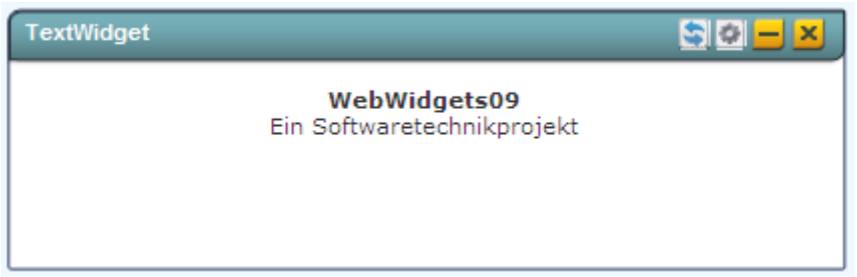
WebWidgets
Handbuch
für Widgetentwickler



WebWidgets ist ein **Softwaretechnikprojekt**
an der Universität Leipzig
in Zusammenarbeit mit der
BI Business Intelligence GmbH Leipzig

UNIVERSITÄT LEIPZIG

BI Business Intelligence
Interactive Decision Support Systems

Überblick

1. WebWidgets

Das Framework „WebWidgets“ stellt eine Umgebung zur Verfügung, mit deren Hilfe sie einfach Widgets für Startseiten erstellen können. Die Erstellung von Widgets für WebWidgets setzt Kenntnisse von Java voraus. Des weiteren sollten Sie sich mit dem smartGWT-Framework vertraut machen. Die Verwendung von Eclipse wird empfohlen.

Abbildung 1 zeigt ein kleines Beispielwidget in verschiedenen Zuständen. Ein Widget hat immer eine Titelleiste mit den vier Buttons: Aktualisieren, Konfigurieren, Maximieren/Minimieren und Schließen.
Darunter befindet sich der Anzeigebereich. Wird der Konfigurationsbutton gedrückt, wird der Anzeigebereich geschlossen und der ConfigBereich geöffnet.

Abbildung 1: Widgetzustände

Um ein neues Widget zu implementieren, erstellen Sie zunächst ein neues Package mit dem Namen Ihres Widgets („mywidget“). Erstellen Sie einen Unterordner „client“. In diesem erstellen Sie alle für das Widget benötigten Klassen. Ein Widget besteht mindestens aus drei Klassen:

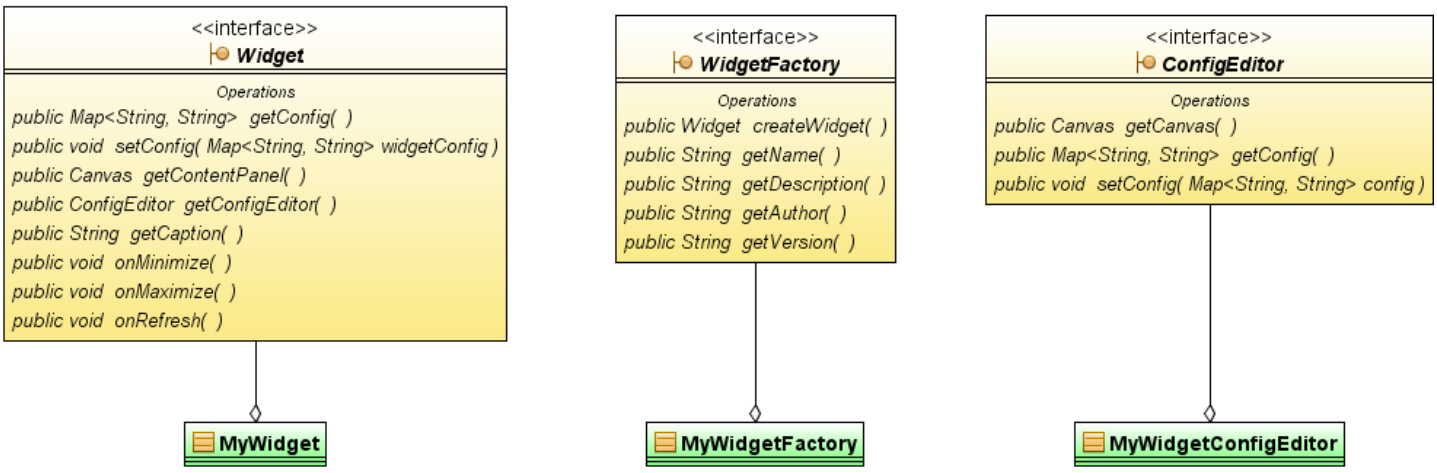


Abbildung 2: UML-Diagramm WebWidgetsFramework-Anbindung

Im folgenden soll die Implementierung eines Widgets erklärt werden. Der Beispielcode erstellt ein einfaches Textwidget, welches einen spezifizierten Text darstellt.

2. Die WidgetFactoryKlasse

Ihre Klasse „MyWidgetFactory“ muss das Interface „WidgetFactory[1]“ implementieren. Diese Klasse wird von der WebWidgetsOberfläche verwendet, um ihr Widget im Widget-Auswahldialog zu integrieren und dessen Erstellung zu initiieren.

```
/**
 * WidgetFactory für MyWidget
 *
 */
public class MyWidgetFactory implements WidgetFactory {

    private static final long serialVersionUID = 1L;

    @Override
    public Widget createWidget() {
      return new MyWidget();

    @Override
    public String getName() {
      return "MyWidget";

    @Override
    public String getDescription() {
      return "desription";
    }

    @Override
    public String getAuthor() {
      return "author";
    }

    @Override
    public String getVersion() {
      return "version";
    }
}
```

3. Die WidgetKlasse

Die WidgetKlasse ist für die Konfiguration und Darstellung des Widgets zuständig. Ihre erstellte Klasse „MyWidget“ muss das Interface „Widget[2]“ implementieren und sollte sich ebenfalls im Ordner mywidget.client befinden. Im Widget definieren Sie dessen Oberfläche und Variablen welche für die Konfiguration notwendig sind. Des weiteren wird hier der ConfigEditor aufgerufen. Im folgenden werden kurz die Methoden der Klasse vorgestellt.

1webwidgets.widget.client.WidgetFactory
2webwidgets.widget.client.Widget

Definieren Sie die Klasse MyWidget:

```
public class TextWidget implements
Widget {
      //Oberflaeche
      private Canvas panel = null;
      //Textfeld
      private Label label = null;
      //Text im Feld
      private String text = null;
}
```

Implementieren Sie alle der folgenden Methoden:
Im Konstruktor (oder einer von ihm aufgerufenen Methode) sollte die Oberfläche ihres Widgets erstellt werden. Verwenden Sie dazu Objekte von Typ Canvas[3].

```
PublicMyWidget(){
      panel = new Hlayout();
      label = new Label();
      label.setWidth100();
      label.setAlign(VerticalAlignment.CENTER);
      panel.setWidth100();
      panel.setHeight(100);
      panel.addMember(label);
}
```

Füllen Sie es mit allen für die Anzeige des Widgets benötigten Objekten. Es wird empfohlen, die Höhe des Widgets explizit festzulegen. Achten Sie darauf, den Konstruktor schlank zu halten, da er erst komplett ausgeführt wird, bevor das Widget angezeigt wird.

Geben Sie in der Methode „getContentPanel“ die erstellte Oberfläche zurück. Diese Methode wird zur Darstellung des Widgets aufgerufen.

```
@Override
public Canvas getContentPanel() {
  return panel;
}
```

Die Methode „onRefresh“ wird jedes mal aufgerufen, wenn der „Aktualisieren“-Button in der Titelleiste des Widgets angeklickt wird. Beachten Sie, das diese Methode beim Erstellen des Widgets nicht automatisch aufgerufen wird. Sollte ihr Widget Daten in einem Callback empfangen, sollte die Methode onRefresh“ danach manuell aufrufen werden.

```
@Override
    public void onRefresh() {
      panel.draw();
    }
```

3 http://www.smartclient.com/smartgwt/javadoc/com/smartgwt/client/widgets/Canvas.html

Die Methode „onMinimize“ wird beim Minimieren des Widgets aufgerufen. Implementieren Sie hier Funktionalität um das automatische Aktualisieren ihres Widgets (bspw. durch Timer) zu deaktivieren.

```
@Override
public void onMinimize() {
}
```

Die Methode „onMaximize“ wird beim Maximieren des Widgets aufgerufen. Implementieren Sie hier Funktionalität um das automatische Aktualisieren ihres Widgets (bspw. durch Timer) zu (re-)aktivieren.

```
@Override
public void onMaximize() {
}
```

Die Methode „getCaption“ wird bei der Erstellung des Widgets aufgerufen. Der Rückgabewert ist der anzeigte Titel des Widgets.

```
@Override
   public String getCaption() {
     return "MyWidget";
   }
```

Die folgenden Methoden dienen der Kommunikation des Widgets mit seinem ConfigEditor. Die Methode „getConfigEditor“ muss ein Objekt vom Typ ConfigEditor zurückgeben.

```
  @Override
  public ConfigEditor getConfigEditor() {
    return new MyConfigEditor;
  }
```

Die Methoden getConfig und setConfig werden nach dem Start und vor dem Beenden des ConfigEditors aufgerufen.
„getConfig“ muss dem ConfigEditor die aktuelle Konfiguation des Widgets in einer Map übergeben.

```
 @Override
   public Map<String, String> getConfig() {
     HashMap<String, String> widgetConfig = new HashMap<String, String>();
     widgetConfig.put("text", text);
     return widgetConfig;
   }
```

„setConfig“ übergibt dem Widget die neue Konfiguration, wenn der configEditor geschlossen wird. Diese sollten Sie in Ihrem Widget speichern, um Sie für den ConfigEditor zur Verfügung zu haben.

```
@Override
public void setConfig(Map<String, String> widgetConfig) {
  text = widgetConfig.get(„text“);
}
```

4. Die ConfigEditorKlasse

Der zu ihrem Widget gehörende ConfigEditor muss das Interface ConfigEditor[4] implementieren und wird aufgerufen, wenn der Konfigurationsbutton angeklickt wird. Implementieren Sie im Konstruktor (oder einer von ihm aufgerufenen Methode) die Oberfläche des ConfigEditors. Hierzu eignet sich eine DynamicForm, Sie können jedoch jedes Canvas-Objektverwenden.

```
public  class SimpleConfigEditor implements
ConfigEditor {

      DynamicForm dynamicForm;
      TextItem text;

    public  SimpleConfigEditor() {
      dynamicForm = new DynamicForm();
      text = new TextItem();
      dynamicForm.setWidth100();
      dynamicForm.setHeight100();
      dynamicForm .setFields("text");
    }
```

Die Methode „getCanvas“ gibt dem Widget dann das erstellte Canvas zurück.

```
@Override
public  Canvas getCanvas() {
  return dynamicForm;
}
```

Die Methode „setConfig“ holt die aktuelle Konfiguration des Widgets um sie im Konfigurationsbereich anzuzeigen. Dazu wird im Widget die Methode „getConfig“ aufgerufen.

```
@Override
  public  void setConfig(Map<String, String> config) {
    text.setContents(config.get("text"));
  }
```

Die Methode „getConfig“ wird beim Klick auf den OK-Button im Konfigurationbereich aufgerufen und übergibt dem Widget in seiner „setConfig“ Methode die neue Konfiguration als Map.

```
@Override
   public  Map<String, String> getConfig() {
     Map<String, String>configValues = new HashMap<String, String>();
     configValues.put("text", text.getDisplayValue());
     return configValues;
   }
```

Beachten Sie: Wird der Configeditor aufgerufen, wird zuerst der Konstruktor ausgeführt und danach die Methode setConfig. Legen Sie besonderes Augenmerk auf NullPointerExeptions.

Damit haben Sie ihr erstes Widget erstellt.

4webwidgets.widget.client.ConfigEditor

5. Integration des Widgets in einen Tomcat-Server

Nachdem Sie ihr Widget fertiggestellt haben, kontaktieren Sie ihren Serveradministrator und bitten Ihn, das Widget in die WidgetCollection einzufügen. Dazu benötigt er alle ihre Klassen in den richtigen Ordner, die Namen aller Services, die Sie verwenden, den Namen Ihren Hauptklasse (“MyWidget“) sowie die Module Beschreibung (“*.gwt.xml“). Stellen Sie des weiteren alle zusätzlich verwendeten Bibliotheken zur Verfügung, CSS Dateien etc.

6. Weitere Hinweise

GWT-RPC-Calls

WebWidgets verwendet unter anderem das GoogleWebToolKit GWT, smartClient und smartGWT.
Daraus resultiert, dass auf dem Clientrechner nur JavaScript-Code ausgeführt wird, obwohl ihr Widget in Java geschrieben ist.
Es ergeben sich allerdings auch einige Probleme, da der GWT Java-to-JavaScript Compiler nicht alle Packages aus Java sowie keine anderen zusätzlich verwendeten Frameworks übersetzen kann.
Um puren JavaCode ausführen zu können muss ein GWT-RPC Call verwendet werden. Dessen Verwendung ist ebenfalls bei allen Zugriffen auf Internetserver erforderlich.
Dazu legen Sie in Ihrem Paket „mywidget“ einen neuen Unterordner „server“ an. In diesem erstellen sie eine Klasse „MyServiceImpl“, im Ordner „client“ die Interfaces „MyService“ und „MyServiceAsync“.
Alle weiteren Informationen über RPC-Calls finden Sie hier.

Lizenzierung

WebWidgets steht unter der „Apache License, Version 2.0“, alle anderen Frameworks unter ihren jeweiligen Lizenzen. Für Ihre selbst geschriebenen Widgets wird die Veröffentlichung unter der „Apache License, Version 2.0“ oder äquivalenter Lizenz empfohlen. Informationen dazu erhalten Sie unter http://www.apache.org und http://www.gnu.org

Bildnachweise

Logo Universität Leipzig: www.uni-leipzig.de

Logo BI Business Intelligence GmbH: www.bi-web.de

Alle anderen Abbildungen: Projektgruppe WebWidgets