fellerLYnk

Beschreibung von Widgets

10.FLYNKWIDG-D.1806/180608

by Schneider Electric

1 Widget-ID

Jedes Benutzertyp-Widget sollte eine eindeutige Kennung innerhalb eines einzelnen Controllers (ID) haben. Jede Zeichenfolge ist als ID geeignet.

Eine Widget-ID wird bei der Installation des Widgets verwendet. Sie erscheint im dazugehörigen Programmcode und kann nach der Installation des Widgets in der App nicht leicht geändert werden. Widget-IDs erscheinen in UI während der Installation. Um das Risiko von Kollisionen zu verringern, wenn Sie Ihre Widgets bei verschiedenen Installationen benutzen, empfehlen wir die Verwendung von Zeichen als ID, die einen eindeutigen Teil des Widget-Entwicklers (Präfix) enthält.

Darüber hinaus sind einige IDs bereits in System-Widgets reserviert. Die App informiert Sie, sobald versucht wird, das Widget zu installieren.

2 Widget-Dateistruktur

Jedes Widget enthält die folgenden Dateien

- script.js ist eine Datei, die im Stammverzeichnis des Widgets benötigt wird. Sie enthält JavaScript-Code, der die Widget-Oberfläche und die Logik ihrer Funktionsweise beschreibt. Das Skript wird einmal auf der Seite geladen, unabhängig von der Anzahl der Widgets dieses Typs, die vom Benutzer in einer bestimmten App erstellt wurden.
- style.css ist eine optionale Datei im Stammverzeichnis des Widgets. Sie enthält CSS-Typen, die für Ihr Widget einzigartig sind. Sofern diese Datei im Widget-Ordner angezeigt wird, wird sie beim Aktualisieren der Seite automatisch geladen.
- Alle anderen Graphik-Dateien, JavaScript- sowie CSS-Dateien, die in Ihrem Widget Verwendung finden. Sie müssen diese in eine der oben genannten Dateien laden.

Wir empfehlen, die folgende Ordnerstruktur für Widgets zu verwenden:

- script.js
- style.css
- Bilder
  Graphikdateien
- js
  zusätzliche JavaScript-Datei und Bibliotheken.

3 Verfügbare Bibliotheken und Frameworks

In Ihrem Widget können Sie die folgenden Bibliotheken und Frameworks verwenden, die im Projekt enthalten sind:

- JQuery (http://jquery.com)
- Plugin JQuery noUiSlider (http://plugins.jquery.com/nouislider/)
- Plugin JQuery knob (http://plugins.jquery.com/knob/)
- Boostrap v3 (CSS+Javascript) (http://getbootstrap.com)
- Font Awesome v4 (http://fontawesome.io)
- UnderScore.js (http://underscorejs.org/)
- Backbone.js (http://backbonejs.org/)

4 File script.js

Bevor wir mit dem weiteren Teil der Dokumentation fortfahren, empfehlen wir die Dokumentation der Backbone.js Bibliothek zu lesen http://backbonejs.org/.

In Ihrem JavaScript-Code in der Sammlung von Widget-Klassen müssen Sie Ihre eigenen Erweiterungen für die Basisklassen Widget (Modell) und WidgetView (Ansicht) hinzufügen, indem Sie die erforderlichen Felder und Methoden abarbeiten, wie im folgenden Beispiel erläutert wird:

```
Widget["your widget ID"] = Widget.extend({
    /* description of widget settings*/
    config : {
        ...
    },

    /*is called when objects values are changed on Logic Machine*/
    changeValue : function(name, value) {
        ...
    }
}

WidgetView["your widget ID"] = WidgetView.extend({
    /*width of widget in cells*/
    width : ...,
    /*height of widget in cells*/
    height : ....,

    /* is called when the object is initialized */
    init : function() {
        ....
    }

    /*main template of widget*/
    template : ...,

    /*template for viewing in editor mode*/
    preview : ....,

    /*is called after rendering of widget during initialization*/
    afterRender : function() { ...}

    ...
})
```

Anstelle von "Ihre widget ID" müssen Sie Ihre Widget-ID verwenden und sie auch angeben, wenn Sie das Widget über die Benutzeroberfläche installieren. Neben den beschriebenen Serviceattributen können Sie während der Umsetzung dieser Klassen auch beliebige Ihrer Hilfsmethoden und Eigenschaften verwenden, die verantwortlich sind für die Datenkonvertierung, die Validierung, die Berechnungen sowie das Arbeiten mit dem DOM Ihres Widgets usw.

Die Basisklassen Widget und WidgetView sind Erweiterungen des Basisklassen-Modells (Backbone.Model) und der Ansicht (Backbone.View). Backbone enthält jeweils eine Reihe von Hilfsmethoden für das Arbeiten mit fellerLYnk-Objekten, die bei Ihrer Implementierung verwendet werden können.

Für jeden Widget-Typ, der von dem Touch-Benutzer erstellt wird, werden folgende Klassen erstellt.

```
model = new Widget["your widget ID"]();
view = new WidgetView["your widget ID"]({model:model});
```

Daher sollten Modellmethoden-Aufrufe innerhalb des Modells wie folgt ausgeführt werden:

```
this.func()
```

Der Aufruf von Eigenschaften und Modellmethoden in der Ansicht sollte folgendermassen durchgeführt werden:

```
this.model.func()
```

4.1 Widget-Konfiguration

Alle Widget-Einstellungen, die zur Verknüpfung mit fellerLYnk-Objekten erforderlich sind, sowie andere Einstellungen und Daten, die für die Formular-Erstellung und Bearbeitung im Touch-Constructor benötigt werden, müssen in der Eigenschaften-*config* Ihres Modells beschrieben werden.

```
Example:
Widget["your widget ID"] = Widget.extend({
    config : {
        title : "Name of widget",
        fields : {
            title : {
                datatype : "string",
                title : "Title"
            }
        },
        objects : {
            key : {
                datatype : dt.bit,
                title : "Object"
            }
            ...
        },
        settings : {
            "min-value" : {
                datatype : "number",
                title : "Minimum value"
            }
            ...
        },
        systems:["blinds", "ac"]
    },
    ...
}
```

In dem Feld „Titel“ wird der Name des Widgets angegeben. Es wird in einem Vorschaumodus und Bearbeitungsmodus angezeigt.

Im Feld „Felder“ werden die Felder des Widgets eingetragen, die vom Benutzer im Constructor konfiguriert werden sollen. Mit Hilfe von Objekt-Schlüsseln ist ein Zugriff auf die Werte dieser Eigenschaften möglich. Der Zugriff auf Felder innerhalb des Modells selbst lässt sich mit dem folgenden Befehl *this.field* durchführen. Wir empfehlen, dieses Feld nur zum Bearbeiten des Eigenschaftstitels und aller anderen Widget-Einstellungen zu verwenden, um den Bereich *Einstellungen* aufzurufen.

Das Feld „Einstellungen“ enthält eine Liste von Widget-Einstellungen, die für die Benutzerbearbeitung im Constructor-Modus zur Verfügung stehen. Objektschlüssel werden für den Zugriff auf die Einstellungen verwendet, und zwar mittels der unten beschriebenen Helfer. Für jede Eigenschaft müssen Sie einen Titel angeben, der im Widget-Bearbeitungsmodus angezeigt wird, sowie *datatype*. Derzeit werden zwei Datentypen unterstützt - *Zeichenfolge* und *Zahl*.

In dem Feld „Objekte“ wird eine Liste von Objekten von fellerLYnk (Gruppenadressen) angezeigt, die mit einem bestimmten Widget verknüpft werden können. Bestimmte Objekte sollten vom Benutzer über das Widget-Erstellungsformular im Touch-Constructor angegeben werden. Objektschlüssel werden für den Zugriff auf die

Objektwerte und Eigenschaften verwendet, und zwar mittels der unten beschriebenen Helfer. *Titel* sollte auch für jedes Objekt angegeben werden, das im Widget-Bearbeitungsformular angezeigt wird sowie die KNX-Datentypnummer. Hier finden Sie eine Liste aller unterstützten Datentypen http://openrb.com/docs/lua.htm#grp-info. Um bestimmte numerische Konstanten bereitstellen zu können, empfehlen wir, das Objekt *dt* mit einer Liste von Feldern zu verwenden, die im obigen Link festgelegt sind. Wenn ein bestimmter KNX-Subtyp als Datentyp angegeben ist, hat der Benutzer die Möglichkeit, Objekte nur mit diesem Typ auszuwählen. Wenn ein bestimmter Typ als Datentyp angegeben ist, hat der Benutzer die Möglichkeit, Objekte nur mit diesem Datentyp oder dessen Subtypen auszuwählen.

Mit dem Feld „systems“ wird eine Liste von Engineering-Systemen angegeben, die in Touch-Navigation verwendet werden. Widgets eines bestimmten Typs werden an einem Bildschirm aller Engineering-Systeme angezeigt, die in einem Array aufgelistet sind. Nachfolgend eine Liste von Kennungen von Engineering-Systemen, die verwendet werden können:

- Beleuchtung
- Jalousie
- Stromversorgung
- Fenster
- AC
- Belüftung
- Fussboden

4.2 Hilfsmethoden der Widget-Klasse (Basis-Modell)

Nachfolgend eine Liste der vordefinierten Widget-Methoden, die Sie in Ihren Modellen und Ansichten verwenden können.

4.2.1 Das Arbeiten mit fellerLYnk-Objekten (Gruppenadressen):

getValue(key)

Gibt den aktuellen Objektwert zurück.

existsValue(key)

Gibt <*true*> zurück, wenn der Benutzer ein spezifisches Objekt im Touch-Konfigurator angegeben hat oder <*false*>, wenn das Objekt jetzt angegeben ist.

getObject(key)

Gibt die Information über das Objekt zurück - Gruppenadresse, Datentyp und Objektname. Antwort-Beispiel:

```
{
    updatetime:1461825143,
    address : "1/1/1",
    units "",
    value : true,
    name : "alarm",
    datatype: 1005
}
```

write(key, value)

Objektwert an Controller/Feldbus schreiben.

4.2.2 Andere Methoden und Eigenschaften

conf(key)

Gibt den Einstellwert (beschrieben im Abschnitt *Einstellungen* der Widget-Konfiguration) zurück, der von dem Benutzer angegeben wurde.

title()

Gibt den Wert des Felds „Titel“ zurück, das von dem Benutzer beim Einrichten des Widgets im Touch-Constructor festgelegt wurde.

ctitle()

Gibt den Widget-Namen zurück, der im Abschnitt „Titel“ der Widget-Konfiguration angegeben ist.

isPreview

Das Feld hat den Wert <*true*>, wenn das Widget im Vorschau-Modus angezeigt wird. Achtung: Das Widget wird vor den Benutzereinstellungen in einem Vorschau-Modus angezeigt, d.h. es sind keine Gruppenadressen oder andere Einstellungen eingerichtet.

4.2.3 Aufruf von Methoden der Widget-Klasse

Um auf die obigen Hilfsfunktionen zuzugreifen, den richtigen Kontext verwenden. Beispiel für den Aufruf einer Funktion title():

- bei Modell-Methoden *this.title()*
- bei Ansicht-Methoden *this.model.title()*
- bei Vorlagen *title()*

4.3 Hilfsmethoden der Klasse WidgetView (Grundansichten)

Liste der vordefinierten Methoden von WidgetView, die bei Ihren Ansichten verwendet werden können.

size(width, height)

Ermöglicht eine Grössenänderung des Widgets nach der Initialisierung (im laufenden Betrieb). Die Widget-Grösse sollte in der Anzahl der belegten Zellen angegeben werden. Die Grösse einer Zelle beträgt 110px*110px, aber unter Berücksichtigung der Felder lässt sich die tatsächliche Grösse mit der folgenden Formel berechnen (110*Breite-10) px * (110*Höhe-10) px

el

DOM-Modell eines HTML-Containers, in dem sich Ihr Widget befindet. Eine ausführlichere Beschreibung der Verwendung befindet sich in der Dokumentation der Bibliothek Backbone.js http://backbonejs.org/#View-el. Zur Vermeidung von Kollisionen mit anderen Widgets empfehlen wir Ihnen, es überall dort zu verwenden, wo Sie für einen Zugriff auf das DOM-Modell Ihres Widgets Selektoren benutzen. Beispiel für die Suche auf Ihrer Widget-Titelseite:

```
$(“div.widget-title”, this.el)
```

Methoden der WidgetView-Klasse können nur aus Ihrer Ansicht aufgerufen werden. Der Zugang ist mit *this* möglich. (Zum Beispiel, this.size(1,2))

4.4 Auslöse-Methoden

Bei Teilen von Methoden, die in der Struktur der JS-Datei beschrieben sind, handelt es sich um Ereignisse. Sie werden vom Programm-Kernel in bestimmten Fällen aufgerufen.

changeValue(key, value) für das Modell

Wenn Sie diese Methode in Ihrem Modell implementiert haben, wird sie jedes Mal aufgerufen, wenn in fellerLYnk ein Objektwert geändert wurde (Gruppenadressen, die sich auf Ihr Widget beziehen). Der Objektschlüssel (wie in der Konfiguration angegeben) sowie dessen neuer Wert werden als Parameter für die Methode angegeben.

init() für die Ansicht

Wenn Sie diese Methode in Ihrer Ansicht implementiert haben, wird sie während der Initialisierung der Ansicht (vor dem Rendern) einmal aufgerufen. Hier können Sie Startparameter der Ansicht initialisieren, die beim Rendern in der Vorlage verwendet werden können.

afterRender() für die Ansicht

Wenn Sie diese Methode in Ihrer Ansicht implementiert haben, wird sie sofort nach dem Rendern des Widgets einmal aufgerufen. Sie eignet sich zur Initialisierung einzelner Steuerelemente, die nicht in der Vorlage vorgenommen werden können (z.B. Dimmerknopf)

4.5 Widget-Vorlagen

In der Widget-Ansichtklasse müssen Sie zwei Methoden umsetzen:

template(model)

Es wird vom System während der Widget-Initialisierung aufgerufen (um das Widget in einer allgemeinen Widget-Liste zu rendern). Als Eingabeparameter wird das initialisierte Modell gesendet.

preview(model)

Es wird vom System während der Widget-Initialisierung für sein Vorschau-Rendering im Touch-Constructor aufgerufen. Bitte beachten, dass das Modell in diesem Fall keine Objekte und Widget-Einstellungen enthält, die vom Benutzer während des Widget-Erstellungsvorgangs festgelegt werden.

Für das Templating empfehlen wir dringend, die Methode _.template aus der Underscore.js-Bibliothek zu verwenden: http://underscorejs.org/#template. Alternativ dazu können auch jedes andere Templating verwenden oder Ihre Widget-DOM-Objekte in JavaScript erstellen. In diesem Fall nicht vergessen, dass sich das gesamte Widget in einem Container befindet, auf den Sie in Ihrer Ansicht von this.el aus zugreifen können.

Alle weiteren Beispiele in dieser Dokumentation nutzen die Underscore.js-Bibliothek, die bereits im Projekt enthalten ist.

4.5.1 Empfohlene Widget-Struktur

Wir empfehlen die Verwendung der folgenden HTML-Struktur, damit alle Basisklassen von System-Widgets angewendet werden können:

```
WidgetView["your widget ID"]  = WidgetView.extend({
    ...
    template: _.template(" \
        <div class=\"widget\"> \
            <div class=\"widget-title\"> \
                <div class=\"txt\"><%=title() || ctitle()%></div> \
            </div> \
            widget controls
        </div> \
    "),
    ...
})
```

Sie können auch eigene Klassen hinzufügen und hierfür Stile in Ihrer style.css festlegen oder eine andere Widget-Struktur verwenden. Denken Sie daran, dass alle erstellten CSS-Klassen für Ihr Widget die Darstellung anderer Elemente der Anwendung beeinflussen können. Zur Vermeidung solcher Kollisionen empfehlen wir die Verwendung von eindeutigen Namen für neue CSS-Klassen.

4.5.2 Verwendung von Methoden und Eigenschaften des Modells in Vorlagen

Wenn Sie _.template Underscore.js zum Templating von Widgets verwenden, können Sie, da ein Widget-Modell als Eingabeparameter in Ihrer Vorlage übergeben wird, ohne Angabe des Kontextes alle Modellmethoden und deren Eigenschaften direkt in der Vorlage aufrufen.

Beispiel:

```
WidgetView["your widget ID"]  = WidgetView.extend({
    ...
    template: _.template(" \
        <div class=\"widget\"> \
            <div class=\"widget-title\"> \
                <div class=\"txt\"><%=title() || ctitle()%></div> \
            </div> \
                 <% if (existsValue('object')) { %> \
                 Object value <%= getValue('object')%> \
                 <% } %> \
        </div> \
    "),
    ...
})
```

4.5.3 Vorschauvorlage

Sie können eine separate Methode implementieren, um das Widget im Vorschau-Modus im Touch-Constructor anzuzeigen:

```
 WidgetView["your widget ID"]  = WidgetView.extend({
    ...
    template: _.template(" \
        <div class=\"widget\"> \
            <% if (existsValue(\"object\") { %> \
                <%=getValue(\"object\")%> \
            <%  } %> \
        </div> \
    "),
    preview: _.template(" \
        <div class=\"widget\"> \
            Object value \
        </div> \
    ")
    ...
})
```

Wenn die *Vorschau*-Methode nicht implementiert wird, wird während der Initialisierungsphase des Widgets die Methoden-*Vorlage* aufgerufen, um eine Vorschau zu erstellen. Somit kann eine einzelne Vorlage für ein Widget implementiert werden. Hierbei daran denken, dass es im Vorschaumodus keine Objekte und Einstellungen enthält, daher sollten alle notwendigen Überprüfungen in der Vorlage vorgenommen werden.

Das vorherige Beispiel, jedoch unter Verwendung einer einzigen Vorlage für beide Widget-Anzeige-Arten:

```
WidgetView["your widget ID"]  = WidgetView.extend({
    ...
    template: _.template(" \
        <div class=\"widget\"> \
            <% if (isPreview) { %> \
                Object value \
            <% } else if (existsValue(\"object\")) { %> \
                <%=getValue(\"object\")%> \
            <%  } %> \
        </div> \
    "),
    ...
})
```

4.6 Datenaustausch zwischen Modell und Ansicht

Jede Eigenschaft des Modells ist in der Ansicht verfügbar, da die Objektinstanz ihr Attribut ist, aber nicht umgekehrt. Ausserdem tritt häufig eine Situation auf, wenn die Attributsänderung des Modells die Ansicht des Widgets beeinflussen sollte. In diesem Fall empfehlen wir die Verwendung von Backbone-Ereignissen http://backbonejs.org/#Events.

Die folgenden Beispiele zeigen, wie der geänderte Objektwert auf der Seite angezeigt wird:

```
Widget["your widget ID"] = Widget.extend({
    ...
    changeValue : function(name, value) {
        //initiate event
        if (name=="status") return this.trigger("changedStatus");
        ...
    }
    ...
}

WidgetView["your widget ID"]  = WidgetView.extend({
    ...
    init : function() {
        //subscribe to a specific event model
        this.listenTo(this.model, "changedStatus", this.changedStatus);
    },
    template: _.template(" \
        <div class=\"widget\"> \
            <%=getValue(\"status\")%> \
        </div> \
    "),
    changedStatus : function() {
        //change the value in HTML code
        $(".widget", this.el).text(this.model.getValue("status"))
    },
    ...
})
```

Feller AG | Postfach | CH-8810 Horgen
Telefon +41 44 728 72 72 | Telefax +41 44 728 72 99

Feller SA | Caudray 6 | CH-1020 Renens
Telefon +41 21 653 24 45 | Telefax +41 21 653 24 51

Service-Hotline | Telefon +41 44 728 74 74 | info@feller.ch | www.feller.ch

by Schneider Electric