30 Minuten von der Idee bis zum ersten selbst erstellten APEX-Plug-in

Andreas Wismann
WHEN OTHERS
D-41564 Kaarst

Schlüsselworte
APEX, Oracle Application Express, Plug-in, APEX-Plug-ins, Plugin

Einleitung
Haben Sie jemals SQL, PL/SQL, HTML, CSS, JavaScript oder sonstigen Programmcode in Ihrer APEX-Anwendung „vergraben“ und hatten Sie anschließend Mühe, den Überblick über die Fundstellen zu bewahren?
Müssen Sie als Entwickler in APEX häufig gleichartige Objekte erstellen und dabei immer dieselben, zeitraubenden Anpassungen vornehmen?
Möchten Sie Ihre selbst entwickelten Komponenten für alle APEX-Anwendungen im Unternehmen bequem und zuverlässig verfügbar machen?
Dann sollten Sie unbedingt APEX Plugins in die engere Wahl nehmen.
Seit der APEX-Version 4.0 besteht die Möglichkeit, Objektdeklarationen und Programmcode als Plug-in zu bündeln und „auszurollen“. Es gibt Plug-ins für Items, Regionen, Dynamic Actions, Seitenprozesse und seit APEX 4.1 auch für Authentifizierungs- und Autorisierungsschemata. Außer „selbst erstellen“ geht auch „herunterladen“: Plug-in-Autoren können ihre Entwicklungen der Allgemeinheit zur Verfügung stellen, da Form und Verfahren standardisiert sind.
Plug-ins erweitern nicht wirklich die Funktionalität von APEX-Anwendungen (da alles, was in einem Plug-in passiert, auch mit herkömmlichen Mitteln programmiert werden kann), aber sie vereinfachen den Umgang mit den beteiligten Code- und Objektartefakten erheblich.

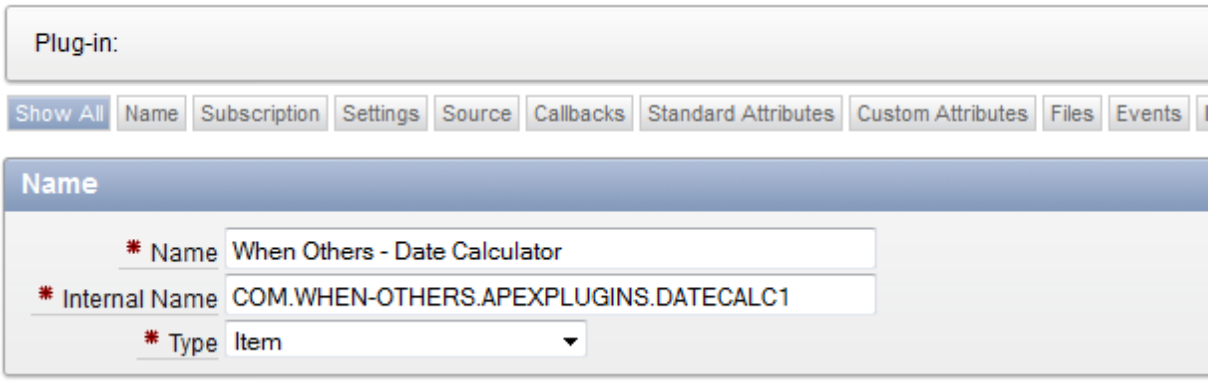
> Weil die APEX-Übersetzung ins Deutsche im Bereich der Plug-ins etwas holprig ausgefallen ist (Beispiel: „Render Function Name“ wurde übersetzt mit „Funktionsname wiedergeben“) und zwecks besserer Nachvollziehbarkeit in der englischsprachigen APEX-Dokumentation werde ich die Vorgehensweise mit der englischen APEX-Version demonstrieren. Außerdem verwende ich in diesem Manuskript ab hier die Schreibweise *Plugin* anstatt *Plug-in*. Man findet im Netz auch die getrennte Schreibweise *Plug In*. Ich finde, über-flüssige Leer Zeichen und Binde-Striche stören den Lese Fluss.

Das Konzept der Plugins
Für APEX-Plugins verwenden Sie typischerweise deklarative Elemente und PL/SQL-Code, wobei APEX Sie durch Assistenten und ausführliche Hilfedialoge hervorragend unterstützt. Den Programmcode können Sie entweder direkt im Plugin unterbringen (was sicherer ist, wenn Sie das Plugin verteilen möchten) oder Sie können auf Datenbankfunktionen, Prozeduren und Packages verweisen (was hauptsächlich innerhalb eines Unternehmens gelingen dürfte, dann aber eine vorteilhafte Alternative darstellt). Die Standardisierung von Plugins wird sichergestellt durch Bibliotheken namens APEX_PLUGIN und APEX_PLUGIN_UTIL (siehe API-Dokumentation), wo vor allem Typen, Funktionssignaturen und Hilfsroutinen definiert werden.

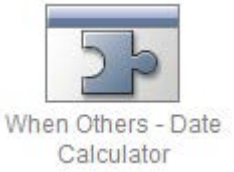
Die Idee
Angenommen, Sie wünschen sich ein Datums-Eingabefeld, das einerseits die übliche Ausstattung besitzt (Popup-Kalender beim Anklicken des Kalender-Icons), darüber hinaus aber auch in gewisser

Weise „rechnen“ kann. Sie möchten beispielsweise, dass bei der Eingabe „+10“ das Datum „heute in 10 Tagen“ ausgerechnet und übernommen wird. Wenn Sie für eine Bank programmieren, sind Sie vielleicht an Datumsberechnungen unter Berücksichtigung der Bankarbeitstage interessiert. Wie auch immer, dies scheint keine unlösbare Aufgabe zu sein – noch dazu, als es in diesem Manuskript nicht so sehr um die tollen Features Ihres Plugins geht, sondern um die Erstellung eines solchen. Fürs erste reicht also eine recht simple, vorläufige Implementierung, es kommt auf das Prinzip an.
Übrigens: Sie werden für eine gelungene Umsetzung dieser Idee neben PL/SQL auch JavaScript-Code benötigen. Dies führt zu einem „runden“ Beispielprojekt im Sinne dieses Manuskripts, aber JavaScript-Programmierung ist nicht zwingend erforderlich zur Erstellung von Plugins. Was Sie mindestens benötigen, sind PL/SQL-Kenntnisse und ein gutes Verständnis von APEX. Auch HTML sollte für Sie kein Buch mit sieben Siegeln sein, zumindest wenn es um die Programmierung von Item- oder Region-Plugins geht.

Die Geburtsstunde
Zunächst erzeugen Sie ein mehr oder weniger funktionsloses Plugin als Ausgangspunkt. Dazu gehen Sie auf „Shared Components > User Interface > Plugins > Create. Zunächst müssen Sie lediglich drei Pflichtfelder ausfüllen, um Ihr Plugin zu deklarieren:

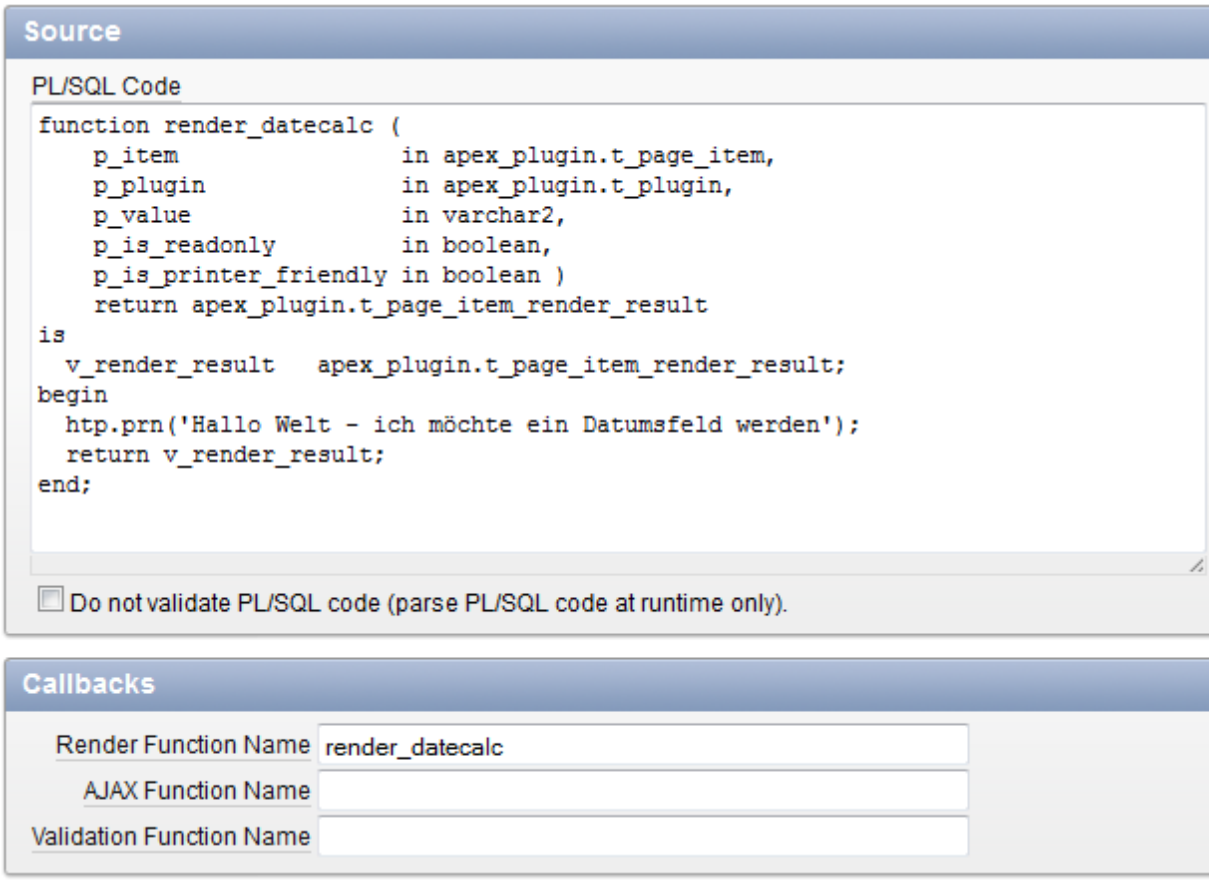


Der *Name* ist später für den Entwickler sichtbar, wann immer er die Plugin-Auswahlliste angezeigt bekommt. Sie sind frei in der Namenswahl. Gerne wird der Firmenname vorangestellt, um in Listenansichten mehrere Plugins vom selben Autor auf einen Blick zu erkennen, aber das ist Geschmackssache. Der *Internal Name* dient der Identifizierung des Plugins innerhalb von APEX und taucht später nicht mehr sichtbar auf.
Weil der *Internal Name* innerhalb Ihrer Anwendung(en) eindeutig sein muss (und sicherheitshalber sogar weltweit einzigartig, um jegliche Konflikte mit anderen Plugins zu vermeiden), bedient man sich der in Entwicklerkreisen weit verbreiteten Reverse-DNS-Notation. Namensgeber ist dabei eine Internet-Domain, deren Eigentümer Sie sind, wie etwa die Internetadresse Ihres Unternehmens. Deren Namensbestandteile werden in Punktnotation „von rechts nach links“ geschrieben, also beginnend mit der Top Level Domain. So haben Sie einen garantiert einmaligen Namensraum definiert und können weitere Ordnungskriterien und letztlich den Bezeichner Ihres Plugins anhängen. Der *Internal Name* darf maximal 40 Zeichen lang sein.
Als Type wählen Sie „Item“, weil Sie das Plugin später als Eingabefeld verwenden wollen. Nach einem Klick auf „*Apply Changes*“ beginnt Ihr Plugin schon zu leben, kann aber noch überhaupt nichts. Es wäre als Item in einer APEX-Seite zwar latent „vorhanden“, aber weder zu sehen noch benutzbar. Dies wird nun, Schritt für Schritt, geändert.



Klicken Sie das Icon Ihres Plugins erneut an, um dessen Eigenschaften zu

bearbeiten. Als erstes schreiben Sie eine rudimentäre *Render Function*, so dass Ihr Plugin in der Lage ist, sich selbst darzustellen. Dazu müssen Sie unter der Region *Callbacks* den (frei bestimmbaren) Namen derjenigen PL/SQL-Funktion angeben, die für das Rendern des Plugins zuständig ist, also für seine visuelle Darstellung. Durch einen Klick auf das Label *Render Function Name* erhalten Sie einen umfangreichen Hilfe-Dialog, aus dem Sie die den Beispielcode einfach kopieren können. Exakt diese vorgegebene Signatur muss von jeder *Render Function* implementiert werden, und sie muss eine Variable vom Typ `apex_plugin.t_page_item_render_result` zurückgeben.
Nachdem Sie den Funktionsrumpf aus dem Hilfe-Dialog in das Feld *Source* übertragen haben, erweitern Sie ihn – wie im folgenden Screenshot gezeigt – derart, dass Ihr Plugin zumindest ein erstes Lebenszeichen von sich gibt. Alles, was in der *Render Function* mit dem Befehl `htp.prn` (oder `htp.p`) ausgegeben wird, wird vom Browser als HTML-Code Ihres Plugins interpretiert.



Es lebt!
Das genügt bereits, um das Item-Plugin in einem Formular sehen zu können. Probieren Sie es aus und erzeugen Sie ein neues Item, indem Sie als Item-Typ „Plug-in“ anklicken und Ihr Baby auswählen – es wird irgendwo auf der APEX-Seite seinen „Hallo Welt“-Spruch aufsagen! Nun müssen Sie dem neugeborenen Datumsfeld zunächst die wesentlichen Fähigkeiten eines erwachsenen Datumsfeldes einpflanzen, bevor es zuletzt um die Realisierung der Sonderwünsche geht.

> *Um den Rahmen dieses Manuskripts nicht zu sprengen, werden beispielhaft einige wesentliche Eigenschaften des Datepickers implementiert; zur Vertiefung gebe ich am Ende Tipps und Literaturempfehlungen.*

Eine Methode, wie Sie das elegant hinbekommen: Sie platzieren im Formular einen „echten“ Datepicker direkt über Ihrem Plugin und vergleichen den HTML-Code des Plugins mit dem des Originals. Das könnte etwa so aussehen wie im folgenden Screenshot:

Ihr HTML-Code für beide Items wird dann ähnlich diesem Schnipsel aussehen, das ich zur besseren Lesbarkeit eingerückt formatiert habe. In der oberen Tabellenzeile das Original, darunter das noch unvollständige Plugin:

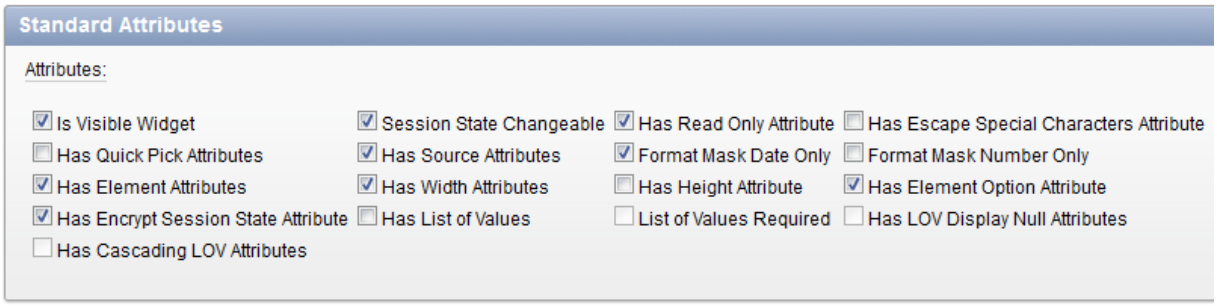
```
…
<tr>
  <td nowrap="nowrap" align="right">
        <label  for="P11111_DATE_PICKER"
                class="uOptional">Date Picker</label></td>
  <td colspan="1" rowspan="1" align="left" valign="middle">
        <input type="hidden"
                name="p_arg_names"
                value="4085007206184223" />
        <input type="text"
                class="datepicker"
                id="P11111_DATE_PICKER"
                name="p_t01"
                maxlength="4000" size="30" value=""
                autocomplete="off"></td></tr>

<tr>
  <td nowrap="nowrap" align="left"></td>
  <td colspan="1" rowspan="1" align="left">
        Hallo Welt - ich möchte ein Datumsfeld werden</td></tr>
…
```

Kleiner Tipp am Rande:

> Schalten Sie unter „Application Properties > Availability > Status“ den Wert auf „Available“ um (also ohne „Edit Links“). Der HTML-Quellcode ist dann viel einfacher zu analysieren.

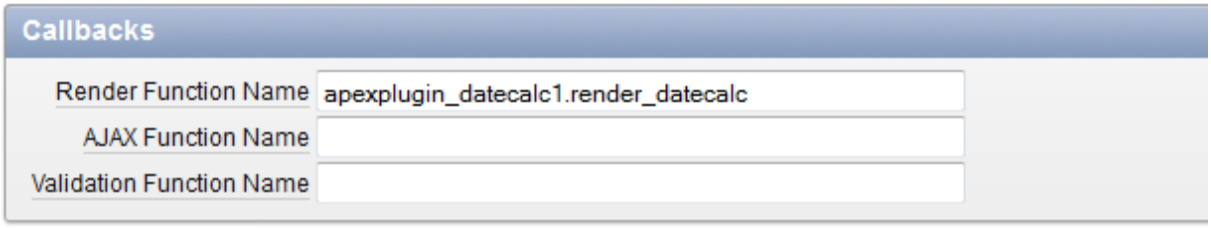
Die Tags `<tr>` und `<td>` erzeugt APEX immer selbst. Was uns eigentlich interessiert, sind die Elemente `<label>` und `<input>`. Wie Sie sehen, fehlen unserem Plugin offensichtlich sämtliche Standard-Attribute eines typischen Datepickers, das Kind kennt noch nicht einmal seinen eigenen Namen.
Wenn Sie sich daraufhin die Items in der APEX-Entwicklersicht anschauen und miteinander vergleichen, stellen Sie fest, dass das Plugin bei weitem nicht die Menge an Einstellungsmöglichkeiten bietet wie der originale Datepicker. Es fehlen beispielsweise die Einstellungen für *Label*, *Settings*, *Source* und *Default Value*. Lediglich die Bereiche *Name*, *Displayed*, *Conditions*, *Security*, *Configuration* und *Comments* sind vorhanden, zum Teil auch noch mit reduzierten Attributen.

Standardattribute auswählen
Im Einstellungsdialog Ihres Plugins („Shared Components > User Interface > Plug-ins“) werden Sie jetzt diejenigen Standard-Attribute hinzufügen, die ein waschechter Datepicker benötigt. Haken Sie dazu die Optionen, die im folgenden Screenshot ausgewählt sind, an – keine von Ihnen ist ursprünglich ausgewählt gewesen.



Schauen Sie im Edit-Dialog Ihres Plugin-Items nach, so entdecken Sie nun alle Einstellungsoptionen, die wir vorhin vermisst haben. Der nächste Schritt besteht darin, auf diese Einstellungen adäquat zu reagieren. Die APEX-Engine teilt der *Render Function* zwar ab sofort mit, was der Entwickler eingestellt hat (dafür zeichnen die IN-Parameter verantwortlich), aber nach wie vor spuckt unser Plugin partout nur „Hallo Welt“ aus, egal was bestellt wurde.

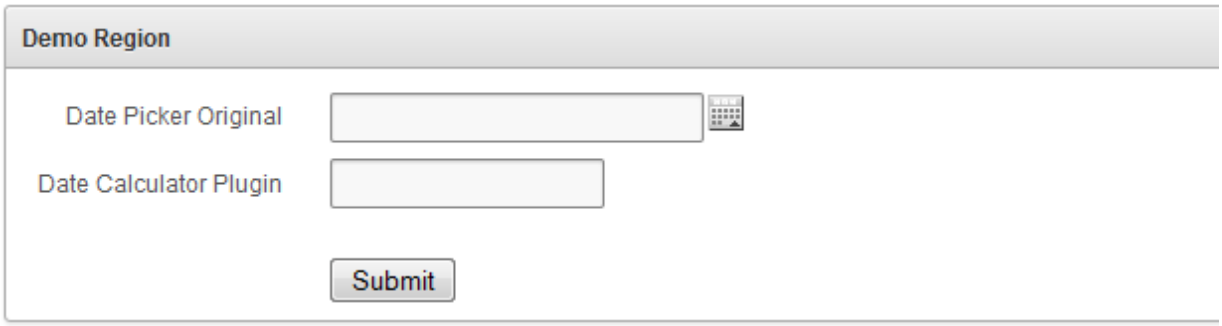
Render Function aufbohren
Um die nachfolgenden Schritte nicht in einem spartanischen APEX-Editorfenster durchführen zu müssen, empfehle ich Ihnen, ein Package für den PL/SQL-Code Ihres Plugins anzulegen. Ich verwende als Packagenamen die Bezeichnung des Plugins, also `CREATE OR REPLACE PACKAGE apexplugin_datecalc1`. In einem Package können Sie die weitere Entwicklungsarbeit mit einem vernünftigen PL/SQL-Editor durchführen. Schon die Tatsache, dass Sie standardmäßig nur elf Codezeilen gleichzeitig im *Source*-Eingabefenster von APEX sehen können, sollte dieser kleinen Mühe wert sein. Sie vergessen natürlich nicht, Ihrem Plugin die Änderung mitzuteilen, indem Sie bei *Render Function Name* den Packagenamen erwähnen.



Zunächst erstellen Sie das HTML-Gerüst für das Formulareingabefeld anhand der Erkenntnisse aus dem Codevergleich. Erfreulich: Für das Label (damit ist die links vom Eingabefeld angezeigte Beschriftung gemeint) brauchen Sie nichts zu unternehmen. Durch die Auswahl von *Is Visible Widget* als Standardattribut wird das Label automatisch verdrahtet, wenn der Entwickler eine Beschriftung angibt.
Das `input`-Tag lässt sich folgendermaßen modellieren (Auszug aus der *Render Function*):

```
BEGIN
  htp.prn('<input type="text" value="' || p_value || '"/>');
  RETURN v_render_result;
END;
```

Das sieht schonmal nicht schlecht aus:



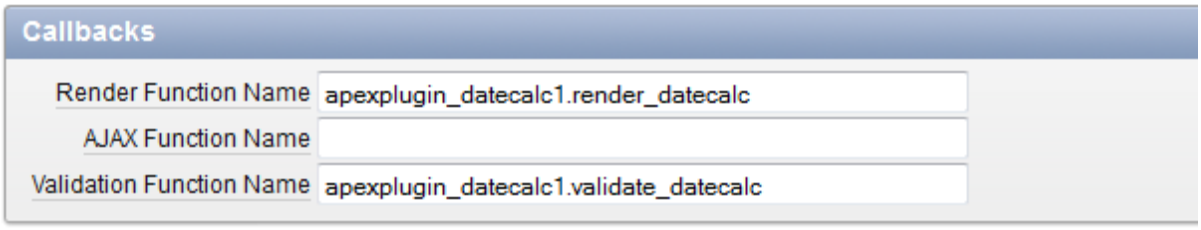
Wie Sie sehen, habe ich unterdessen auf der Seite eine Standard-Schaltfläche hinzugefügt und beide Label beschriftet. Beim Klicken auf *Submit* werden Sie feststellen, dass keine Ihrer Eingaben vom Plugin übernommen wird. Sie erhalten stets ein leeres Eingabefeld. Um das zu ändern, erweitern Sie die *Render Function* erneut:

```
FUNCTION render_datecalc
(
  p_item                IN apex_plugin.t_page_item
 ,p_plugin              IN apex_plugin.t_plugin
 ,p_value               IN VARCHAR2
 ,p_is_readonly         IN BOOLEAN
 ,p_is_printer_friendly IN BOOLEAN
) RETURN apex_plugin.t_page_item_render_result
IS
  v_name          VARCHAR2(100);
  v_render_result apex_plugin.t_page_item_render_result;
BEGIN

  -- siehe API-Doku: name-Attribut bei der APEX-Engine abholen
  v_name := apex_plugin.get_input_name_for_page_item(FALSE);

  -- HTML-Gerüst
  htp.prn('<input type="text"'
          || ' id="'    || p_item.name || '"'
          || ' name="'  || v_name      || '"'
          || ' value="' || p_value     || '"'
          || ' />');

  -- ebenfalls siehe Doku: Das Item muss begehbar sein
  v_render_result.is_navigable := TRUE;

  RETURN v_render_result;

END render_datecalc;
```

Der Aufruf von `apex_plugin.get_input_name_for_page_item` wird in der API-Dokumentation näher beschrieben: Ohne das Attribut `name="…"` ist ein HTML-Formularelement nämlich nicht in der Lage, den vom Benutzer eingegebenen Wert an den Server weiterzugeben. Ein Blick auf den nun entstandenen HTML-Code lässt Gutes ahnen (ich habe bereits einen Wert in das Plugin-Eingabefeld eingetippt und das Formular abgeschickt):

```
<input type="hidden" name="p_arg_names" value="4084723485798388" />
<input type="text" id="P11111_DATECALC" name="p_t02" value="01.02.1964" />
```

Auf jeden Fall benötigt das Plugin jetzt eine Validierung, denn zurzeit werden unsinnige Eingaben noch akzeptiert. Hierzu dient die Validation Function. Tragen Sie hier, analog zur Render Function, den Namen der von Ihnen zu erstellenden Prüffunktion ein und kopieren Sie die Funktionssignatur wie gehabt aus dem Hilfedialog in Ihr Plugin-Package. Anschließend schreiben Sie eine zunächst sehr einfache Validierung, um sicherzustellen, dass die Datumsprüfung funktioniert:




```
FUNCTION validate_datecalc
(
  p_item   IN apex_plugin.t_page_item
 ,p_plugin IN apex_plugin.t_plugin
 ,p_value  IN VARCHAR2
) RETURN apex_plugin.t_page_item_validation_result
IS
  v_validation_result apex_plugin.t_page_item_validation_result;
BEGIN
  --
  IF NOT wwv_flow_utilities.is_date(p_value, 'DD.MM.YYYY') THEN
    v_validation_result.message := 'Ungültiges Datum in '
                                   || p_item.plain_label;
  END IF;
  RETURN v_validation_result;

END validate_datecalc;
```

Geben Sie anschließend im Formular einen Wert ein, der nicht als gültiges Datum durchgeht, und submitten Sie die Seite:

Der Link „Gehen Sie zum Fehler“ funktioniert ebenfalls.
Auch wenn der bis hierhin geschriebene Code so noch nicht „in Produktion“ gehen kann (sondern zunächst das Prinzip demonstriert), zeigen die Beispiele, wie zügig man erste Ergebnisse mit APEX Plugins erreichen kann. Darauf aufbauend zeigt mein DOAG-Vortrag das weitere Vorgehen. Der Schlüssel zu den ausstehenden Funktionalitäten sind die *Custom Attributes* sowie die Einbindung des JavaScript-Codes.

Bestandsaufname: Was wurde schon erreicht, was fehlt noch?

- Wir haben ein APEX-Plugin erstellt, das kompiliert und gültigen HTML-Code ausgibt.
- Das Plugin rendert sich als Texteingabefeld und akzeptiert Datumseingaben.
- Das Texteingabefeld besitzt ein Label, das sich einstellen lässt und auf Änderungen des Label Templates reagiert.
- Die Benutzereingabe wird an die APEX-Engine weitergegeben und beim nächsten Aufruf wieder dargestellt.
- Das eingetippte Datum wird gegen eine Formatmaske validiert, die vorläufig hartcodiert ist.
- In den *Settings* fehlen dem Plugin einige Datepicker-Einstellungen, unter anderem die Anzeige des Kalender-Icons und die Einstellungen für *Minimum Date* und *Maximum Date*
- Auf Änderungen der *Source* (zum Beispiel *SQL Query Return Single Value*) reagiert das Plugin noch nicht.
- Die „Sonderfunktionen“, derentwegen wir das Plugin entwickeln, werden erst ganz zum Schluss hinzugefügt.

Projekte beginnen meistens mit vollständigen Spezifikationen, während die Umsetzung vieler Komponenten parallel anläuft. Gerade dann zeigen APEX-Plugins ideale Eigenschaften: Entwickler können Plugin-Instanzen frühzeitig in APEX-Formulare einbauen, selbst wenn deren Programmierung noch in den Anfängen steckt. Wird das Plugin dann fertig, besitzen alle Instanzen quasi „wie von selbst“ die gewünschte Funktionalität, ohne dass im Idealfall weitere Anpassungen der Items nötig wären.

Literaturempfehlung
Neben der APEX API Reference gibt es ein empfehlenswertes englischsprachiges Buch zur Plugin-Entwicklung von Martin Giffy D’Souza: „Expert Oracle Application Express Plugins (Building Reusable Components)“; Apress, 2011.

Ausblick auf den Vortrag
Am Mittwoch, dem 21.11.2012, um 13:00 Uhr im Stream „Development“ zeige ich, welche Schritte zur Fertigstellung des Plugins noch zu gehen sind und wie man das fertige Plugin ausrollt. Unmittelbar nach dem Vortrag können Sie die Präsentationsfolien, eine Beispielanwendung und natürlich auch das Plugin hier herunterladen:

http://when-others.com/download/doag/2012

Kontaktadresse:
Andreas Wismann
WHEN OTHERS
Hirschstraße 10
D-41564 Kaarst

Telefon: +49 (0) 2131-314 9966
E-Mail wismann@when-others.com
Internet: www.when-others.com