Dieser Artikel stellt Erweiterungen vor, die Apex um eine Reihe äußerst nützlicher Funktionalitäten ergänzen, jedoch nicht im jetzigen Umfang des Werkzeugs implementiert sind. Das Spektrum umfasst Verbesserungsmöglichkeiten für die Benutzerfreundlichkeit von Anwendungsentwicklungen, die Erhöhung der Flexibilität in heterogenen Netzwerkumgebungen sowie die Erweiterung der Datenvisualisierung.

Die Funktionalität von Apex 4.0 erweitern

Ines Möckel, Johannes Kirchner und Sandy Frenzel, OPITZ CONSULTING GmbH

Die Funktion der Autovervollständigung von Eingabefeldern, die Apex aktuell anbietet, unterstützt den Elementtyp „Tabular Form“ leider nicht. Apex bietet in der Version 4.0 zwar den Elementtyp „Text Field with autocomplete“ an. Dieses Textfeld reagiert auf Benutzereingaben und listet Werte dynamisch auf, die bereits eingegebene Zeichenfolgen beinhalten. Die vorgeschlagenen Werte stammen aus einem SELECT-Statement, das in der Konfiguration innerhalb der Apex-Entwicklungsoberfläche hinterlegt ist.

Zur Verbesserung der Bedienbarkeit und zur Erhöhung der Datenqualität ist es allerdings sinnvoll, das Feature der Autovervollständigung in die Eingabefelder der „Tabular Form“ zu integrieren, um die Eingabeprozesse komfortabler, schneller und konsistenter zu realisieren. Leider steht diese mächtige Eigenschaft derzeit für Apex nicht innerhalb der „Tabular Form“ zur Verfügung.

Wenn man verstanden hat, wie Apex den Elementtyp „Text Field with autocomplete“ originär realisiert, erscheinen die erforderlichen Anpassungen für Eingabefelder innerhalb einer „Tabular Form“ recht trivial. Zudem kommt die Implementierung der Autovervollständigung ohne die Integration von zusätzlichen Apex-Plugins und -Bibliotheken aus. Die Daten des zugrundeliegenden SELECT-Statements werden beim Laden der Seite als Skript-Array-Feld in die HTTP-Response integriert, zum Beispiel „function updateAjaxFieldAcData()“:

```
{var gFAHE_HERSTELLERdata=
[„BMW“, „Mercedes Benz“,
...]; …}
```

Dabei werden die entsprechenden Eingabefelder der „Tabular Form“ auf eine Skriptfunktion registriert. Ein gezieltes Nachladen über einen AJAX-Request wäre zwar bei einer überschaubaren Anzahl von Datensätzen ebenfalls denkbar, jedoch lässt sich mit dem beschriebenen Vorgehen ein besseres Ansprechverhalten der Eingabe auf die vorgeschlagenen Werten erzielen. Dabei sollte man allerdings beachten, dass für große vorgeladene Datenmengen ausreichend Übertragungskapazitäten und Client-Hauptspeicher zur Verfügung stehen.

Für das Vorladen der Daten wird innerhalb der Seitenkonfiguration in Apex ein neuer Prozess als anonymer PL/SQL-Block angelegt und darin eine externe Utility-Prozedur aus einem Datenbank-Package ausgeführt. Diese Prozedur integriert über ein dynamisches SELECT-Statement und den Befehl „HTP.p“ den entsprechenden Skriptcode als HTTP-Response in die Seite, die vom Client aufgerufen wird.

```
[...]
HTP.p ('<script type="text/javascript">');
HTP.p ('function updateAjaxFieldAcData(){');

-- Deklaration von AJAX Feld Werten
HTP.p ('var g' || UPPER (pi_ajax_data_field_name)  || 'data=[');
v_stmt := 'SELECT DISTINCT ' || pi_ajax_db_field_name ||' FROM '
        || pi_ajax_db_schema_name || '.' || pi_ajax_db_table_name
        || ' ORDER BY '  || pi_ajax_db_field_name || ' ASC';
v_is_first_value := TRUE;

-- Definition von AJAX Feld Werten
OPEN c_stmt_cursor FOR v_stmt;
LOOP
   FETCH c_stmt_cursor INTO v_ajax_db_field_value;
   EXIT WHEN c_stmt_cursor%NOTFOUND;
   IF LENGTH(v_ajax_db_field_value) > 0 THEN
      IF v_is_first_value THEN
         HTP.p ('"' || v_ajax_db_field_value || '"');
         v_is_first_value := FALSE;
      ELSE
         HTP.p (',"' || v_ajax_db_field_value || '"');
      END IF;
   END IF;
END LOOP;
CLOSE c_stmt_cursor;

HTP.p ('];');

-- Zuordnung von AJAX Feld Werten zu Eingabefeldern
[...]
HTP.p ('updateAjaxFieldAcData();');
HTP.p ('</script>');
[...]
```

Abbildung 1: Auszug aus der Utility-Prozedur

Dabei ist es sinnvoll, die aufgerufene Prozedur möglichst generisch anzulegen und so eine flexible Nutzung zu gewährleisten. Abbildung 1 zeigt einen Auszug aus der Prozedur.

Um die Autovervollständigung ebenfalls auf die Felder einer neu hinzugefügten Zeile zu registrieren, ist zusätzlich die Skriptfunktion „updateAjaxFieldAcData()“ nach dem Skriptaufruf „javascript:addRow();“ einzufügen.

Aufgrund seiner Berechtigungen hat nicht jeder Benutzer Zugriff auf alle Menüpunkte der erstellten Anwendung. Der Apex-Standard gewährleistet lediglich eine starre Navigation zwischen Haupt- und Untermenüpunkten. Diese stößt leider bei einer flexiblen und individuellen Benutzerführung schnell an ihre Grenzen. Wählt der Nutzer beispielsweise einen Hauptmenüpunkt und gelangt darüber in einen für ihn gesperrten Bereich, wird er zu einem Dialog mit einer nicht benutzerfreundlichen Fehlermeldung navigiert. An dieser Stelle endet für den Nutzer die Arbeit mit der Applikation, da weitere Navigationsmöglichkeiten verhindert werden.

Wünschenswert wäre eine Funktion, die nach ihrer Implementierung die dynamische Anpassung der Menünavigation anhand eines vorhandenen Berechtigungskonzepts zulässt. Die Konfiguration des „Tab Target“ eines „Tab Set“ bietet hingegen in der aktuellen Apex-Version nur zwei Möglichkeiten: „Seite in dieser Anwendung“ und „URL“. Über einen gezielten URL-Request sollte es hingegen möglich sein, die Navigation zwischen Haupt- und Untermenüpunkten nach Anwendungsvariablen individuell zu steuern.

Jede Navigation innerhalb von Apex wird über die zentrale „f“-Prozedur gesteuert. Es liegt daher nahe, diese in einer weiteren Prozedur zu kapseln, um sie später bedarfsgerecht aufzurufen. Zu diesem Zweck wird im Apex-Schema eine neue Prozedur mit dem Namen „ff“ angelegt. Parameter sind Array-Felder der Auflistungen von Parameternamen und -werten (siehe Abbildung 2).

Anschließend muss in den Konfigurationseinstellungen von Apex der Aufruf der Prozedur als Wert der URL eingetragen werden, zum Beispiel:

```
!ff?app=&APP_ID.
&page=46&session=&APP_SES-
SION. &user =&APP_USER.
```

Die Prozedur ermittelt über den Parameterwert „&APP_USER.“ die erste Seite der Untermenüpunkte, die zum Hauptmenüpunkt gehören und auf die der angegebene Benutzer zugreifen darf. Diese wird dann als Wert der Parameterliste beim Aufruf der „f“-Prozedur übergeben. Bei dieser Vorgehensweise gilt es jedoch, einige wesentliche Dinge zu beachten: Zum einen muss die „ff“-Prozedur über das „Embedded PL/SQL Gateway“ aufrufbar sein. Dies wird in der Apex-Funktion „wwv_flow_epg_include_mod_local“ konfiguriert. Zum anderen können die Aufrufe, die in der „ff“-Prozedur angesteuert werden, nicht ohne Weiteres auf das Schema der zugrundeliegenden Apex-Seitenverwaltung zugreifen. An dieser Stelle müssen die entsprechenden Synonyme und Rechte vergeben werden.

Apex unterstützt standardmäßig keinen parallelen Einsatz unterschiedlicher Authentifikationsverfahren. In heterogenen Netzwerkumgebungen kann jedoch die Notwendigkeit bestehen, einen flexiblen Zugriff auf „OpenLDAP“ und „Microsoft Active Directory“ zu gewährleisten.

```
[...]
-- Verarbeitung der gefilterten
Parameter und Aufruf der
zentralen Prozedur F
f(   v_f_p_arr ('app')
  || ',:'
  -- Ueberpruefung, ob ein Be-
nutzer in Anwendung auf be-
stimmte Seite zugreifen kann
  || app_util.get_next_
accessible_page(v_f_p_arr
('page'), v_f_p_arr ('user'))
  || ':'
  || v_f_p_arr ('session')
  || ':'
  || v_f_p_arr ('request')
  || ':'
  || v_f_p_arr ('debug')
  || ':'
  || v_f_p_arr ('clearcache')
  || ':'
  || v_inames
  || ':'
  || v_ivalues
  || ':'
  || v_f_p_arr ('printer-
friendly'));
[...]
```

Abbildung 2: Auszug aus der „ff“-Prozedur

Die Anforderung in einem Kundenprojekt verlangt unter Umständen die Möglichkeit der Authentifikation über das vor Ort installierte „OpenLDAP“. Die zu erstellende Apex-Anwendung wird jedoch als In-House-Projekt realisiert, wobei in diesem Fall lediglich ein „Microsoft Active Directory“ zur Verfügung steht. Die Anbindung an den entsprechenden Verzeichnisdienst lässt sich in der Apex-Entwicklungsoberfläche unter „[...] > Gemeinsame Komponenten > Authentifizierungsschemas“ konfigurieren. Bei einem Export beim Dienstleister und dem darauf folgenden Import der Anwendung beim Kunden, müssten alle LDAP-Einstellungen erneut an die jeweilige Umgebung angepasst werden. Hingegen könnte eine entsprechende Erweiterung einem Unternehmen den agilen Einsatz diverser Authentifikationsverfahren sehr wohl ermöglichen.

Um eine flexible Anbindung in unterschiedlichen Umgebungen zu gewährleisten, erfolgt die Prüfung der Benutzer/Kennwort-Kombination in einer ausgelagerten externen Utility-Funktion. Der Aufruf dieser Funktion mit dem Namen „is_authenticated_user“ wird mit der festgelegten Signatur (p_username IN VARCHAR2, p_password IN VARCHAR2) innerhalb der Apex-Entwicklungsoberfläche hinterlegt. Da die Berechtigungseinstellungen der einzeln anzubindenden LDAP-Benutzer in der Apex-Anwendung selbst konfiguriert werden, sollte es eine Anmeldung über einen anwendungsspezifischen Administrator geben, der unabhängig von der LDAP-Umgebung funktioniert. Die spezifischen LDAP-Einstellungen (Serveradresse, Port etc.) sind einmalig in einem Konfigurationsdialog eingestellt oder per Installationsskript in eine Datenbanktabelle der Anwendung gelegt.

Besondere Aufmerksamkeit gilt dabei dem Kennwort des anwendungsspezifischen Administrators. Dieses sollte als Hashwert (z. B. SHA-1) – vorzugsweise mit einem hinterlegten „Salt“ – gespeichert werden, um die Sicherheit der

Anwendung zu erhöhen. Die Anmeldungsprüfung kann nun exakt auf die jeweilige LDAP-Umgebung abgestimmt sein. Während beim „Microsoft Active Directory“ die Authentifizierung über einen „User Principal Name“ (name@domain) durchführbar ist, benötigt „OpenLDAP“ zu jedem Benutzer entsprechende „Distinguished Name“-Parameter. Dazu ein Beispiel:

```
ou=example,ou=users,dc=domai
n,dc=local
```

Diese Parameter sind als Wert in einer Spalte der Benutzertabelle hinterlegt. Die Entscheidung, welche Form der Authentifizierung angewandt wird, kann einmalig konfiguriert oder automatisch ermittelt werden. Auf ein wesentliches Sicherheitsleck sei an dieser Stelle jedoch hingewiesen: Der Parameter „p_password“ wird unverschlüsselt an die Funktion übergeben. Bei einem entsprechenden Angriff könnten die Kennwörter einzelner LDAP-Benutzer an dieser Stelle abgefangen werden.

Apex unterstützt prinzipiell die Darstellung von binären Inhalten (BLOB), jedoch gilt dies nicht für den Elementtyp „Bericht“. Standardmäßig bietet Apex die Nutzung binärer Daten in Eingabemasken über den Elementtyp „Display Image“. Möchte man jedoch die in der Datenbank abgelegten Binärdaten innerhalb eines Berichts anzeigen, ist der Elementtyp „Display Image“ nicht möglich. Es sollen jedoch PDFs sowie Bilder angezeigt werden, die in unterschiedlichen Datenquellen vorliegen. Mithilfe einer neuen Funktion ist es möglich, den gewünschten binären Inhalt (Bilddateien, PDF, etc.) über das „Embedded PL/SQL Gateway“ per HTTP an den Webbrowser zu übermitteln.

Um binäre Daten aus der Datenbank innerhalb eines Berichts anzeigen zu können, ist eine Prozedur „GET_BINARY_CONTENT_FROM_DB“ erforderlich. Diese selektiert die binären Daten und übermittelt diese über das „Embedded PL/SQL Gateway“ mithilfe von HTTP an den Webbrowser. Da sowohl Bilder als auch PDF-Dateien angezeigt werden sollen, muss die Prozedur den Content über Dynamic SQL ermitteln. Hierfür ist es notwendig, die Tabelle, die zu selektierende Spalte und den Primärschlüssel nebst Wert als Parameter an die Prozedur zu übergeben. Auf diese Weise können binäre Daten aus unterschiedlichen Quellen selektiert und an den Webbrowser übertragen werden, ohne für jede Quelle eine neue Prozedur anlegen zu müssen. Nachdem man den Content ermittelt hat, muss der Header an den Browser übergeben werden, wobei die Größe sowie der Name der Datei bekannt sein müssen. Anschließend werden die Daten an den Browser übergeben. In Abbildung 3 sind die notwendigen PL/SQL-Aufrufe dargestellt.

Diese Prozedur ermittelt den darzustellenden Content, ist jedoch noch nicht über das „Embedded PL/SQL Gateway“ aufrufbar. Um dem „Embedded PL/SQL Gateway“ den Zugriff auf die Prozedur zu ermöglichen, muss diese noch in der Apex-Funktion „WWV_FLOW_EPG_INCLUDE_MOD_LOCAL“ registriert sein. Erst mit dieser Bekanntgabe kann der Webbrowser auf die Prozedur zugreifen und die binären Inhalte darstellen.

```
[...]
   -- Mimetype des Bildes für den Browser setzen
   OWA_UTIL.mime_header (NVL (v_mine_type, 'application/octet'),
FALSE);

   -- Setzen der Dateigroesse, um Browser mitzuteilen wie gross
Datei wird
   HTP.p ('Content-length: ' || v_blob_content_length);
   [...]

   -- Filterung Dateiname - unnoetige Zeichen entfernen
   v_file_name := REGEXP_REPLACE (v_file_name, ,[^a-zA-Z0-9_-]+',
,_');
   [...]

   -- Setzen des Dateinamens, um Browser mitzuteilen wie Datei
heisst
   HTP.p ('Content-Disposition: filename="' || v_file_name ||
'"');
   OWA_UTIL.http_header_close;

   -- Daten an den Browser senden - dieser stellt das Bild dar
   WPG_DOCLOAD.download_file (v_blob_content);
[...]
```

Abbildung 3: Übergabe Binärdaten an Browser

Ines Möckel
OPITZ CONSULTING GmbH
ines.moeckel@opitz-consulting.com

Johannes Kirchner
johannes.kirchner@opitz-consulting.com

Sandy Frenzel
sandy.frenzel@opitz-consulting.com