In diesem Beispiel wird die Entwicklung eines Apex-Region-Type-Plug-ins vorgestellt. Um eine inhaltlich möglichst umfangreiche Lösung abzubilden, kommt die jQuery-Image-Slider-Technik zum Einsatz.

Erstellung eines Apex Plug-ins

Tobias Arnhold, IT-Consulting

Warum benötigt man eigentlich Apex Plug-ins? Sie machen die Wiederverwendung von Web- und SQL-Techniken modularer einsetzbar. Das heißt, es muss kein PL/SQL-, kein JavaScript-Code und auch keine Datei kopiert beziehungsweise angepasst werden, wenn eine bestimmte Funktionalität wiederverwendet werden soll.

Wann lohnt sich der Aufbau eines Plug-ins? Er lohnt sich immer dann, wenn man Codes innerhalb einer Apex-Umgebung häufig wiederverwenden will. Auch wenn nachträglich Erweiterungen an dem Plug-in vorzunehmen sind, ist die Aktualisierung im Rahmen der Apex-Umgebung recht einfach realisierbar. Es gibt die folgenden Plug-in-Typen: Region-Type-Plug-ins, Item-Type-Plug-ins, Dynamic-Action-Plug-ins, Process-Type-Plug-ins, Authorization-Type-Plug-ins und Authentication-Type-Plug-ins.

Ein Apex Plug-in erstellen

Unter „Shared Components“ findet man den Punkt „User Interface -> Plug-ins“. Nach einem Klick auf „Plug-ins“ erscheint die Übersicht aller erstellten und installierten Plug-ins.

Abbildung 1: Übersicht der Plug-ins

Mit einem Klick auf „Create ->“ gelangt man zur Maske für die Erstellung eines Apex Plug-ins. Folgende Menüreiter stehen zur Verfügung:

- *Name*
  Beschreibung des Plug-in-Namens und -Typs
- *Subscription*
  Definition eines Master-Plug-ins, von dem aus zentral aktualisiert werden soll. Weitere Informatio-nen dazu unter http://tinyurl.com/6tfqohv
- *Settings*
  File-Prefix definiert, ob die verwendeten Dateien aus der Oracle-Datenbank oder dem Webserver geladen werden.
- *Source*
  Definition der PL/SQL-Plug-in-Logik. In diesem Bereich sind die PL/SQL-Funktionen angelegt, die für das Ausführen der Plug-in-Prozesse notwendig sind.
- *Callbacks*
  Funktionsnamen, die zum Aufruf des Plug-ins notwendig sind. Der „Render Function Name“ ist zwingend notwendig, da über diesen Funktionsnamen auf die Startfunktion im „Source“-Bereich zugegriffen wird.
- *Standard Attributes*
  Zusätzliche festdefinierte Attribut-Optionen zur Verwendung des Plug-ins. Die Attribute variieren zwischen Plug-in-Typen.
- *Custom Attributes*
  Benutzerdefinierte Attribute, die beim Anwenden des Plug-ins innerhalb der Apex-Anwendung gesetzt werden. Diese Attribute finden innerhalb des PL/SQL-Codes Verwendung.
- *Files*
  Upload-Bereich von JavaScript- und CSS-Files, die innerhalb des Plug-ins verwendet werden. Dateien sind in der Datenbank abgelegt.
- *Events*
  Erstellung von benutzerdefinierten Events
- *Information*
  Infos zur Version und der URL des Herstellers
- *Help Text*
  Hilfe-Text zum Plug-in
- *Comments*
  Entwicklerkommentare

Der PL/SQL-Code des Plug-ins

Neben den vielen Einstellungsmöglichkeiten kommt es vor allem auf den PL/SQL-Code des Plug-ins an. Dabei stellt sich die Frage: Was unterscheidet diesen Code von normalen PL/SQL-Funktionen?

Der Aufbau der Hauptfunktion ist mit festen Übergabe-Parametern vorgegeben und beinhaltet auch einen fest vordefinierten Rückgabewert. Dabei unterscheidet sich die Benennung der Parameter und des Typs je nach Plug-in-Art. Der Name der Funktion kann frei gewählt werden und ist später im Feld „Render Function Name“ zu hinterlegen.

Um ein besseres Verständnis vom Aufbau zu bekommen, wurde für dieses Beispiel eine Art leere Hauptfunktion erstellt. Diese generiert lediglich eine DIV-Box mit einem Beispieltext und dem Wert des einzigen Übergabeparameters (siehe Listing 1).

Innerhalb des Codes sind verschiedene Kommentare hinterlegt, die auf einzelne Teilbereiche aufmerksam machen sollen:

1. *Funktionsaufruf mit Definition der Parameter*
   Aufbau des oben erwähnten Funktionsaufrufs
2. *Plug-in-Attribute per Variable übergeben*
   Innerhalb des „Declare"-Bereichs sind die Parameter und die „Custom Attributes" zu definieren. Dabei wird der Typ des im Plug-in definierten Attributs verwendet: „apex_application_page_regions.attribute_01%type". Außerdem wird der Parameter an eine sprechende Variable übergeben: Sprechende Variable „l_text" und Übergabewert „p_region.attribute_01"
3. *Laden einer JavaScript-Datei*
   Funktionsaufruf, um eine JavaScript-Datei zu laden. Beim Aufruf einer CSS-Datei unterscheidet sich die zu verwendende Funktion nur im Namen „apex_css.add_file (…);"
4. *Generierung des Inhalts für die Darstellung auf dem Bildschirm*
   Die Erstellung des Outputs kann beliebig komplex werden, da dies vom Inhalt beziehungsweise vom Ergebnis des Plug-ins abhängig ist. Hier ist besonders die Nutzung der Variablen „p_region.static_id" hervorzuheben. Sie ermöglicht es, den vom Apex-Item gelieferten Regionsnamen im Quellcode zu verwenden. Innerhalb des Items kann der Regionsname vom Entwickler definiert werden oder es wird automatisch ein eindeutiger Name generiert. Dies hat den Vorteil, dass das Plug-in mehrfach auf der Seite aufgerufen werden kann, das erstellte Element jedoch immer einen eindeutigen Namen besitzt.
5. *Debug-Informationen generieren*
   Der gezeigte Code kann an den jeweils relevanten Stellen verwendet werden, um entscheidende Debug-Informationen hinzuzufügen. Diese Informationen können nur vom Entwickler ausgelesen werden.

Sobald man sich an die Art und Weise der Plug-in-Funktionslogik gewöhnt hat, ist es möglich, komplexere Plug-ins mit wenig Aufwand zu erstellen.

Erstellung des jQuery-Image-Slider-Plug-ins

Im vorangegangenen Beispiel wurde eine simple DIV-Box generiert. Das folgende Beispiel geht nun auf die Erstellung eines jQuery-Image-Slider-Plug-ins ein. Der jQuery-Image-S3Slider blendet verschiedene Bilder nacheinander mit einem grafischen

```
-- 1. Funktionsaufruf mit Definition der Parameter
function render_s3slider (
      p_region              in apex_Plug-in.t_region,
      p_Plug-in              in apex_Plug-in.t_Plug-in,
      p_is_printer_friendly in boolean
) return apex_Plug-in.t_region_render_result is

       -- 2. Plug-in Attribute per Variable übergeben
    l_text           apex_application_page_regions.attribute_01%type := p_region.attribute_01;
begin
  -- 3. Laden einer Javascript Datei
  apex_javascript.add_library (
    p_name      => ‚s3slider‘,
    p_directory => p_Plug-in.file_prefix,
    p_version   => null
  );

  -- 4. Generierung des Inhaltes für die Darstellung auf dem Bildschirm
  sys.htp.p(‚<div id="‘||apex_Plug-in_util.escape(p_region.static_id, true)||‘_slider“>‘);
  sys.htp.p(‘Mein erstes Plug-in mit Einstellungswert für l_text: ‘ || to_char(l_text));
  sys.htp.p(‚</div>‘);

  -- 5. Debuginformationen generieren
  if wwv_flow.g_debug then
     wwv_flow.debug(‚DIV container finished.‘);
  end if;
  return null;
end render_s3slider;
```

Listing 1

```
select
  id_col as id,
  ,#OWNER#.my_image_display?p_image_id=‘ || to_char(id) as image,
  text,
  description,
  ,left‘ as position
from my_image_table
```

Listing 2

```
function com_aaw_s3slider(pRegionId, pPlug-inName, pItemList) {
  var aItemList;
  apex.event.trigger(
    $x(pRegionId+“_slider“),
    „apexbeforerefresh“
  );

  var lAjaxRequest = new htmldb_Get(
    null,
    $v(,pFlowId‘),
    „PLUG-IN=“ + pPlug-inName,
    $v(,pFlowStepId‘)
  );
  if (pItemList != „“) {
    aItemList = pItemList.split(„,“);
    for (var i=0;i<aItemList.length;i++) {
      lAjaxRequest.add(aItemList[i], $v(aItemList[i]));
    }
  }
  lAjaxRequest.addParam(,p_debug‘, $v(,pdebug‘));
  var lAjaxResult  = null;
  lAjaxResult = lAjaxRequest.get();
  $x(pRegionId+“_slider“).innerHTML = lAjaxResult;
  eval(pRegionId+‘_run_slider()‘);
  apex.event.trigger(
    $x(pRegionId+“_slider“),
    „apexafterrefresh“
  );
}
```

Listing 3

Übergang ein. Außerdem ist es möglich, individuelle Texte dem Bild hinzufügen

Um solch eine Lösung umzusetzen, muss der Plug-in-Code entsprechend erweitert werden. Dazu die Vorstellung der einzelnen Plug-in-Bereiche mit den hier verwendeten Werten:

- *Name*
  Es werden die Namen definiert und der Plug-in-Typ ausgewählt. Bei dem Feld „Internal Name“ sollte ein möglichst eindeutiger Name verwendet werden. Als Name „S3Slider for Apex”, als Internal Name „COM.AAW.S3SLIDER” und als Type „Region“.
- *Subscription und Settings*
  Diese Werte werden nicht verändert, da kein Master-Plug-in existiert und die Dateien aus der Datenbank geladen werden sollen.
- *PL/SQL-Code*
  Es folgt der etwas komplexere PL/SQL-Code. Als Code-Basis wurde die sehr gut aufbereitete Logik von Carsten Czarski, Oracle Deutschland, verwendet, die er in seinem Label-Cloud-Plug-in nutzt. Auf Basis dieser Grundlage war es möglich, innerhalb von ein paar Stunden eine funktionsfähige Implementierung in Apex zu schaffen. Der Code nutzt zwei Funktionen:

  – *render_s3slider*
    Dieser Teil wird beim Start der Seite einmalig aufgerufen, um die JavaScript-, CSS- und HTML-Komponenten des Slider zu laden. Zum Schluss wird die zweite Funktion aufgerufen, die entsprechend Inhalte ausliest. Hier ist die Bedeutung der Plug-in-Attribute hervorzuheben, da diese an allen entscheidenden Stellen verwendet werden, um dynamisch und eindeutig die Komponenten des Image-Slider abzubilden.
  – *render_s3slider_ajax*
    Dieser Part selektiert die Daten des übergebenen Select und gibt die Inhalte formatiert aus. Als wichtigste Komponente kommt hier der SQL-Handler ins Spiel. Dieser wandelt das vom Apex-Entwickler erstellte Select in eine Cursorähnliche Funktionalität um, zum zeilenbasierenden Auslesen und Verarbeiten der Werte. Dadurch sind ein besseres Fehlerhandling und eine einfache Verwendung der Spalten möglich.

Das entsprechende Listing steht unter http://www.doag.org/go/doagnews/asc zum Download bereit.

Callbacks

Hier sind die PL/SQL-Funktionen referenziert, damit diese während der Laufzeit aufgerufen werden können. Der Render-Function-Name ist „render_s3slider“ und der AJAX-Function-Name lautet „render_s3slider_ajax“.

Standard Attributes

Bei der Definition der Standard-Attribute ist es wichtig, Häkchen bei „Region Source is SQL Statement“ und bei „Region Source Required“ zu setzen. Dadurch trägt der Entwickler ein entsprechendes Select ein. Das hier angegebene Beispiel nutzt genau fünf Spalten, weshalb auch unter „Minimum und Maximum Columns“ jeweils „5“ eingetragen werden muss. Listing 2 zeigt einen Beispiel-Select.

Custom Attributes

In diesem Bereich werden die einzelnen Attribute je nach Typ definiert. Im Beispiel kommen Text-, Number- und Select-Listen-Felder vor. Bei den Select-

Listen muss zusätzlich die mögliche Auswahl vordefiniert sein.

Files

Es werden das Originalprogramm und eine eigens erstellte Erweiterung hochgeladen (aaw_s3slider.js, s3slider.js). Beides wird dann im PL/SQL-Code aufgerufen und im Browser ausgeführt. Die Erweiterung macht das dynamische Refreshen der Region während der Laufzeit möglich, indem beim Aufruf der Funktion ein Datenbank-Request erstellt und der neue Wert aus der Datenbank selektiert wird. Listing 3, Seite 27, zeigt den JavaScript-Code.

Soweit zu den wichtigsten Informationen, die für dieses Plug-in relevant sind. Die letzten Bereiche (Events, Information, Help Text, Comments) werden entweder nicht genutzt oder nur informativ gefüllt (beispielsweise Information -> Version: 1.0). Beim Ausführen des Plug-ins mit einem entsprechenden Select erscheint nun eine ansehnlich inszenierte Apex-Implementation des jQuery-S3Slider. Das Ergebnis ist online unter http://apex.oracle.com/pls/apex/f?p=65560:12 zu sehen. Neben diesem Beispiel existiert auch eine einfache Anleitung zum Erstellen eines Dynamic-Action-Plug-ins unter http://tinyurl.com/7lrstme.

Wichtige Links zum Thema

- SQL-Selektionsbeispiel für das Plug-in: http://tinyurl.com/7dxbp7l
- Original S3Slider-Beispiel: http://tinyurl.com/6be3af
- Generelle Apex-Dokumentation: http://tinyurl.com/898drb8
- Apex_PLUG-IN_UTIL-Dokumentation: http://tinyurl.com/7efamfyQuell-code-Vorlage für das Label-Cloud-Plug-in von Carsten Czarski: http://tinyurl.com/7c696yb
- Buch über Apex Plug-ins: http://tinyurl.com/6lmtsfr

Tobias Arnhold
tobias-arnhold@hotmail.de
http://www.apex-at-work.com

„Bereit für die Cloud und gerüstet für Mobile Apps ...“

Bei der letzten Oracle OpenWorld eröffnete Joel Kallman seine Keynote mit den Worten „I was involved in Apex Development since the first line of code – and since that moment we were ready for the cloud.” Das war 1999. Als Nachfolger einiger anderer Ansätze wurde Apex zunächst eher kritisch beobachtet und nicht wirklich angenommen. Heute stellt Joel Kallman völlig zurecht fest, dass Apex zum Industrie-Standard geworden ist, der auch die Entwicklung und den Betrieb von unternehmenskritischen Applikationen abdeckt und ein modernes Software Engineering ermöglicht. Wie er als Produktverantwortlicher die Entwicklung von Apex, aber auch Test, Dokumentation, Q&A und Produktsupport gewährleistet, beschreibt Kallman sehr schön in seinem Interview auf DOAG.tv unter http://www.doag.org/home/doagtv.html. Zwei Kriterien machen Apex heute so erfolgreich:

- Beim visionären Schwenk auf Cloud Computing sind künftig Entwicklungsumgebungen gefragt, deren Infrastruktur völlig dezentral und Standort-unabhängig ist. Wer weiß, dass Apex mit Apex entwickelt wird und das Team dabei über den ganzen Erdball verteilt sitzt, versteht auch, dass diese Eigenschaften erfüllt sind. Als Bestandteil der Datenbank und mit lediglich einem Browser auf der Client-Seite ist Apex ideal gestaltet.
- Beim Software-Design ist heute schon die Forderung nach „Any Devices“ maßgeblich und dominiert klar vor weiteren funktionalen Erweiterungen. Dass Apex heute in der Lage ist, Applikationen für mobile Endgeräte zu erstellen, hat auch die DOAG schon mehrfach vorgestellt, diskutiert und dokumentiert. Mit der 2012 erscheinenden Apex Version 4.2 und der darin integrierten Funktionalität von jQuery Mobile sieht sich Oracle mit Apex bestens gerüstet für das Mobile Computing.

Patrick Wolf ist einer der Mitglieder des Apex-Entwicklungsteams. Er weist bei der Frage nach einem Migrationsweg aus Oracle Forms wohl zu Recht darauf hin, dass neben der optimalen Architektur auch das Wissen und die Logik aus Forms-Anwendungen sehr gut übertragen werden können.
SQL und PL/SQL sind ebenfalls Bestandteile von Apex. Der Umstieg ist somit für Entwickler einfacher und schneller. Anstelle einer automatisierten Migration sollte man den Wechsel von Forms nach Apex jedoch als Re-Engineering betrachten. Transaktionen lassen sich nicht unbedingt „1:1“ abbilden. Die schrittweise Migration und Übertragung der Logik ist zwar vereinfacht, muss und sollte aber auch im neuen Zielsystem überdacht werden.

Stefan Kinnen, Leiter der Development Community
stefan.kinnen@doag.org