalte der Formularelemente automatisch an das aufgerufene Gadget gesendet.

Im folgenden Beispiel wird ein einfaches Formular verwendet, um eine persönliche Anrede zu generieren:

```
function showFormGadget() {

	this.execute = function() {

		if(gadgetContext.gadgetEvent == 'send') {

			var htmlText = "Hallo ";


			if(gadgetContext.formParams.geschlecht == "male") {

				htmlText += "Herr ";

			}

			else if (gadgetContext.formParams.geschlecht == "female") {

				htmlText += "Frau ";

			}

			htmlText = "<h1>" + htmlText

			+ gadgetContext.formParams.firstname + " "

			+ gadgetContext.formParams.name + "</h1>";


			var html = new otris.gadget.gui.HTML(htmlText, 'testform');

			return html.transfer();

		}

		else {

			myForm = new otris.gadget.gui.Form();

			myForm.addTextField('name', "Name");

			myForm.addTextField('firstname', "Vorname");

			var options = [['male','männlich'],['female','weiblich']];

			myForm.addSingleSelectList('geschlecht',

					"Geschlecht", options);

			myForm.addGadgetActionButton('send', 'absenden');
		return myForm.transfer();

		}

	}
}
```

Wird das Gadget zum ersten Mal angezeigt, ist der Parameter *gadgetContext.gadgetEvent* noch nicht gesetzt und es wird ein einfaches Formular angezeigt welches einen GadgetActionButton enthält (Siehe Abb. 10).

Abb. 10 Anzeige des Formular-Gadgets

Wird der GadgetActionButton betätigt, wird das Gadget erneut aufgerufen, dieses Mal aber mit dem Parameter *gadgetContext.gadgetEvent,* der nun den Namen des GadgetActionButtons enthält. Die Anrede wird angezeigt (Siehe Abb. 11)

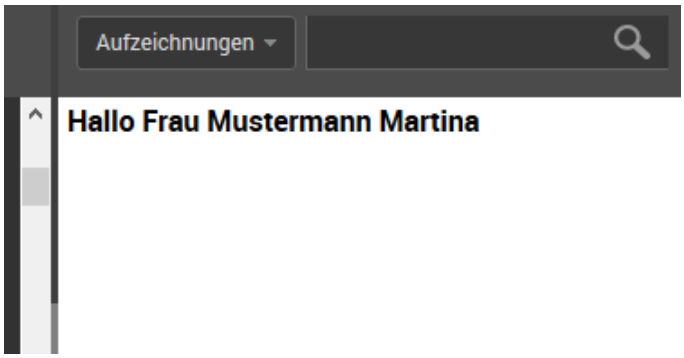

Abb. 11 Ergebnis des Formular-Gadgets

5.5 Diagramme

Diagramme können mit Hilfe des Chart-Objektes *otris.gadget.gui.Chart* erstellt werden. Die Diagramme werden mit dem JavaScript-Framework Chart.js erzeugt. In der detaillierten Dokumentation des Frameworks können alle Möglichkeiten der Konfiguration nachgeschlagen werden. Die Dokumentation steht online unter **http://www.chartjs.org/docs/** zur Verfügung.

Mit dem folgenden Beispiel (**GadgetSample_ChartGadget.js**) kann ein einfaches Kreisdiagramm erstellt werden.

```
//#import "Gadget_API_Controller"
function showChart() {
        this.execute = function() {
                var chart, data, config;
                data = {
                    labels: [
                             "Kategorie A",
                             "Kategorie B",
                             "Kategorie C"
                    ],
                    datasets: [
                        {
                            data: [32, 15, 23],
                            backgroundColor:      [
                                'rgba(0, 67, 88, 1)',
                                'rgba(255, 225, 26, 1)',
                                'rgba(192, 48, 0, 1)'
                            ],
                               hoverBackgroundColor: [
                                    'rgba(0, 67, 88, 0.7)',
                                    'rgba(255, 225, 26, 0.7)',
                                    'rgba(192, 48, 0, 0.7)'
                               ]
                        }
                    ]
                };
                config = {
                               title: {
                                       text: 'Kategorien',
                            display: true
                        }
                };
                chart = new otris.gadget.gui.Chart("pie", data, config);
                chart.setStyle('background', '#ffffff');
                return chart.transfer();
        }
}
```

5.6 Tabellen (ScriptList)

Tabellen können mit Hilfe der Script-Extension *ScriptList* erstellt werden. Ein Beispiel für die Verwendung einer *ScriptList* als Gadget findet sich in Kapitel 7.1.

5.7 Das Ext JS Framework

Die auf dem Ext JS Framework der Firma Sencha Inc. (**www.sencha.com**) basierenden Komponenten sind zwar noch Teil der Gadget-API sollten jedoch nicht mehr eingesetzt werden. Die Funktionalität wird mittlerweile durch andere Komponenten der API zur Verfügung gestellt.

Die Klassen *otris.gadget.gui.ExtTable, otris.gadget.gui.ExtComponent, otris.gadget.gui.BarChart* und *otris.gadget.gui.PieChart* sind als *deprecated* markiert und sollten möglichst zeitnah durch die neuen Klassen ersetzt werden.

Um das Ext JS Framework aus Kompatibilitätsgründen dennoch zu verwenden muss in den Documents Einstellungen die Property *extJsLibMode* eingefügt und auf *true* gesetzt werden.

6. Integration von Gadgets

6.1 Allgemeines

Bei der Integration von Gadgets in DOCUMENTS werden die Gadget-Konfigurationen an verschiedenen Stellen als Eigenschaften hinterlegt. Im Normalfall wird dazu eine einzelne Gadget-Konfiguration wie folgt eingetragen:

```
{gadgetScript:'GadgetSample_HelloGadget', gadgetAction:'showHelloMessage'}
```

Folgende Eigenschaft unterstützt auch die Angabe mehrerer Gadgets:

- *marginGadgetConfigs* (Siehe Abschnitt 6.3 In der Seitenleiste von Mappen“)

Bei dieser Eigenschaft muss immer die folgende Form verwendet werden:

```
[{gadgetScript:'GadgetSample_HelloGadget',gadgetAction:'showHelloMessage'},

{gadgetScript: 'GadgetSample_HtmlGadget', gadgetAction: 'showHtmlText'}]
```

Soll an dieser Stelle nur ein einzelnes Gadget verwendet werden, so muss dies trotzdem in eckige Klammern geschrieben werden:

```
[{gadgetScript:'GadgetSample_HelloGadget',gadgetAction:'showHelloMessage'}]
```

6.2 An öffentlichen Ordnern

Um ein Gadget an einem öffentlichen Ordner zu verwenden, muss die Gadget-Konfiguration als Eigenschaft „gadgetConfig“ am Ordner hinterlegt werden (siehe Abb. 12).

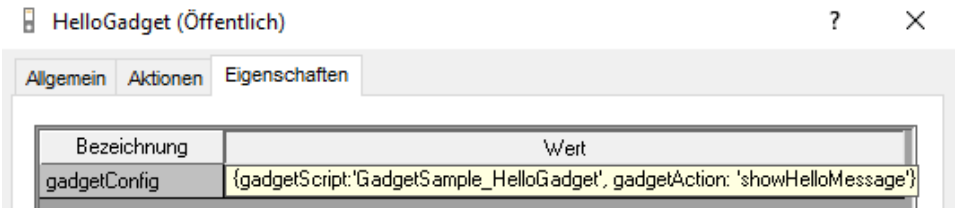

Abb. 12 Gadget-Konfiguration am öffentlichen Ordner

Das hier konfigurierte Gadget nimmt dann den gesamten Platz im Hauptbereich von DOCUMENTS ein (siehe Abb. 13).

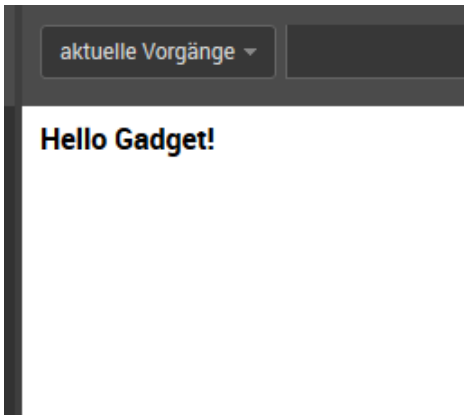

Abb. 13 Gadget am Ordner

6.3 In der Seitenleiste von Mappen

In der Seitenleiste von Mappen können ebenfalls Gadgets verwendet werden. Dazu muss die Eigenschaft „marginGadgetConfigs“ für den jeweiligen Mappentypen festgelegt werden. In dieser Eigenschaft können mehrere Gadgets verwendet werden, die in der Seitenleiste untereinander dargestellt werden.

Im unten stehenden Beispiel (siehe Abb. 14) wurde folgende Gadget-Konfiguration verwendet:

```
[{gadgetScript:'GadgetSample_HelloGadget',gadgetAction:'showHelloMessage'},
{gadgetScript: 'GadgetSample_HtmlGadget', gadgetAction: 'showHtmlText'}]
```

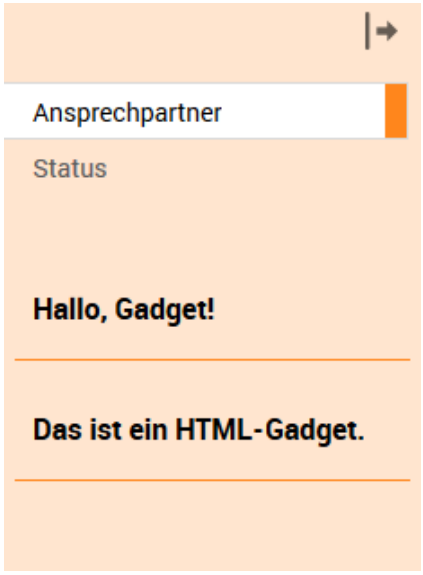

Abb. 14 Zwei Gadgets in der Seitenleiste

6.4 In Mappen-Registern

Gadgets können auch an Mappenregistern verwendet werden. Hierzu wird dem Mappentyp ein neues Register des Typs „Externer Aufruf“ hinzugefügt (siehe Abb. 15).

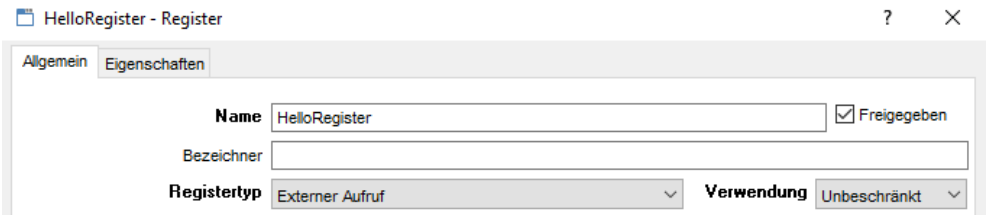

Abb. 15 Ein neues Register anlegen

Im Reiter „Eigenschaften“ wird dann in der Eigenschaft „gadgetConfig“ die Gadget-Konfiguration hinterlegt (siehe Abb. 16).

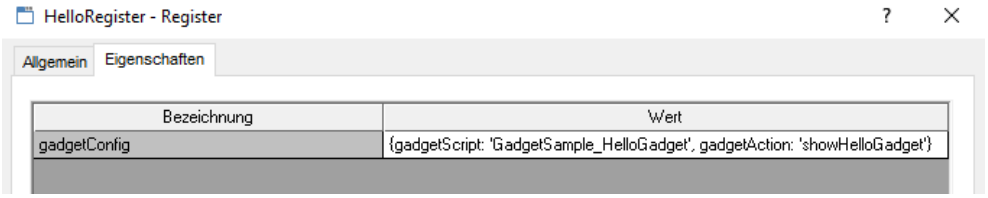

Abb. 16 Gadget-Konfiguration am Register

Wichtiger Hinweis:

Wenn das neue Gadget-Register auch in bereits vorhandenen Mappen des Mappentyps verwendet werden soll, müssen diese nach der Anlage des Registers über die Schaltfläche „Mappen ändern“ angepasst werden.

Das konfigurierte Gadget wird dann bei Auswahl des angelegten Registers angezeigt (siehe Abb. 17).

Abb. 17 Gadget an einem Register

6.5 Als Mappen-Feld

Ein Gadget kann auch als Feld innerhalb einer Mappe verwendet werden. Hierzu wird in einem Mappentyp ein neues Feld vom Typ „Gadget“ angelegt (Siehe Abb. 18). Im Register „Eigenschaften“ kann dann eine Gadget-Konfiguration in der Eigenschaft „gadgetConfig“ hinterlegt werden.

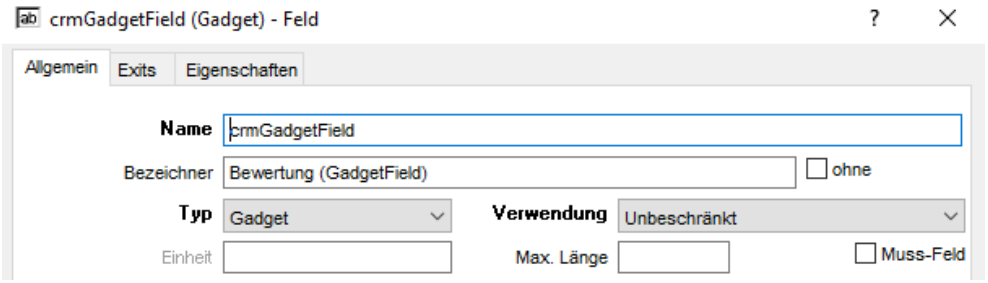

Abb. 18 Mappen-Feld vom Typ Gadget

Das in „gadgetConfig“ eingestellte Gadget wird dann als Feld in die Mappenansicht integriert. (Siehe

Abb. 19)

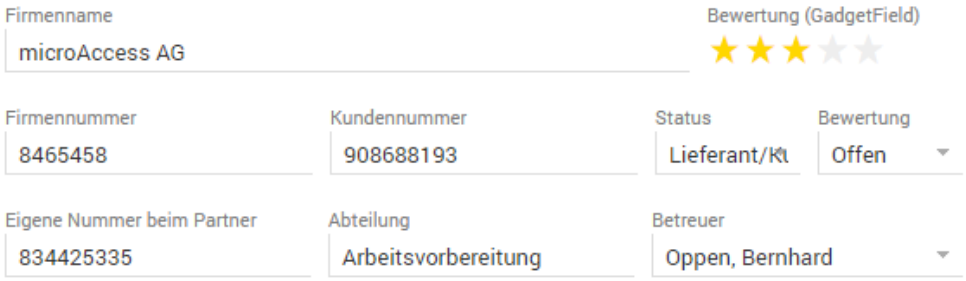

Abb. 19 Gadget als Mappen-Feld

Soll das Gadget Werte in der Mappe speichern, wird ein zusätzliches verstecktes Feld benötigt, in dem das Gadget dann über den Documents-Kontext Inhalte Speichern kann.

6.6 Als benutzerdefinierte Aktion

Ein Gadget kann als benutzerdefinierte Aktion ausgeführt werden. Hierzu muss eine neue benutzerdefinierte Aktion zu dem jeweiligen Mappentypen hinzugefügt werden (siehe Abb. 20). Als Typ wird „Gadget“ ausgewählt. Im Register „Eigenschaften“ kann dann eine Gadget-Konfiguration in der Eigenschaft „gadgetConfig“ hinterlegt werden.

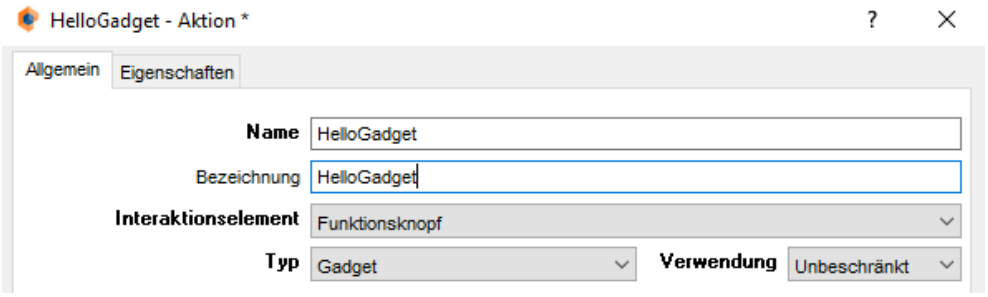

Abb. 20 Benutzerdefinierte Aktion (Gadget)

Beim Ausführen der Aktion wird das Gadget nun in einem modalen Dialog geöffnet (siehe Abb. 21).

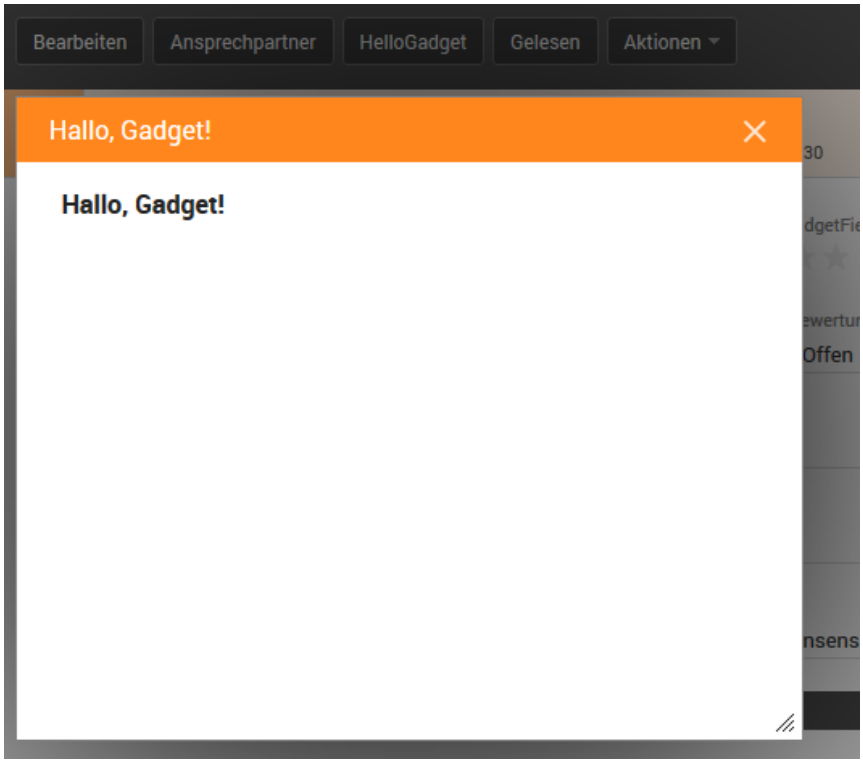

Abb. 21 Gadget in einem modalen Dialog

6.7 Nutzung als User-Exit bzw. Email-Exit

Die Nutzung von Gadgets als User- bzw. Email-Exit ist im Moment noch nicht implementiert.

6.8 Einbindung in Dashboards

Die Integration von Gadgets in einem Dashboard wird in der Dashboard Dokumentation ausführlich beschrieben.

7. Beispiele

In diesem Abschnitt werden vollständige Gadget-Beispiele detailliert beschrieben, die jeweils ein bestimmtes Problem lösen.

7.1 Das „LastUsed-Gadget“

Der vollständige Programm-Quelltext des folgenden Beispiels ist in der Datei **GadgetSample_LastUsedGadget.js** *zu finden.*

Das „LastUsed-Gadget“ soll eine dreispaltige Tabelle anzeigen, die eine Liste der zuletzt angezeigten Dokumente des aktuell angemeldeten Benutzers anzeigt. Bei einem Klick auf einen Eintrag soll das entsprechende Dokument geöffnet werden.

7.1.1 Implementierung des Gadgets

Für das LastUsed-Gadget wird eine *ScriptList* erzeugt und diese dann in einem *ScriptListWrapper* an den Client transferiert.

Die Aktion showLastUsed

```
01:  //#import "ScriptList"
02:  //#import "Gadget_API_Controller"
03:
04:  function showLastUsed() {
05:
06:    this.execute = function() {
07:      var myFile, myRow;
08:      // Create wrapper & scriptList
09:      var wrapper = new otris.gadget.gui.ScriptListWrapper();
10:      var scriptList = new otris.scriptlist.List("Zuletzt verwendet");
11:
12:      // Get data
13:      var user = context.getSystemUser();
14:      var lastusedfolder = user.getPrivateFolder('lastused');
15:      var myFRS = lastusedfolder.getFiles();
16:
17:      // Add column definition
18:      scriptList.addColumn("type", "Typ", "STRING");
19:      scriptList.addColumn("title", "Titel", "STRING");
20:      scriptList.addColumn("owner", "Eigentümer", "STRING");
21:
```

```
22:       // Add rows
23:       for (myFile = myFRS.first(); myFile; myFile = myFRS.next()) {
24:         myRow = {
25:           id : myFile.getAutoText("id"),
26:           type : myFile.getAutoText("fileTypeTitle"),
27:           title : myFile.getAutoText("title"),
28:           owner : myFile.getAutoText("fileOwner")
29:         };
30:         scriptlistRow = scriptList.addRow(myRow.id, myRow);
31:         scriptlistRow.setAction("showFile:" + myRow.id);
32:       }
33:
34:       // Add the information that the list is complete
35:       scriptList.endList();
36:
37:       // Add ScriptList to wrapper & transfer
38:       wrapper.addScriptList(scriptList);
39:       return wrapper.transfer();
40:     }
41:   }
```

In Zeile 9 und 10 werden das *ScriptListWrapper*-Objekt und die das *ScriptList*-Objekt angelegt. Die *ScriptList* wir mit dem Titel „Zuletzt verwendet“ versehen.

```
09:       var wrapper = new otris.gadget.gui.ScriptListWrapper();
10:       var scriptList = new otris.scriptlist.List("Zuletzt verwendet");
```

In den Zeilen 18-20 werden die Spaltenüberschriften der Tabelle festgelegt. Die ScriptList enthält die Spalten Typ, Titel und Eigentümer.

```
17:       // Add column definition
18:       scriptList.addColumn("type", "Typ", "STRING");
19:       scriptList.addColumn("title", "Titel", "STRING");
20:       scriptList.addColumn("owner", "Eigentümer", "STRING");
```

Die Daten werden in den Zeilen 23-31 zur *ScriptList* hinzugefügt. Über *addRow()* wird jeweils eine Zeile mit der *id* und den Daten hinzugefügt. In Zeile 31 wird die Aktion festgelegt, die bei einem Klick auf die Zeile aufgerufen werden soll.

```
22:       // Add rows
23:       for (myFile = myFRS.first(); myFile; myFile = myFRS.next()) {
24:         myRow = {
25:           id : myFile.getAutoText("id"),
26:           type : myFile.getAutoText("fileTypeTitle"),
27:           title : myFile.getAutoText("title"),
28:           owner : myFile.getAutoText("fileOwner")
29:         };
30:         scriptlistRow = scriptList.addRow(myRow.id, myRow);
31:         scriptlistRow.setAction("showFile:" + myRow.id);
32:       }
```

In Zeile 34 wird festgelgt, dass die *ScriptList* keine weiteren Daten vom Server holen soll, da in diesem Beispiel direkt alle Einträge direkt an den Client geschickt werden. In Zeile 37 und 38 wird die *ScriptList* dem *ScriptListWrapper* zugewiesen und dieser an den Client transferiert.

```
34:       // Add the information that the list is complete
35:       scriptList.endList();
36:
37:       // Add ScriptList to wrapper & transfer
38:       wrapper.addScriptList(scriptList);
39:       return wrapper.transfer();
```

7.2 Das „Quickorder-Gadget“

Der vollständige Programm-Quelltext des folgenden Beispiels ist in der Datei **GadgetSample_QuickOrder.js** *zu finden.*

Das „QuickOrder-Gadget“ soll ein Formular anzeigen, mit dem es möglich ist direkt aus dem Gadget heraus eine neue Bestellung d.h. eine neue Mappe vom Mappentyp „ftOrder“ anzulegen.

7.2.1 Implementierung des Gadgets

Für das Quickorder-Gadget werden zwei Gadget-Aktionen benötigt. Die Aktion *showOrderFile* Zeigt ein Formular mit Eingabefeldern zur Anlage einer Bestellung an. Die Aktion *saveOrderFile* legt eine neue Bestellung an und speichert die im Formular gesammelten Daten. Die Aktion *showSmallOrderFile* zeigt das gleiche Formular in einer reduzierten Version an, sodass das Gadget auch in der Seitenleiste von Mappen benutzt werden kann, auf diese Aktion wird in diesem Beispiel aber nicht weiter eingegangen.

7.3 Die Aktion showOrderFile

```
003:  function showOrderFile() {
004:    this.execute = function() {
005:      var orderId, orderDate, total, company, contact, email, telephone, comment,
actionSave;
006:      var form = new otris.gadget.gui.Form();
007:
008:      form.setTitle("Bestellung anlegen");
009:
010:      orderId = form.addTextField("orderId", "Bestellnummer", "");
011:      orderId.setMandatory(true);
012:
013:      total = form.addTextField("total", "Gesamtpreis", "");
014:      total.setMandatory(true);
015:      total.setInLine(true);
016:
017:      orderDate = form.addDateField("orderDate", "Bestelldatum", "");
018:      orderDate.setInLine(true);
019:
020:      company = form.addTextField("company", "Firma", "");
021:      contact = form.addTextField("contact", "Kontakt", "");
022:      email = form.addEMailField("email", "Email", "");
023:      email.setInLine(true);
024:
025:      telephone = form.addTextField("telephone", "Telefon", "");
026:      telephone.setInLine(true);
027:      comment = form.addTextArea("comment", "Bestellung", "");
028:
029:      actionSave = form.addGadgetActionButton("actionSave", "Speichern", {
030:        gadgetScript: "GadgetSample_QuickOrder",
031:        gadgetAction: "saveOrderFile"
032:      });
033:
034:      actionSave.setInLine(false);
035:      return form.transfer();
036:    }
037:  }
```

In Zeile 6 wird ein neues Formular-Objekt erzeugt. In den darauffolgenden Zeilen (bis Zeile 27) werden dem Formular Eingabefelder hinzugefügt. Die Vorgehensweise wird hier nochmal am Beispiel des Feldes „total“ erklärt:

```
013:      total = form.addTextField("total", "Gesamtpreis", "");
014:      total.setMandatory(true);
015:      total.setInLine(true);
```

In Zeile 13 wird dem Formular das Feld mit der Methode „form.addTextField“ hinzugefügt. Damit die Formularelemente später noch bearbeitet werden können, liefern die Methoden zum Hinzufügen von Formularelementen immer das neu hinzugefügte Element zurück. In Zeile 14 wird das Feld als Pflichtfeld markiert. Mit der Methode setInLine(true) (Zeile 16) wird das Formularfeld in derselben Zeile wie das vorherige Formularfeld angezeigt.

```
029:      actionSave = form.addGadgetActionButton("actionSave", "Speichern", {
030:        gadgetScript: "GadgetSample_QuickOrder",
031:        gadgetAction: "saveOrderFile"
032:      });
033:
034:      actionSave.setInLine(false);
035:      return form.transfer();
```

In Zeile 29 wird als letztes Formularfeld ein GadgetActionButton hinzugefügt. Dieser enthält einen Namen, eine Beschriftung und eine Gadget-Konfiguration die den Gadget-Aufruf enthält an den das Formular gesendet werden soll.

In Zeile 35 wird dann das fertige Formular als Ergebnis der Gadget-Aktion zurückgegeben.

7.4 Die Aktion saveOrderFile

```
071:  function saveOrderFile() {
072:
073:    this.execute = function() {
074:      var newFtOrder, form, message;
075:      var $fParams = gadgetContext.formParams;
076:
077:      if($fParams.orderId && $fParams.orderId !=""
078:        && $fParams.total && $fParams.total != "") {
079:
080:        newFtOrder = context.createFile("ftOrder");
081:        newFtOrder.orderId = $fParams.orderId;
082:        newFtOrder.orderDate = $fParams.orderDate;
083:        newFtOrder.total = $fParams.total;
084:        newFtOrder.company = $fParams.company;
085:        newFtOrder.contact = $fParams.contact;
086:        newFtOrder.emailAddress = $fParams.email;
087:        newFtOrder.telephone = $fParams.telephone;
```

```
088:         newFtOrder.comment = $fParams.comment;
089:         newFtOrder.sync();
090:
091:         if(gadgetContext.gadgetEvent == "actionSaveAndShow") {
092:           form = new otris.gadget.gui.Form();
093:           form.addHeadLine("Mappe wurde erfolgreich angelegt!");
094:           form.addGadgetActionButton("backToQuickOrder", "Zurück zum Formular", {
095:             gadgetScript: "GadgetSample_QuickOrder",
096:             gadgetAction: "showSmallOrderFile"
097:           });
098:
099:           form.onGadgetLoad(
100:             'function onGadgetLoad() {'
101:               + 'documentsContext.openFileView("'
102:               + newFtOrder.getAutoText("%id%")
103:               + '", null, null, {startFileEditMode : true, registerBarState :
"open"});'
104:             + '}'
105:           );
106:           return form.transfer();
107:         }
108:         else {
109:           message = "Mappe wurde erfolgreich angelegt!";
110:           return new otris.gadget.gui.Message(message, "info").transfer();
111:         }
112:       }
113:       else {
114:
115:         if(!$fParams.orderId || $fParams.orderId == "") {
116:           message = "Bestellnummer ist ein Pflichtfeld";
117:         }
118:         if(!$fParams.total || $fParams.total == "") {
119:           message = "Gesamtpreis ist ein Pflichtfeld";
120:         }
121:         return new otris.gadget.gui.Message(message, "error").transfer();
122:       }
123:     }
124:   }
```

In Zeile 75 wird zunächst eine „Abkürzung“ angelegt d.h. eine Variable mit dem Namen „$fParams“ die eine Referenz auf die Variable „gadgetContext.formParams“ enthält, damit nicht bei jedem Zugriff auf die Formular Parameter der volle Name der Variable ausgeschrieben werden muss.

In Zeile 77 und 62 wird überprüft ob die Pflichtfelder des Formulars ausgefüllt worden sind.

```
077:      if($fParams.orderId && $fParams.orderId !=""
078:        && $fParams.total && $fParams.total != "") {
```

Ist dies der Fall, wird in Zeile 80 eine neue Mappe vom Typ ftOrder angelegt. Die Formularwerte werden dann ab Zeile 81 in die neu angelegte Mappe übertragen. In Zeile 89 werden die Änderungen in die Mappe übernommen.

```
080:        newFtOrder = context.createFile("ftOrder");
081:        newFtOrder.orderId = $fParams.orderId;
082:        newFtOrder.orderDate = $fParams.orderDate;
083:        newFtOrder.total = $fParams.total;
084:        newFtOrder.company = $fParams.company;
085:        newFtOrder.contact = $fParams.contact;
086:        newFtOrder.emailAddress = $fParams.email;
087:        newFtOrder.telephone = $fParams.telephone;
088:        newFtOrder.comment = $fParams.comment;
089:        newFtOrder.sync();
```

Die Zeilen 91-107 betreffen ausschließlich die reduzierte Version des Quickorder-Gadgets für die Seitenleiste, hier wird dafür gesorgt, dass die neu angelegte Mappe nach dem Absenden des Formulars in Documents angezeigt wird.

Handelt es sich nicht um die reduzierte Variante wird in den Zeilen 109-110 eine Nachricht als Ergebnis der Gadget-Aktion zurückgegeben.

```
109:          message = "Mappe wurde erfolgreich angelegt!";
110:          return new otris.gadget.gui.Message(message, "info").transfer();
```

In den Zeilen 115-121 wird eine Fehler-Nachricht zurückgegeben falls eines der Pflichtfelder nicht ausgefüllt wurde.

```
115:        if(!$fParams.orderId || $fParams.orderId == "") {
116:          message = "Bestellnummer ist ein Pflichtfeld";
117:        }
118:        if(!$fParams.total || $fParams.total == "") {
119:          message = "Gesamtpreis ist ein Pflichtfeld";
120:        }
121:        return new otris.gadget.gui.Message(message, "error").transfer();
```

8. Abbildungsverzeichnis