APPLICATION EXPRESS "CHEAT SHEET 3"

Karen Cannell, Integra Technology Consulting

Introduction

Remember the old days of cramming all the need-to-know information onto one "cheat sheet" for that important calculus exam? Back then, we were allowed a single piece of paper, front and back, onto which we could stuff all formulas, notes or other information we may need for the exam.

When dealing with software, we don't have a formula for everything, but we do have attributes, settings and syntax. We don't always have a single how-to answer, but an example often provides the guidance needed to address our current problem. With Oracle Application Express (APEX), we have formulas, attributes, settings and syntax, a mix of several languages, templates, style sheets and a myriad of wizards to wrestle with in the development cycle. An APEX quick reference of some sort is in order.

While it is not possible to cram all-one-needs-to-know about APEX onto one two-sided page, or even into one article, it is possible to assemble a collection of the most-used formulas, tips, tricks and syntax. The following is a collection of those hints, tips, how-to's and syntax "formulas" that I found myself referencing most often as I worked on my first few APEX applications. I offer them here as a collection – a cheat sheet – for others to use and build upon in their development process.

This is not an in-depth treatment of any of these topics; it is a high-level reference with a few simple examples that demonstrate how to proceed. Where applicable I include a reference to more detailed information. I present this cheat sheet as a starting point for others to use and build upon in their development process.

F?P Syntax

The f?p syntax is simply a URL that defines a complete reference to an APEX application page, including session, page and data elements, debug, caching and printer-friendly indicators.

The f?p syntax is central to APEX. It contains the directions for how a page is rendered. It tells you (and the APEX engine) what application is running, on what page, with what variables and settings. Understanding f?p is the key to building custom pages and navigation within your applications, and it is the roadmap to learning more about APEX by reviewing what was done where and how. In short, f?p is essential – learn it and understand it, because you are going to be reading, generating or consuming it throughout your work with APEX.

The full f?p syntax has two parts. The first portion of the URL is the standard Apache syntax to the APEX server, and is the same for all APEX pages on that server:

```
http://<hostname>:<port>/pls/apex/...
```

where <hostname> is the name of the host machine and <port> is the port on which the HTTP server is listening. The default HTTP server port is 7777. Your deployment may use port 7778 or something different.

The second portion of the f?p syntax is a call to the core APEX procedure f with its single parameter p. F is a stored database PL/SQL procedure that renders HTML pages. The parameter p is a colon-delimited string of nine arguments that instructs the f procedure which application and page to render using the designated settings, values and indicators.

```
f?p=(App_Id:Page:Session:Request:Debug:ClearCache:ItemNames:ItemValues:PrinterFriendly)
```

Table 1 summarizes the nine components of the p parameter of the f procedure. I remember them by the silly phrase:

APEX Programmers **S**martly **R**equest **D**ouble **C**ash **IN** **V**irtual **P**rograms

for

App Id : **P**age : **S**ession : **R**equest : **D**ebug : **C**lear Cache : **I**tem **N**ame(s) :**V**alue(s) : **P**rinter Friendly

Of course, you may develop your own way of remembering the P arguments.

The colons between the arguments are essential to maintain the order of the arguments. An argument may be omitted, but the colons must be entered to maintain order if later arguments in the string are provided. The final colons may be omitted if all remaining values are defaulted.

Table 1 – f?p Parameter Arguments

Argument	**Definition**	**Common Usages Notes,** **Built-In Substitution String, Syntax**
Application	The APEX application number.	Application number, or APP_ID or APP_ALIAS
Page	Page number or page alias.	Page number or Page Alias name, or APP_PAGE_ID to reference the current page.
Session Identifier	Session identifier.	APP_SESSION or SESSION
Request	Request – often from the button. The HTML request.	REQUEST or the actual request, often the Button name
Argument	**Definition**	**Common Usages Notes,** **Built-In Substitution String, Syntax**
Debug	Debug flag. ON displays embedded debug information.	YES or NO, must be uppercase.
Clear Cache	Clear Cache for the listed Page number(s)or Page Alias(es)	Page number(s) or Page Alias(es) of the page(s) to clear, or the keyword APP to clear all pages.
Item Names	List of Item Names	P1_ITEM, P2_ITEM, etc. No punctuation on the item names
Item Values	List of Item Values, in the same order as the Item Names.	:P1_ITEM, :P2_ITEM bind variable or &P1_ITEM., &P2_ITEM. substitution string Do not forget the trailing . !
Printer Friendly	Printer Friendly flag. ON renders a printer friendly version of the page.	YES, NO, must be uppercase.

Exactly how to reference the values in construction of the P parameter string depends on where the f?p syntax is being constructed: in a URL, embedded in HTML, in PL/SQL or within an APEX construct. Some of the most common usages include:

- As part of a URL:

  `http://tunahuntress.tunahunter.com:7777/pls/apex/f?p=700:12:6395099096881925::NO::P12_COMMITTEE_CODE,P12_COMMITTEE_NAME:459,KALEIDOSCOPE%20LIASONS`

  This example references application 700, page 12 for the session id 6395099096881925 (possibly expired by now), no debug, passing the values 459 and KALEIDOSCOPE%20LIASONS for items P12_COMMITTEE_CODE and

P12_COMMITTEE_NAME. In such a reference having a specific session identifier, if the specified session is indeed expired, and the application requires authentication, the user will be prompted to log in to the APEX application.

- In HTML, as in an APEX URL Target:

```
<a href= ”f?p=(:APP_ID:12:APP_SESSION::NO::P12_COMMITTEE_CODE,P12_COMMITTEE_NAME:
&P12_COMMITTEE_CODE.,&P12_COMMITTEE_NAME.)”>
```

This example references page 12 in application APP_ID for the current session, no debug, passing the value of P12_COMMITTEE_CODE and P12_COMMITTEE_NAME for items P12_COMMITTEE_CODE and P12_COMMITTEE_NAME.

- In the APEX builder ,as a URL argument to a javascript function:

```
javascript:popupURL('f?p=&APP_ID.:46:&SESSION.::NO::P46_MEMBERID:&P44_MEMBERID.')
;
```

The f?p in this example references the current application, page 46, the current session, default request, no debug, setting item P46_MEMBERID with the value of P44_MEMBERID. The javascript: popupURL call opens a popup window with the contents of the page rendered by the f?p call.

- In PL/SQL, building an HTML string:

```
  l_anchor_html := ‘<p><a href="http://'||
                   OWA_UTIL.GET_CGI_ENV('HTTP_HOST')||
                   OWA_UTIL.GET_CGI_ENV('SCRIPT_NAME')||
   '/f?p=550:33:::::'||'P33_REPORT_SEARCH:'||:P1_LAST||'">View KALEIDOSCOPE
Request for: '||
       :P1_FIRST||' '||:P1_LAST||'</a></p>';
```

This example references the current APEX server, application 550, page 3, for no specific session (so a new one will be initiated; the user will be asked to log in), no debug, setting P33_REPORT_SEARCH to the value of P1_LAST. The entire f?p link is hidden beneath the display anchor text “View KALEIDOSCOPE Request for: <P1_FIRST> <P1_LAST>” where P1_FIRST and P1_LAST are displayed as the values of the P1_FIRST and P1_LAST items. A click on the resulting HTML link will bring the user to a login screen, and then to APEX application 550, page 3, with all default actions processed using the P1_LAST value in the P33_REPORT_SEARCH field. That is the long way of saying this code creates a link that brings the user to the Search page, searching on the value of P1_LAST.

Learn from APEX

A major benefit of APEX being developed in APEX is that we can learn from looking at the core APEX applications. The core APEX applications are themselves exported APEX applications that can be imported into your APEX environment for inspection. While these imported APEX core applications cannot be executed because of the security structure, they are extremely valuable as learning tools.

Given that we know the f?p syntax, we can determine where in the APEX code a particular feature is programmed by dissecting the URL. We can then go to that application and page as an APEX developer to view how the APEX team did their magic.

To examine the APEX code, import the APEX application into an APEX workspace, and examine it as you would any other APEX application. Import and install the APEX applications into a separate (separate from your work) workspace, created just for this purpose, to segregate the APEX code from your code.

To import an APEX core application for inspection:

1. Create a new workspace, APEX_APPS, for example, to hold the APEX core applications. You will need close to 200MB for this workspace to import all the core applications, so you may want to create the user and default tablespace first from outside of APEX. Create a new APEX Developer user with access to the APEX_APPS workspace.

2. Determine the application number for the feature you are interested in. Look in the URL. Recall from the f?p syntax section that the application number is the first number in the parameter sequence, followed by the page number, then the session id (we can ignore that for this purpose).
3. Go to the file system APEX installation directory, in the builder directory. The series of nine f4nnn.sql files in this directory are the APEX core applications.
4. Log in as the APEX user with access to the APEX_APPS workspace. Import the f4000.sql (or f4101.sql, etc.) application into the workspace, using the Import Application wizard:
   - Click on the Import Application button.
   - On the Import page:
     - Select the application file to Import: C:\<apex_install_home>\builder\f4000.sql
     - File Type: Application Page or Component Export
     - File Character Set: Leave the default
     - Click Next.
   - Click the Install button.
   - On the Install Application page:
     - Parse As Schema: APEX_APPS (use the name of your new workspace)
     - Build Status: Run and Build Application
     - Install As Application: Change Application ID ß **VERY IMPORTANT. Be sure to change the application id, since you do not want to overwrite the real application!**
     - New Application: 400 (or use your own numbering scheme for the imported APEX core applications)
   - Click the Install Application button.
   - On the Application Installed page, click on the Edit Application Attributes link.
   - On the Application Attributes page:
   - For the Name, add the number of the file loaded to the front of the name, or use some other convention to clearly identify the APEX application code. For example: 4000-APEX Application Builder
   - Application Alias: CLEAR THIS FIELD so it is NULL. ß **VERY IMPORTANT, you do not want to interfere with the operating core application with the same alias.**
   - Click Apply Changes.

5. View the APEX application. Recall the page number from step 2. Go to that page and view how the APEX developers achieved the feature that caught your eye. In most cases, looking at the example gives enough clues to employ the same tactics in your own application.

Referencing Items: When to use :, & _. , #...#, V and NV

Within APEX, there are no less than five ways to reference the different types of variables. Knowing which syntax to use when saves development time in reduced edit and debug cycles.

Remember that APEX is retrieving session state, much like retrieving the value of a global variable. The different reference types are required because the APEX engine executes different types of code – SQL, PL/SQL, HTML - in rendering the various components of the page. The correct reference type to use depends on where the reference is made. Table 2 is a quick reference to the substitution string reference types, their syntax and their most common usage notes.

Table 2 – Substitution String Reference Types

Reference Type	Syntax	Usage	Usage Notes
Bind Variable	:P1_ITEM	Within SQL statements; page processes, validations, computations. N.	Up to 30 characters
Substitution String	&P1_ITEM.	f?p=&APP_ID.:1:&APP_SESSION::::	Remember the . at the end! The reference is not interpreted without it.
PL/SQL	V('P1_CHAR_ITEM') NV('P1_NUMERIC_ITEM)'	In PL/SQL procedures.	Must be current, authenticated session – V() and NV() do not work in background jobs!
Direct PL/SQL*	APEX_APPLICATION.G_FLOW_ID	In PL/SQL procedure	Must be current, authenticated session – no background jobs.
Template Substitution	#P1_ITEM# #IMAGE_PREFIX#	Referencing report column values, within templates	Examine the APEX Page Region and Region templates for examples.

* See the Oracle Database Application Express User's Guide for the complete list of APEX_APPLICATION and built-in substitution strings.

In general:

- Use the bind variable syntax within SQL statements for page processes, validations and computations.
- Use the substitution string reference for substitution within HTML text, as when editing page items.
- Use the V and NV functions or Direct SQL syntax to reference variables from within PL/SQL stored procedures, including session variables and page items. V and NV are database functions that return the value of character and numeric items, respectively.
- Use the #P1_ITEM# template substitution syntax for system-defined substitution variables, as seen in page, region, report and other templates.

Use of V and NV

A caution when using V or NV – these are not DETERMINISTIC functions, so something as innocent looking as SELECT x,y,z FROM abc WHERE w = V('P1_ITEM') may execute once for every row in table abc. The obvious solution is to avoid using V and NV in SQL statements – in reports, for example. Use bind variables instead, because of the potential performance impact of calling the V or NV function once for each row.

An alternative is to try using the wrapper functions suggested on the Inside Oracle APEX forum, which adds the DETERMINISTIC optimizer hint to the query. See the full information on the wrapper function and a full description of the V and NV performance issue at the Inside APEX blog entry Drop in replacement for V and NV, http://inside-apex.blogspot.com/2006/12/drop-in-replacement-for-v-and-nv.html, with full credit to Patrick Wolf for this contribution to the APEX community.

One more note on the Direct SQL syntax, V and NV: they can read session state only when called directly from an authenticated APEX session. Using Direct SQL syntax, V or NV in a detached background or scheduled job does not work. When you really need to access APEX values from a detached job, stored the values in tables and access the values from the database.

Substitution String Syntax Examples

The following simple examples illustrate the use of each of the substitution string reference types to achieve similar results:

- From within an HTML Region, Header or Footer, use the substitution string syntax:

```
<a href="f?p=&APP_ALIAS.:4:&APP_SESSION."Click here for Page 5</a>
```

- In PL/SQL, as in a Process, Validation or Computation, use the Direct PL/SQL syntax or V function call:

```
htf.anchor('f?p='||APEX_APPLICATION.G_FLOW_ID||':4:'||V('APP_SESSION'),'Click for Page 4');
```

or

```
htf.anchor('f?p='||V('APP_ID')||':4:'||V('APP_SESSION'),'Click for Page 4');
```

- In a SQL Query, as in a Report, use the bind variable syntax:

```
SELECT htf.anchor('f?p='||:APP_ID'||':4:'||:APP_SESSION,'Click for Page 4') FROM DUAL;
```

or

```
SELECT * FROM contacts WHERE contact_username = :APP_USER
```

Built-in Substitution Strings

APEX includes a number of built-in substitution strings that may be referenced in your applications. These include strings that are useful in f?p constructs: the application APP_ID for the application identifier, APP_PAGE_ID for the current page, APP_SESSION and SESSION for the current session, DEBUG and PRINTER_FRIENDLY flags, REQUEST and more. See Appendix A for a list of APEX built-in substitution strings. See the Oracle Database Application Express User's Guide for the complete list with descriptions.

About Case

APEX can be annoyingly fussy when it comes to the case of your code when referencing items, database objects and session state. Use the wrong case, and your code will not run as intended. The reference will be ignored, translated literally, or cause an error in page rendering. In short:

- APEX item names work in either case, uppercase or lower.
- Database object names – tables and views – are case-sensitive when in quotes, and must be uppercase. For example, select 1 from "dual" will fail, select 1 from "DUAL" will not. A simple solution here is to not use quotes around table names. Beware that APEX adds the quotes on generated elements (report queries in a report created from a wizard, for example), so if you copy then edit, make sure you use the appropriate case if there are quotes, or, remove the quotes.
- Reference to substitution strings should be in UPPERCASE, as in :APP_ID, &SESSION. or PRINTER_FRIENDLY.
- YES and NO DEBUG and PRINTER_FRIENDLY options in the f procedure p argument must be uppercase.
- References to button names are case sensitive.
- APEX passwords ARE case sensitive!

Page, Region and Item Types, Positions and Defaults

APEX default page, region and item settings determine the overall layout and format of a page, and for most instances, these defaults are OK. An APEX page is a collection of regions. Each region contains a collection of items. APEX provides a series of page, region and item types that cover most standard web application page layouts and elements, and a series of themes that provide a standard look-and-feel across the page, region and item types of an application. Understanding the APEX page, region and item types and their default positions and formats enables a developer to make significant layout changes with simple item type, position and format changes.

Basic Item Types

The Basic Item Types are just what one would expect from their name – a collection of basic web page item type: text, display only, select list, radio group, etc. A change in the Item Type changes the format attributes of the item, which is often a quick way to get the desired look. Experiment with the APEX-supplied item types, as these fit most needs for text fields, display only fields, text areas with and without editors, select, pop-up and managed lists, date pickers of several formats, checkboxes and radio buttons. If more item types are needed, they can be derived from the existing item types. Figure 1 displays some default APEX item types; all of these are selectable from the APEX Page Item Type select list.

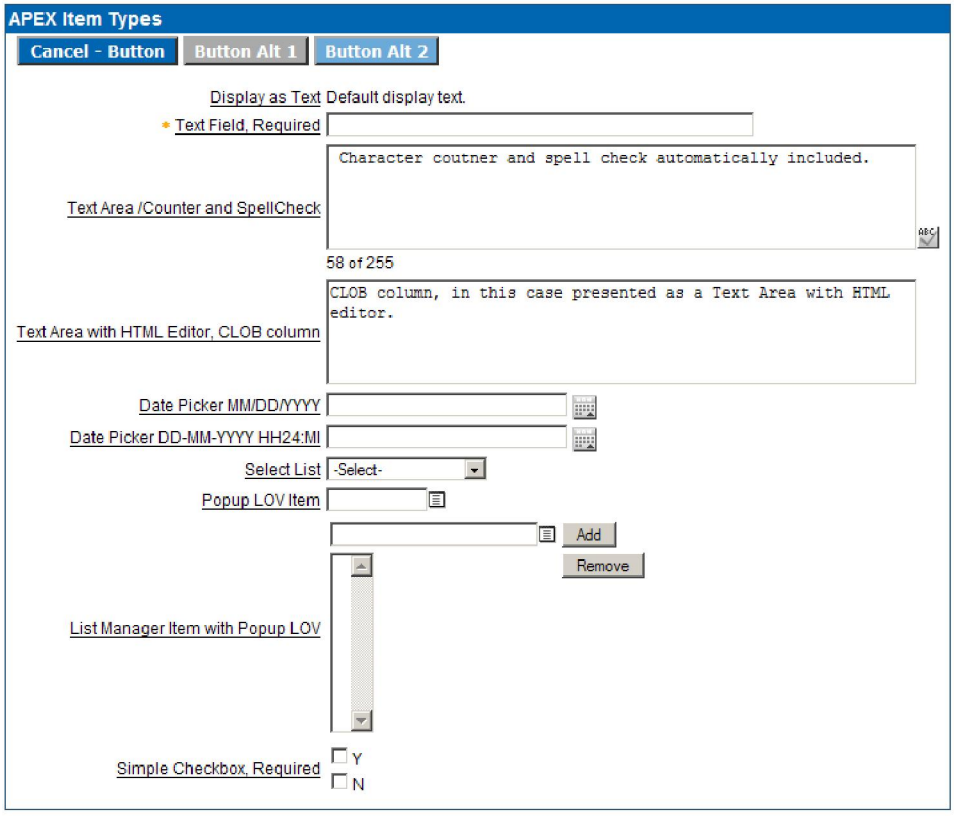

Figure 1 - A few APEX Default Item Types

Advanced Item Types

In addition to the basic items types APEX includes some more interesting item types that provide a bit more functionality "out-of-the-box" for both developers and users. These item types include a File Browser, a Multiple Select item, a Shuttle, a Password item, and a Stop and start table item. Note the more complex items of Text Field with Calculator Popup, Text Area with Spell Checker or Counter, and separate HTML Editor types. These are convenient in that add-on Calculator, Spell Checker and HTML Editor features are incorporated in a single item type. Figure 2 displays samples of these item types. Note that the Multiple Select and Shuttle items return a list of values separated by a delimiter. Code that uses these items must account for the list rather than a single value. A full discussion of how to do this is beyond the scope of this article, however, the APEX Form has numerous examples of how to use the values returned by a multi-select list or shuttle. The Start and stop table item is not really a visual item. It inserts an HTML table end tag (</table>) followed by an HTML table open tag (<table>). It is used to stop and start an HTML table within a region to control region width. Figure 2 displays a sample of these more advanced item types. Again, the reader is encouraged to experiment with these items types and adopt those that fit the needs of your application.

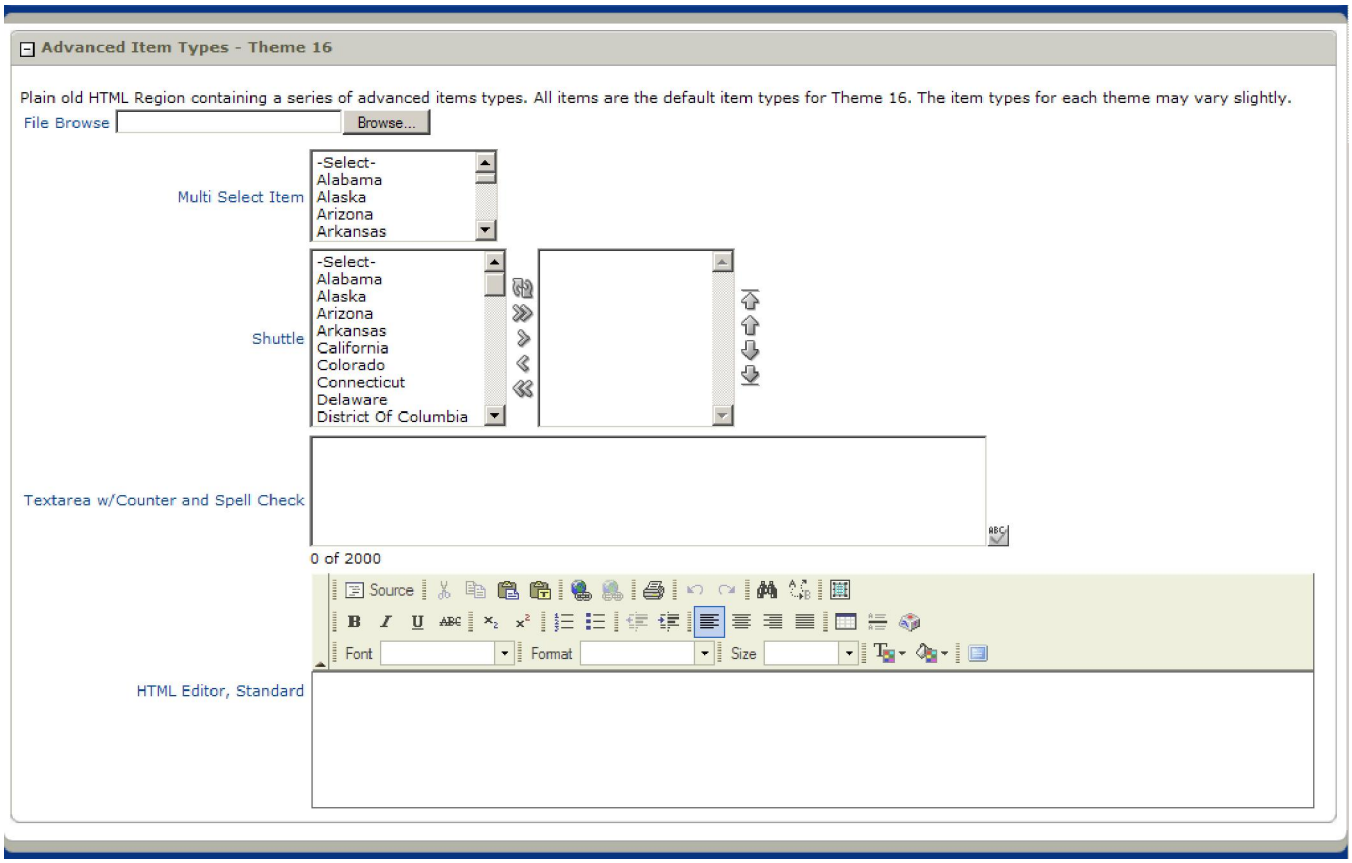

Figure 2 – Advanced Item Types

.

Item Position

APEX item position within a region is controlled by setting in the Displayed element, pictured in Figure 3. The **Begin on New Line?** and **New Field?** Displayed item attributes control where an item is placed relative to the preceding items on the page. Begin on New Line places the item on a new line. New Field places the item in the same row but in a new column. The default column width is determined by item defaults, and defaults to the widest item in the region. Adjusting **ColSpan** increases the width of a column; adjusting **RowSpan** adjusts the height of the item. **RowSpan** could be increased for text areas, for example.

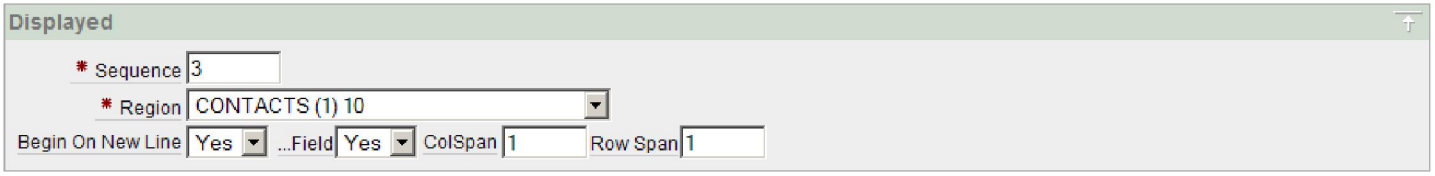

Figure 3 – Page Item Displayed Attributes

Drag and Drop Item Layout

The Drag and Drop Item Layout page allows one to quickly and easily align, reorder and change the type of items in a region. The Drag and Drop Item Layout page is accessed from the Items region of the Page, by clicking on the rectangular icon to the right of the Items region name subheading, just right of the up/down arrow Reorder Region icons (see Figure 4).

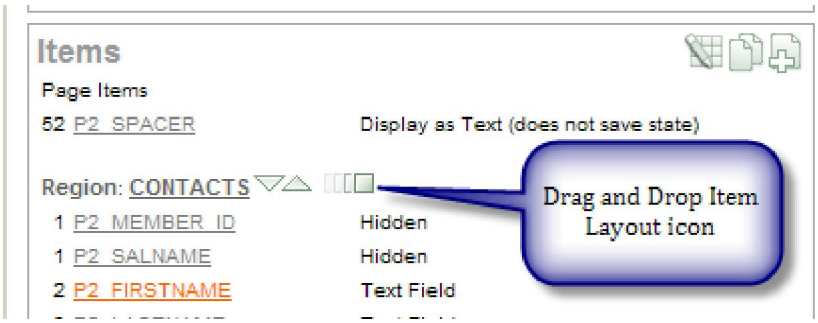

Figure 4 – Drag and Drop Item Layout Icon

On the Drag and Drop Item Layout interface, displayed in Figure 5, one can click and drag to create one or more new items, reorder existing items, or drop items into the recycle bin. It is a graphical representation of your page, where each item is represented by a rectangle on a grid. Each item rectangle has an icon that indicates its item type. Dragging item rectangles around on the gird rearranges items in the region. The Drag and Drop Item Layout page is a quick and easy way to adjust item positions and types, or to add multiple items to a page.

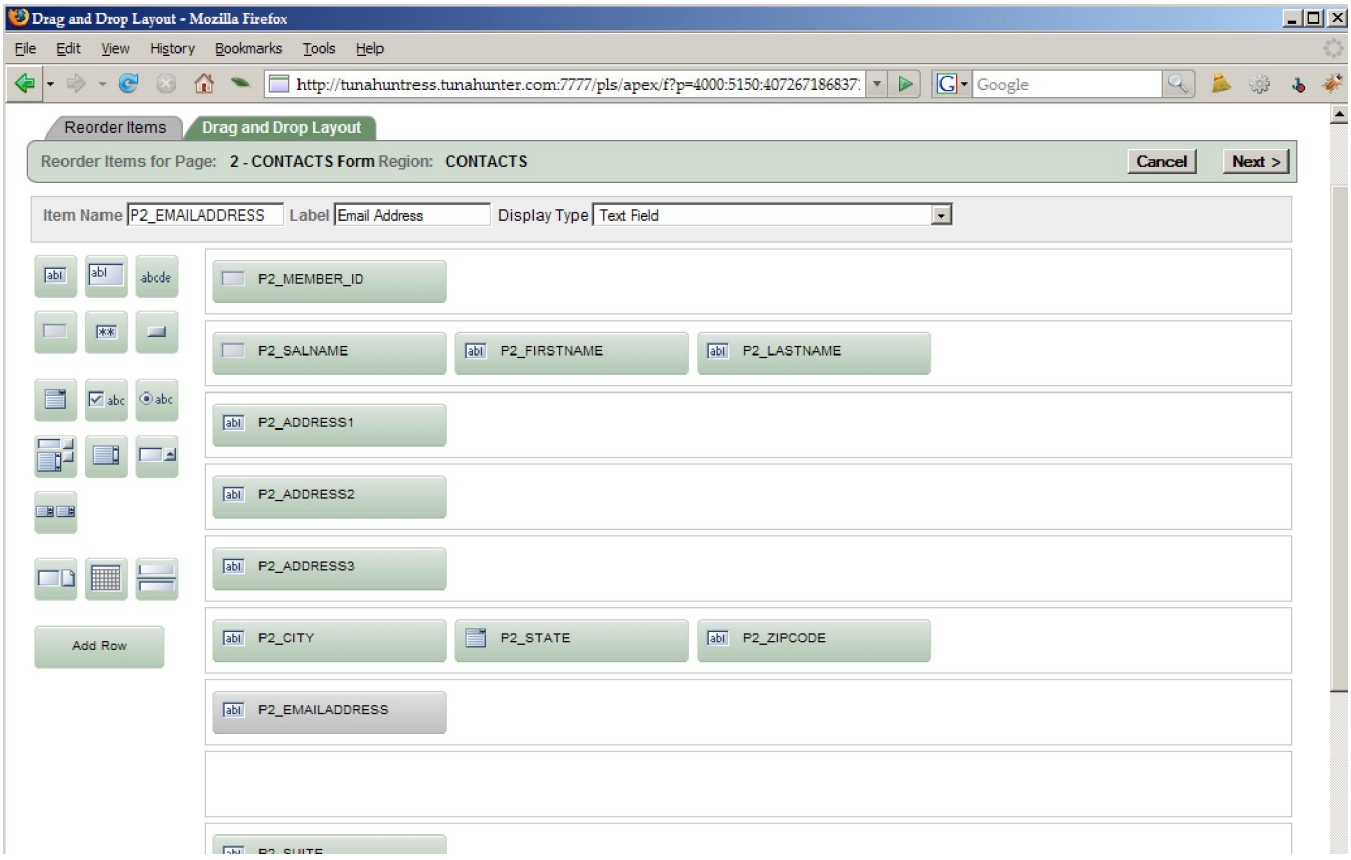

Figure 5 – Drag and Drop Item Layout Interface

User Interface Defaults

The APEX User Interface Defaults feature allows a developer to set default display attributes for all the columns in a table. While it seems time consuming to set up, it is worth doing if you are going to use the same table columns in more than one page, for example in a form and a report. Access the User Interface Defaults from the Application Builder à Shared Components à User Interface Defaults menu, then select the table or view name for which to set defaults. The interface allows for setting item type, label, format, alignment, form and report sequence, required or not, group by or aggregate by and more. Figure 6 illustrates the Grid Edit view of the User Interface Defaults interface.

The temptation is to rush through and see what a new form looks like with all the default settings – then go back and do the User Interface Default formatting. The reality is that from here on, you are doing your formatting twice. The recommendation is to set the User Interface Default formats as soon as possible in the development cycle.

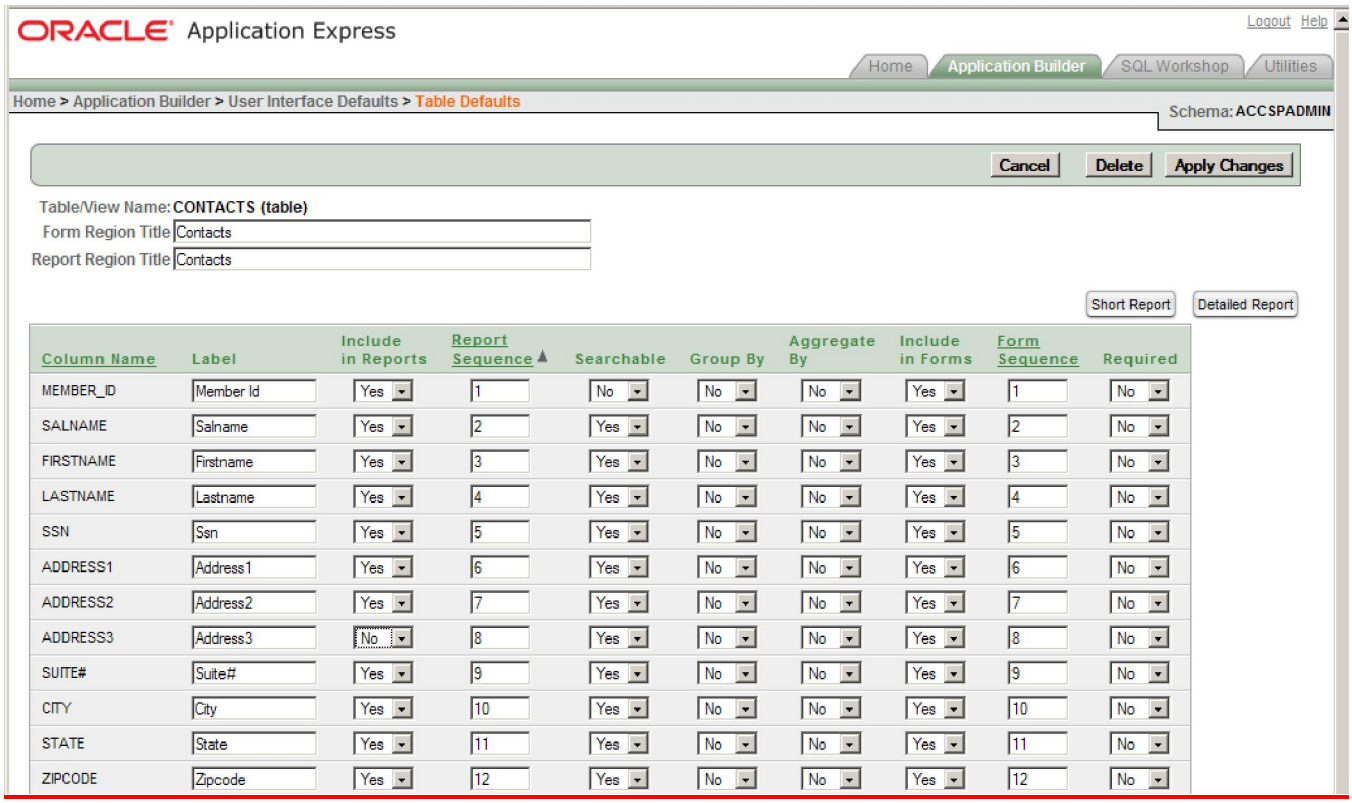

Figure 6 – User Interface Defaults – Grid Edit Interface

Layout Control

The appearance of most APEX pages is determined by the theme and the page, region and item attributes. Page or form layout is generally left to the defaults, and for most purposes, this is acceptable.

Themes control the overall look and feel of the application: color scheme and fonts. Regions control what is placed where on a page, and the general format of that placement window.

Page Region

One question I had each time I created a new region was Which Page Region to use? The Page Region Diagram, accessible by clicking on the flashlight icon to the right of the Region Display Point select list, displays the default locations for standard regions on a page. Figure 7 shows the default APEX Page Region locations. These page region locations may vary slightly from theme to theme. This Page Region Diagram shows that Region 3 is best for Sidebar

Regions and the Page Template Body regions are best for the core application regions. Early on, I printed this diagram for quick reference. In short time I was able to retire my printed page region diagram.

Region Templates

The next question when creating a region is which region template to use? What do the region templates look like? Here I created my own visual cheat sheet, so I could see which region template was which. Over time, I gravitated to a few that I used most often. Note that the region templates will vary slightly in look and feel from theme to theme. Figure 8 shows a collection of default region templates for my current theme.

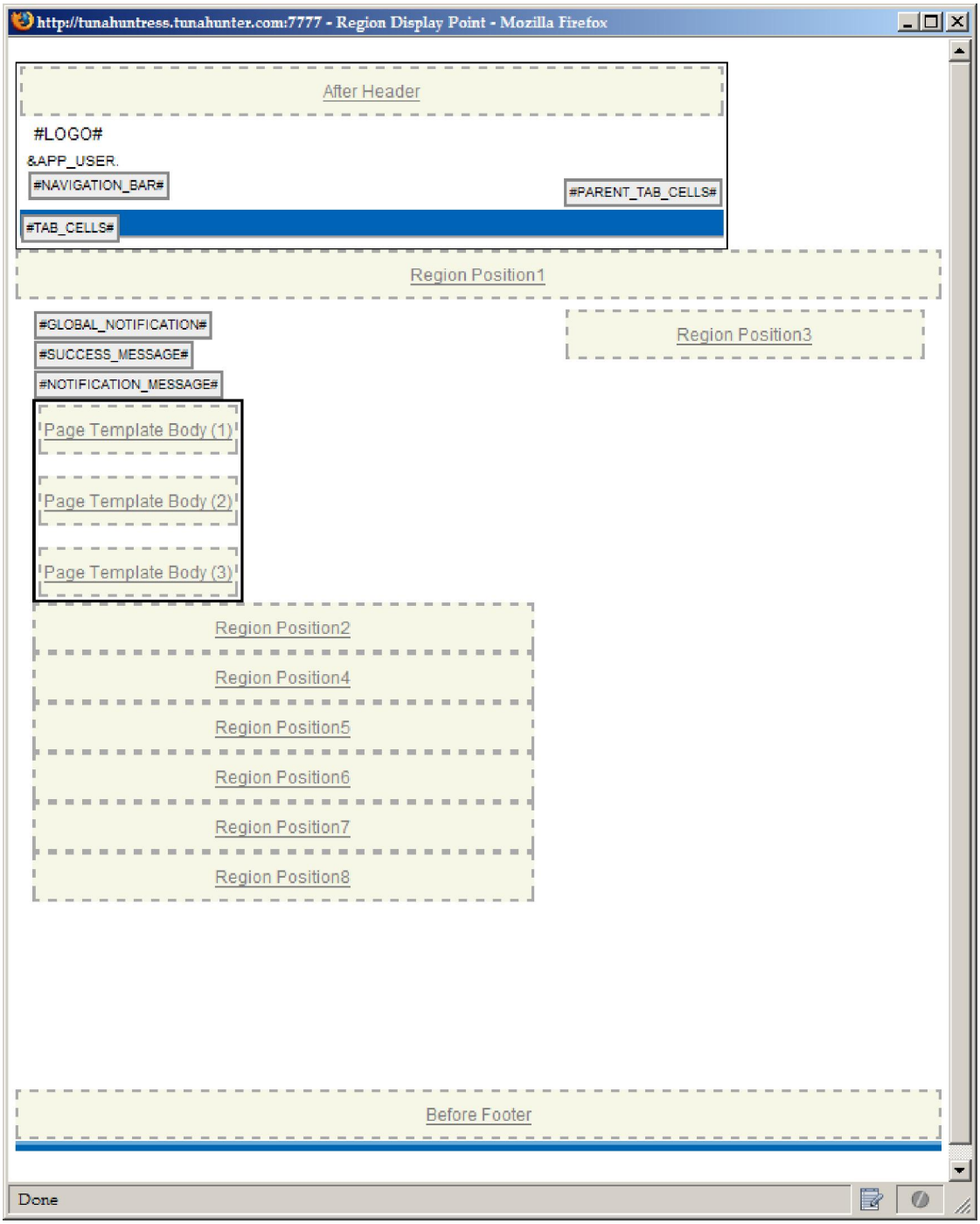

Figure 7 – APEX Default Region Template Locations

Figure 8 – Default Region Templates – Theme 16

Forcing Layout Behavior and Item Alignment

Sometimes the APEX default item or region alignment is not acceptable. Sometimes you need to line page or region elements up exactly, for a more planned, polished result. Figure 9 shows an application page where the center regions required some adjusting to align with each other, as illustrated. There are a few quick tips for setting the size and alignment of APEX components:

Regions

Set the size of the region specifically by adding height and width attributes in the Region Header:

```
<table height="400" width="400"><tr><td>
```

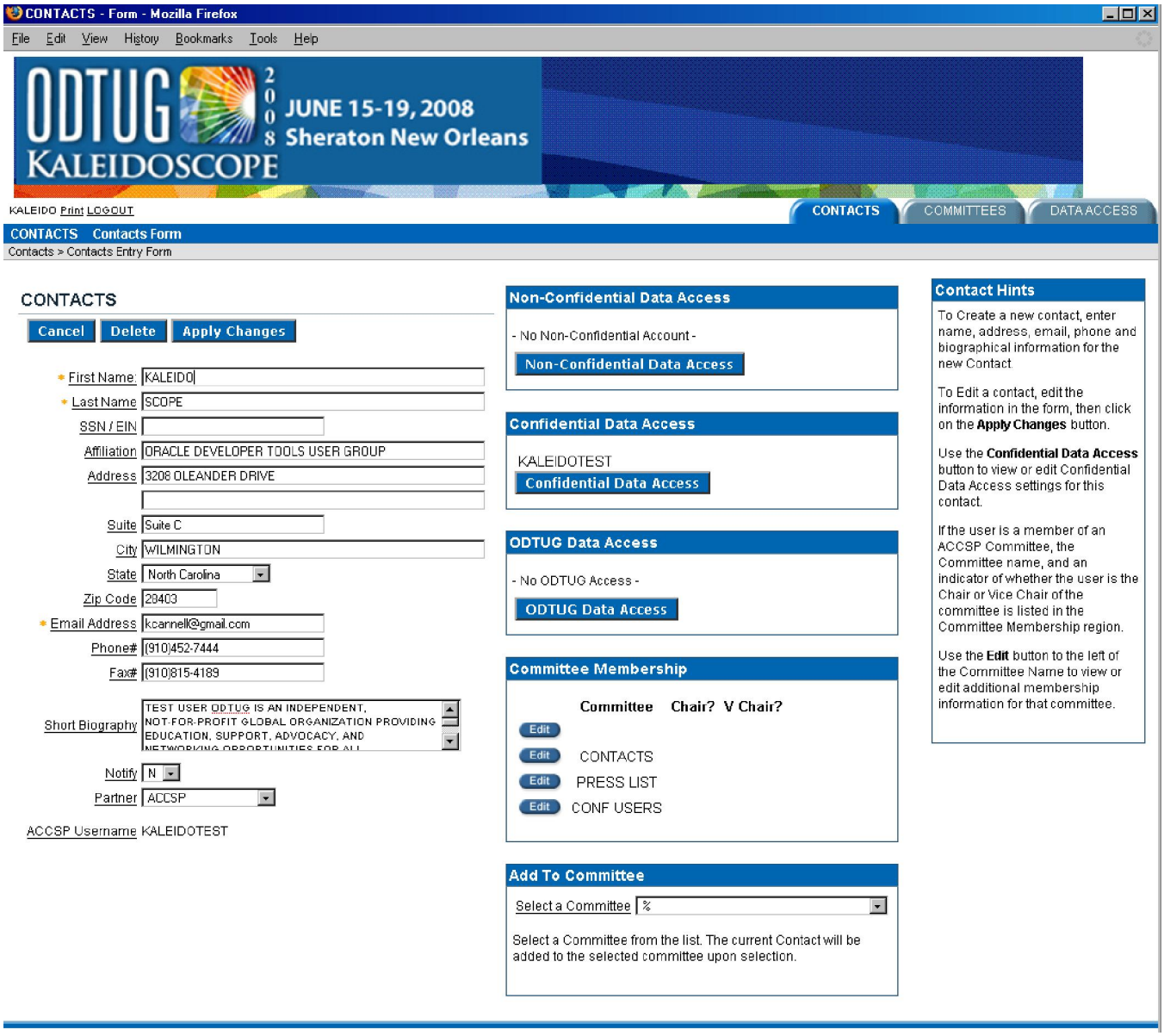

Figure 9 – Forced Region Alignment

Then, close the scope of those attributes in the Region Footer:

```
</td></tr></table>
```

An alternative way to force a region's height or width is to use a simple image in the Region Header, and control the size of that image, forcing the region into the desired horizontal or vertical alignment. In the Region Header, enter something like:

```
<img src="#IMAGE_PREFIX#/1px_trans.gif" width="355" height="0">
```

Use either of the above ways of controlling region size in all of the regions you need to align. For example, in this sample layout of one form region, four center regions and one sidebar region, the central regions do not automatically match in

width. Applying either of the above techniques forces the central regions to the same width, achieving the desired center region alignment:

Select Lists

Set the size of a select list by using an n HTML style tag. Make sure that your width accommodates the longest text in your list, or the text will be cut off. In the HTML Form Element Attributes enter this text:

```
style="width:80;"
```

Form Fields

Use the APEX Displayed element attributes to control the layout of elements relative to each other. Use the Begin On New Line and Field boxes to control whether the item starts a new rows or column relative to the previous item. Change the ColSpan and RowSpan settings to control the placement of an item relative to the items preceding and following it. Note that all of these Display settings are relative to other items, not exact settings. Figure 10 displays settings for an item in the Contacts region that begins on a new line and on a new field, spans four columns (perhaps the whole width of the page?) and one row.

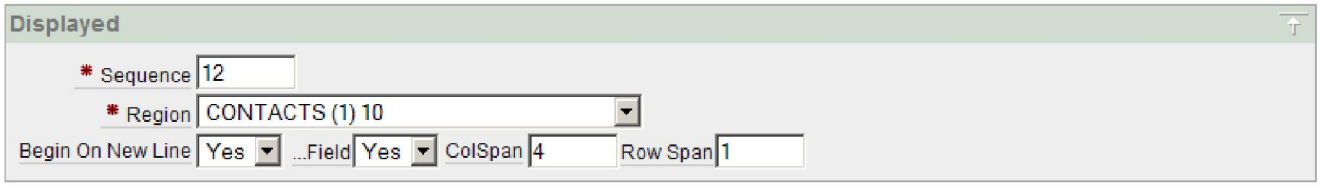

Figure 10 –Page Item Displayed Attributes

For more exact settings, enter HTML style or JavaScript settings in the Element Attributes field of tabular form elements to control tabular form field features, as in Figure 11. For example:

- To control the field width:

  ```
  style="width:80;"
  ```

- To force characters to uppercase:

  ```
  style="text-transform:uppercase;"
  ```

- To set the maximum number of characters in a field:

  ```
  onFocus="javascript:this.maxlength=30;"
  ```

- To set the font, width, height, and scrollbar settings of a text area, and to force the entered text to uppercase:

  ```
  style="font-family:Arial;font-size:10px; height:50px;width:300px;overflow:auto;"
  onBlur="javascript:{this.value = this.value.toUpperCase(); }"
  ```

Figure 11 – JavaScript entry in Page Item Element Attributes

As you can see from these few examples, you can adjust just about anything, and there is more than one way to achieve the same result. The above examples are simple cases of what can be achieved. See more examples on the APEX Forums and the APEX Studio. The full link for these resources is listed at the close of this article.

Customized Look-and-Feel

APEX allows for simple to complicated look-and-feel customizations. Simple customizations will suffice for many situations where the default themes and templates are acceptable. When a corporate look-and-feel is mandated, APEX allows for such customizations as well.

Static Header Logo

One of the first things I do when putting together a new application is "brand" it by incorporating an appropriate logo at the top of the page. This header logo is very simple to achieve and in many cases all the visual customization that is required.

To add a simple, static header image or logo, go to Application Builder à Application à Edit Attributes à Definition à Logo. Enter the full or relative path to the header image, or enter the header text, as depicted in Figure 12. That's it!

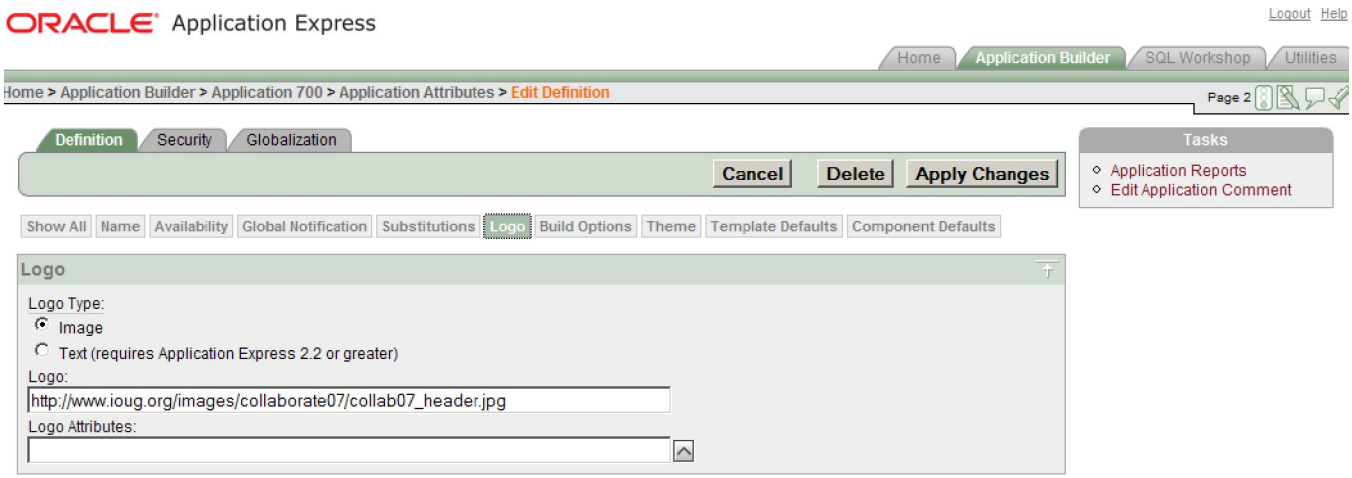

Figure 12 - Application Static Logo Image

Corporate Look-and-Feel

When a complete corporate look-and-feel is required, as when the corporate header and menu system must be applied for all applications, custom theme and page templates are required. Fortunately these are not all that complicated to achieve. The APEX page template is an HTML template with a few APEX-specific tags inserted. In my first real-life venture I was able to take our fishing charter business website template and build an APEX Page Template in under an hour. I am by no means a graphics designer, a CSS expert or a web design wizard. I followed the steps in Scott Spendolini's Cloning your Corporate UI blog series, used a bit of intuition and followed the HTML code until I had a page template that looked like it was right out of FrontPage. Web templates I have replicated since then have progress faster, and I am braver with header and footer menus and including my own template and style sheet customizations.

As of APEX 3.1, one can specify a custom theme as the default theme for the Workspace or the entire APEX instance. This default theme specification is a great convenience in situations where corporate standards must be enforced, or where a common look and feel is desired across web applications.

For more in depth information on how to create a custom page template, see Scott Spendolini's Cloning your Corporate UI with HTML DB blog series at http://www.orablogs.com/scott/archives/001189.html.

Validation

On every form, one expects to have some form of validation. Some is simple stuff: make sure the phone number is complete, make sure the correct format is entered or a date, make sure an email address is real. Other validations may enforce more complex business rules.

In APEX, the column default formats and settings cover the data type (number, character, and date, precision), simple format and required versus optional settings. These become simple APEX validations. APEX automatically creates validations for database column constraints in forms generated by the Create Form wizard. For additional simple validations, JavaScript is an excellent option. For more complex validation, APEX validation types include the use of PL/SQL modules or built-in string comparison methods for validation.

JavaScript Validation

There are many online sources for JavaScript. A browser search for "JavaScript phone number validation", for example, will no doubt yield several versions, one or more of which will suit your needs. The best advice for JavaScript beginners is to search online for what you want, and if you don't find it modify something else to fit.

Some common simple JavaScript validation scripts:

- Force item contents to uppercase:

```
onBlur="javascript:{this.value = this.value.toUpperCase(); }"
```

- Social Security Number format validation:

```
onBlur="javascript:SSNValidation(this.value);"
```

where the SSNValidation function is included (was added to) in the page header JavaScript section:

```
function SSNValidation(ssn) {
  var matchArr = ssn.match(/^(\d{3})-?\d{2}-?\d{4}$/);
  var numDashes = ssn.split('-').length - 1;
  if (matchArr == null || numDashes == 1) {
    alert('Invalid SSN. Must be 9 digits or in the form NNN-NN-NNNN.');
    msg = "does not appear to be valid";
  }
  else
  if (parseInt(matchArr[1],10)==0) {
    alert("Invalid SSN: SSN's can't start with 000.");
    msg = "does not appear to be valid";
  }
  }
```

- Simple Phone Number format validation:

```
onBlur=”javascript:isPhoneNumber(this.value);”
```

where the isPhoneNumber function is included (was added to) in the Page Header JavaScript section:

```
function isPhoneNumber(s)
{
     // Check for correct phone number
     rePhoneNumber = new RegExp(/^\([1-9]\d{2}\)\s?\d{3}\-\d{4}$/);
     if (!rePhoneNumber.test(s)) {
          alert("Phone Number Must Be Entered As: (555) 555-1234");
          return false;
     }
return true;
```

```
}
```

Such simple JavaScript validations – or other more complex JavaScript calls – are entered in the Page Item Element attributes section, in the HTML Form Element Attributes field, as shown in Figure 13:

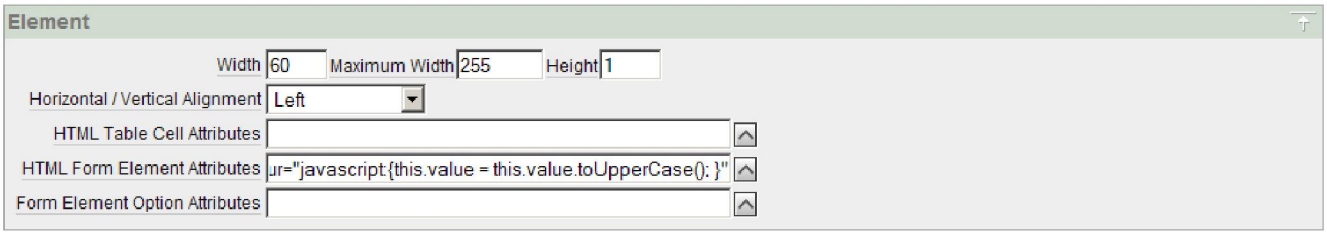

Figure 13 – JavaScript entry in Page Item Element HTML Form Element Attributes

For other JavaScript validations, or for more general JavaScript information for developing your own more complex JavaScript solutions, refer to the APEX JavaScript API's (available with APEX 3.1, and online at http://download.oracle.com/docs/cd/E10513_01/doc/appdev.310/e10499/api.htm#CHDBJJDC) or search online for non-APEX solutions. There are numerous JavaScript code sources, tutorials and forums out there for your learning experience. I found, modified and incorporated a JavaScript format-as-you-type phone number script, which is included as food for thought as Appendix C. The point is that you do not have to be a JavaScript expert to incorporate JavaScript in APEX, you just need to have a bit of incentive and get clever.

APEX Validations

APEX Validations allow a developer to enforce some basic concepts, as in string comparisons, or to build in more complex validation formulas, as may be required by corporate business rules. Check out the APEX validation types, and only write code when the supplied options do not suffice. When one needs to write code, whatever can be accomplished in SQL and PL/SQL can be incorporated in the validation element. The APEX validation interface includes settings for an error message, the location of the error message, and application of a condition.

Validations can be applied to an item or a page. When APEX validation fails, the user is returned to the page, or to the error page. Consider that when using an error page, the user is redirected to the error page as soon as the process hits a validation error. If there are several validation errors, the user will be redirected to the error page several times. If errors are displayed on the current page, all validation errors will be displayed on the current page at the same time, and can be corrected in one pass.

Note that APEX Validations are only applied after a page is submitted, before page processing, to verify the data the user has entered. In contrast, JavaScript can be applied as the user enters or leaves the field.

Reports

Reports pose all kinds of interesting options, because APEX reports are not just reports, they can actually be updateable forms, with embedded links to other URLs or edit pages. In fact, the detail portion of a master-detail form is actually an updateable report. The updateable report concept is quite powerful building block.

The essential tips and tricks for handling reports are covered thoroughly in the APEX How-To's and User's Guide and will not be repeated here. Three key report features to know regarding reports that deserve inclusion on our cheat sheet are interactive reports, how to implement an edit link and how to embed a link in a SELECT statement.

Interactive Reports

APEX version 3.1 introduces interactive reports, which add an entire new dimension to ready-built report functionality. Interactive reports have built-in search-by-column and customizable sort, filter, highlight, computed column and break options. The out-of-the-box functionality is commendable. Figure 14 illustrates the interactive report functions available to the user. A developer can specify which of these features are enabled or disabled, and save the report with default settings.

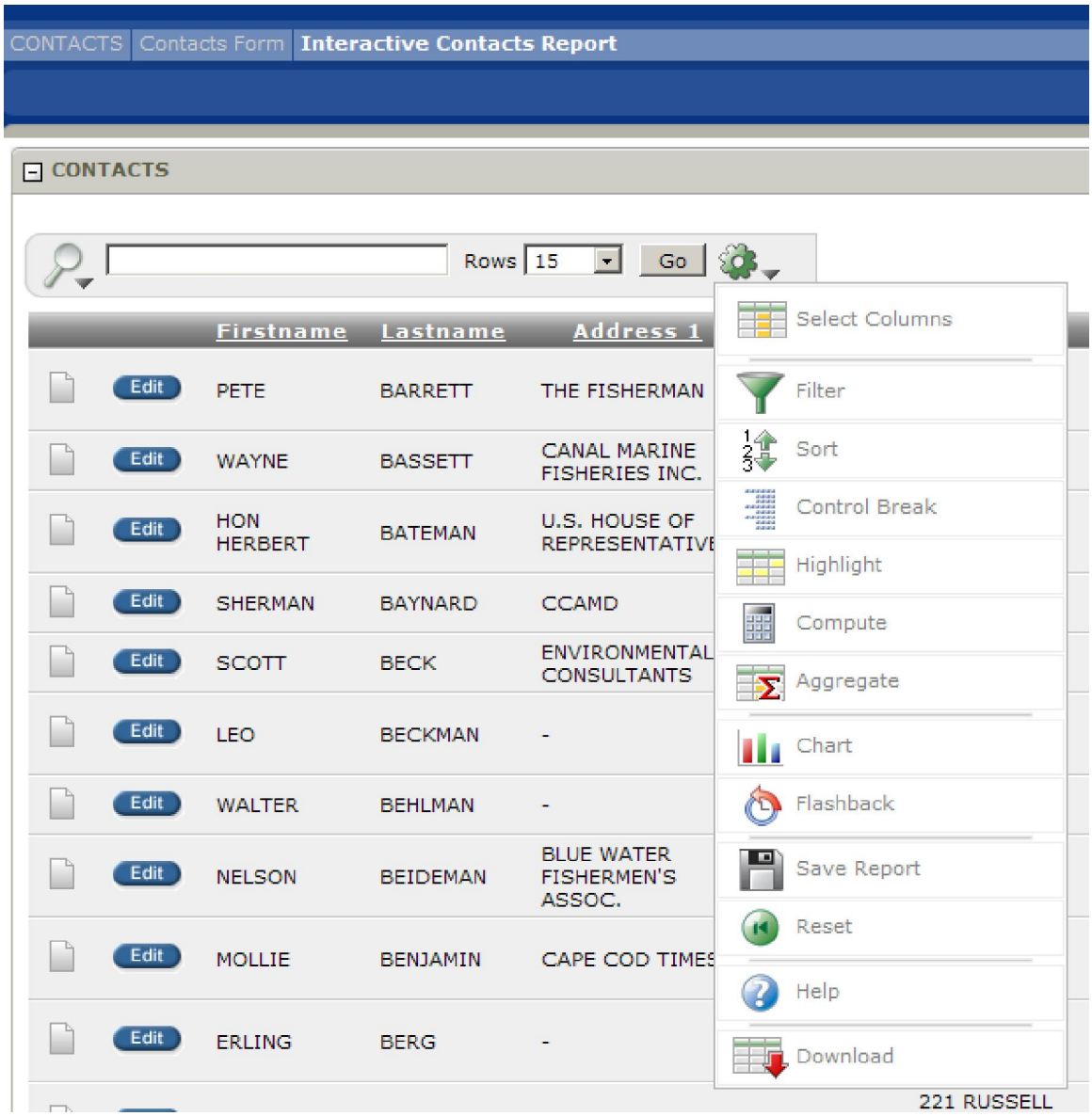

Figure 14 – Interactive Report Menu Options

Existing non-interactive reports can be migrated to interactive reports by using the Migrate to Interactive Report selection in the Tasks menu of the Region Definition of the report.

Interactive reports have a few limitations:

- A single report template. The interactive report template cannot be customized (the HTML cannot be changed); in fact in cannot be viewed. So customized report templates that incorporate conditional displays cannot be applied to interactive reports. One can however add CSS styles to column headers or column data.
- Cannot be updateable. Interactive reports are only reports, they cannot be updateable tabular forms.
- There can be only one interactive report per page

Report Edit Link

The Report Attributes interface of the APEX report offers an optional Column link attribute that enables the column to display as a link as opposed to a standard report column. The link can be a Page in This Application or URL. By now the reader should know URL includes any f?p syntax. The interface also allows for passing item values to the linked page, clearing cache and resetting pagination, all through the column link attributes. The link can be displayed as text, a default icon or any custom image accessible to the application. There is lots of potential here!. Figure 15 displays an example report column link.

Embed Link in Query

This "feature" is nothing more than including a URL – including an f?p syntax URL, in the report SELECT statement. Doing so results in a report that contains a link to that URL – which could be a page in the current application, a different application, or some external URL. The possibilities are endless and interesting.

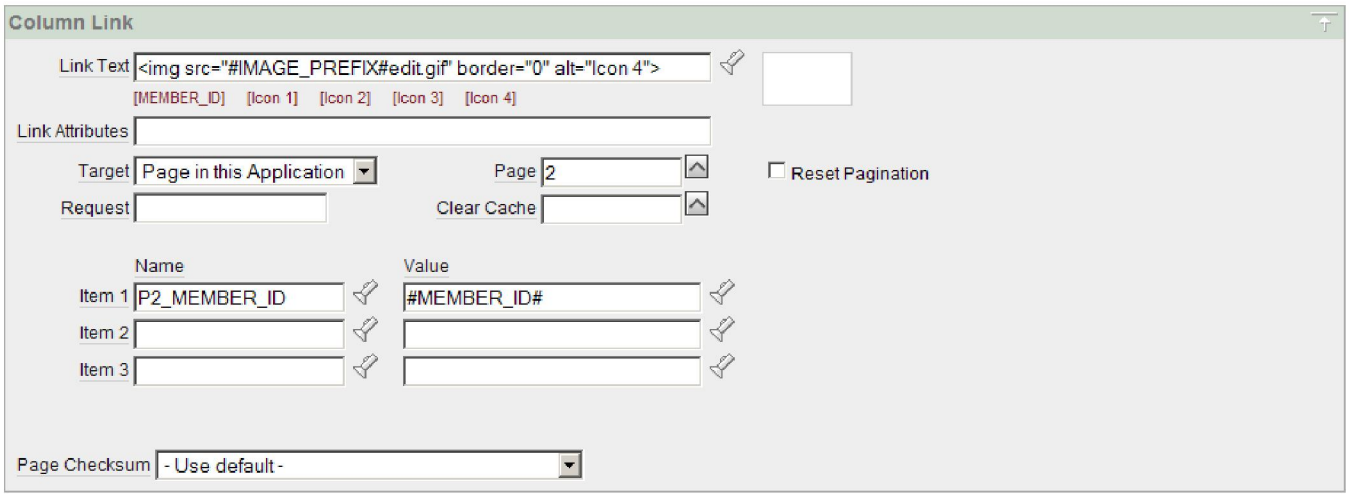

Figure 15 – Report Region, Report Attributes, Column Link Attributes

Popup Options

The most common kind of popup window in APEX is the popup List of Values. The Popup LOV, in several forms, is a default APEX item type. It is not fancy, but does the trick of presenting a selection list in a popup window and returning the return value to the form item. This popup is actually a simple JavaScript popup window generated by APEX with contents based on the list of values attributes of the form item.

When you need a more complex or custom popup window, it is possible to build a separate APEX page and call that page as a popup window through the same JavaScript popupURL function. For example, to call page 46 of the current application and session passing the value of the P44_MEMBERID item to the P46_MEMBERID item in the popup window, use this text in the URL target attribute of a button, or as the URL target of an HTML anchor:

```
javascript:popupURL('f?p=&APP_ID.:46:&SESSION.::NO::P46_MEMBERID:&P44_MEMBERID.');
```

The simplest how-to reference for building a popup window that returns a value back to the calling page is in the How-To section of the OTN APEX main page:

Build Custom Popup Pages -
http://www.oracle.com/technology/products/database/application_express/howtos/how_to_create_custom_popups.html

The basic steps are:

1. Build the calling form page.

2. Build the popup page with fields appropriate to your task.
3. Add JavaScript to call the popup page, passing values to the popup page as needed.
4. Add a popup link or button on the calling page to invoke the popup. For a link, place this HTML in the Post Element Text field attribute of the item to receive the values from the popup. For a button, use the URL Target element attribute field and just the JavaScript portion.

```
<a href="javascript:callMyPopup('P1_ENAME','P1_JOB','P1_SAL');">Click for LOV</a>
```

5. Add JavaScript to the popup page to pass values back to the calling page, as need be.

Since we know the f?p syntax is just an F procedure call to render a page, this popup call syntax is reduced to a simple JavaScript call to open the page we want in a new window. Simple!

Send Email

The ability to send mail from within an APEX process, and to embed a link <back> to an APEX application is a powerful tool. APEX_MAIL is a PL/SQL package for sending mail from within APEX applications. The parameters and syntax are what one would expect for a send mail package. As of APEX 3.1, the APEX_MAIL includes an ADD_ATTACHEMENT procedure

Again, the syntax seems complicated at first glance, but going back to the basics of HTML and the f?p APEX page rendering syntax, the task becomes straightforward. The key things to know are the APEX_MAIL.SEND syntax, how to construct the message body and the PUSH_QUEUE procedure.

Environment Setting – SMTP Mail Server

A prerequisite for sending email from an APEX installation is that the APEX Administrator set the Email SMTP server and port in the Manage Services à Manage Environment Settings, Email interface, as in Figure 16:

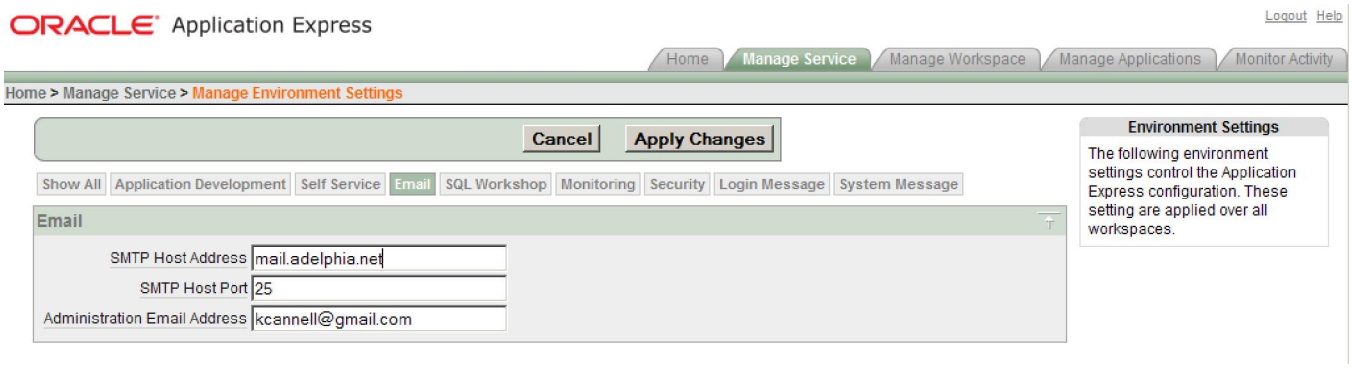

Figure 16 – APEX Manage Environment Settings, Email Server Settings

APEX_MAIL.SEND

The APEX_MAIL.SEND program has all of the usual parameters of a send mail module.

```
APEX_MAIL.SEND (
 p_to IN VARCHAR2,
 p_from IN VARCHAR2,
 p_body IN VARCHAR2 | CLOB,
 p_body_html IN VARCHAR2 | CLOB,
 p_subj IN VARCHAR2 DEFAULT NULL,
 p_cc IN VARCHAR2 DEFAULT NULL,
```

```
 p_bcc IN VARCHAR2 DEFAULT NULL);
```

The call to APEX_MAIL.SEND is made the same as any other PL/SQL procedure call. The parameters may be referenced as strings, bind variables (to reference session state), or PL/SQL variables, as in this example:

```
DECLARE
   v_to   contacts.emailaddress%TYPE;
   v_bcc  VARCHAR2(30) := 'support@ioug.org';
BEGIN

   SELECT emailaddress
     INTO v_to
     FROM contacts
    WHERE member_id = :P2_MEMBER_ID;

   v_body := chr(10)||'     '||chr(10)||'A simple text email message about '
             ||'KALEIDOSCOPE 2008.';

   v_body_html :=
      '<html><body>A simple HTML message about '||
      '<a href="http://www.ioug.org/KALEIDOSCOPE07/>KALEIDOSCOPE 2008</a>.'||
        '<p> A real conference for real technologist using APEX, Oracle Fusion
Middleware, Hyperion and more! </p></body></html>';

   APEX_MAIL.SEND( p_to  => v_to,
                   p_from => 'support@ioug.org',
                   p_subj  => 'KALEIDOSCOPE 2008'
                   p_body => v_body,
                   p_body_html => v_body_html,
                   p_cc  => :P2_EMAILADDRESS,
                   p_bcc  =>  v_bcc);

  APEX_MAIL.PUSH_QUEUE;
END;
```

The p_body and p_body_html parameters may be either VARCHAR2 or CLOB. The two must be the same type, either both VARCHAR2 or both CLOB, but they do not need to have the same content. P_body is for a plain text version of the message body. P_body_html is for an HTML version of the same message, including all HTML tags to form a full HTML document, including the <html> and <body> tags. A single HTML line cannot exceed 1000 characters; one must insert carriage return or line feed characters to split lines longer than 1000 characters. Note the differences in the plain text and HTML body versions in the example above.

PUSH_QUEUE

Mail is sent – pushed – from the APEX mail queue in ten minute intervals. The APEX_MAIL.PUSH_QUEUE procedure forces the mail to be sent immediately. This push step is not necessary in normal operations, but is useful when developing and testing, so you can see the results quickly. The full PUSH_QUEUE syntax is:

```
APEX_MAIL.PUSH_QUEUE( p_smtp_hostname IN VARCHAR2 DEFAULT,
                      P_smtp_portno   IN NUMBER DEFAULT)
```

where p_smtp_hostname and p_smtp_portno are the hostname and port of the SMTP server set by the APEX Administrator in the APEX environment configuration.

Embedded Link to APEX Page

To embed a link to an APEX page in an email message, build the f?p syntax pointing to the application and page into the body of the email. Use the f?p syntax as the href target in an HTML anchor tag.

```
DECLARE
  l_body_html varchar2(4000);
  lv_tolist varchar2( 2000) := 'joedba@yourcompany.com';
BEGIN
  l_body_html := '<p> '||
   'To view matching KALEIDOSCOPE Contacts click the link: </p>'||
    '<a href="http://tunahuntress.tunahunter.com:7777'||
    '/f?p=550:33:::::'||'P33_REPORT_SEARCH:'||:P1_LAST||
    '">View KALEIDOSCOPE Contacts</a></p>';
  APEX_MAIL.SEND(
    P_TO        => lv_tolist,
    P_FROM      => :P1_EMAIL,
    P_BODY      => l_body_html,
    P_BODY_HTML => l_body_html,
    P_SUBJ      => 'New Mail Message from APEX';
END;
```

When referring to an APEX application page from email, we don't know the session identifier, so leave that blank. Do include the application id, the page number (in this case hard coded) and any item names and values that may apply. Note the use of bind variables in constructing this f?p string to set the P33_REPORT_SEARCH item with the value of the current application's P1_FIRST item. When the user clicks on the APEX application link, they will be asked to log in to APEX, because application 550 requires authentication.

Attachments

With APEX 3.1 APEX_MAIL does support attachments with the APEX_MAIL.ADD_ATTACHMENT procedure. This procedure sends an outbound email message from an application as an attachment. Use multiple calls to ADD_ATTACHMENT to add multiple attachments to a single email.

```
APEX_MAIL.ADD_ATTACHMENT(
    p_mail_id                   IN    NUMBER,
    p_attachment                IN    BLOB,
    p_filename                  IN    VARCHAR2,
    p_mime_type                 IN    VARCHAR2);
```

Table 3 ADD_ATTACHMENT Parameters

Parameter	Description
p_mail_id	The numeric ID associated with the email. This is the numeric identifier returned from the call to APEX_MAIL.SEND to compose the e-mail body.
p_attachment	A BLOB variable containing the binary content to be attached to the e-mail message.
p_filename	The filename associated with the e-mail attachment.
p_mime_type	A valid MIME type (or Internet media type) to associate with the e-mail attachment.

In versions prior to APEX 3.1, one can use UTL_SMTP or Java stored procedures called from PL/SQL to send mail attachments. See the OTN APEX Forum thread regarding email attachments at http://forums.oracle.com/forums/thread.jspa?messageID=1383936 for more information.

Custom Authentication

APEX has three standard types of authentication: Database, APEX – all users logging in must be APEX users, and None, meaning the application is "Public", anyone can access the application. Chances are you want to use something a bit more complicated.

Custom Authentication is actually simple to implement, but the first time through the wizard can be confusing. Following the Custom Authentication Wizard: The steps to create your own basic, custom authentication scheme are:

From the Shared Components, Authentication Schemes flow, click on the Create button to start the Create Custom Authentication Schema Wizard.

Step 1 - Select the From Scratch radio button to begin creation of an authentication scheme from scratch, as in Figure 17.

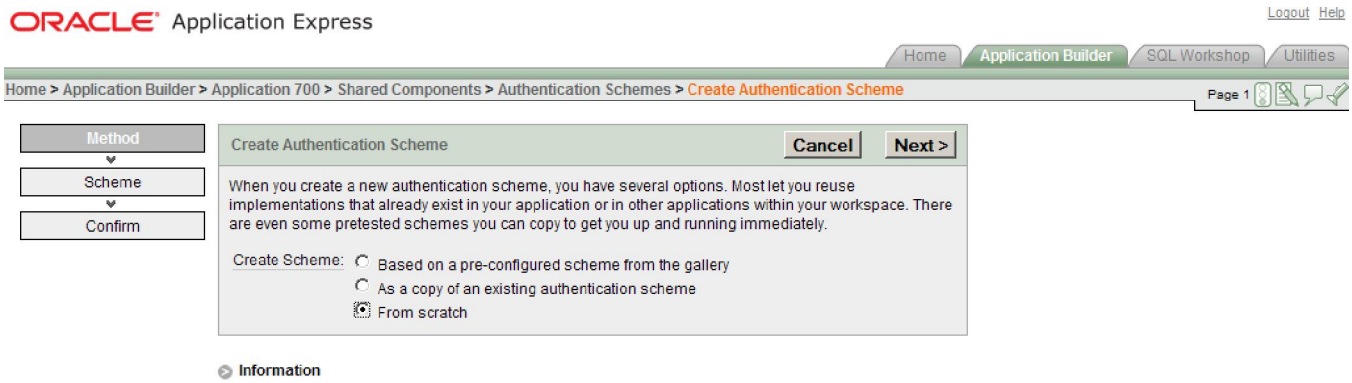

Figure 17 – Custom Authentication Method

In Step 2, name the scheme and provide a description, as displayed in Figure 18.

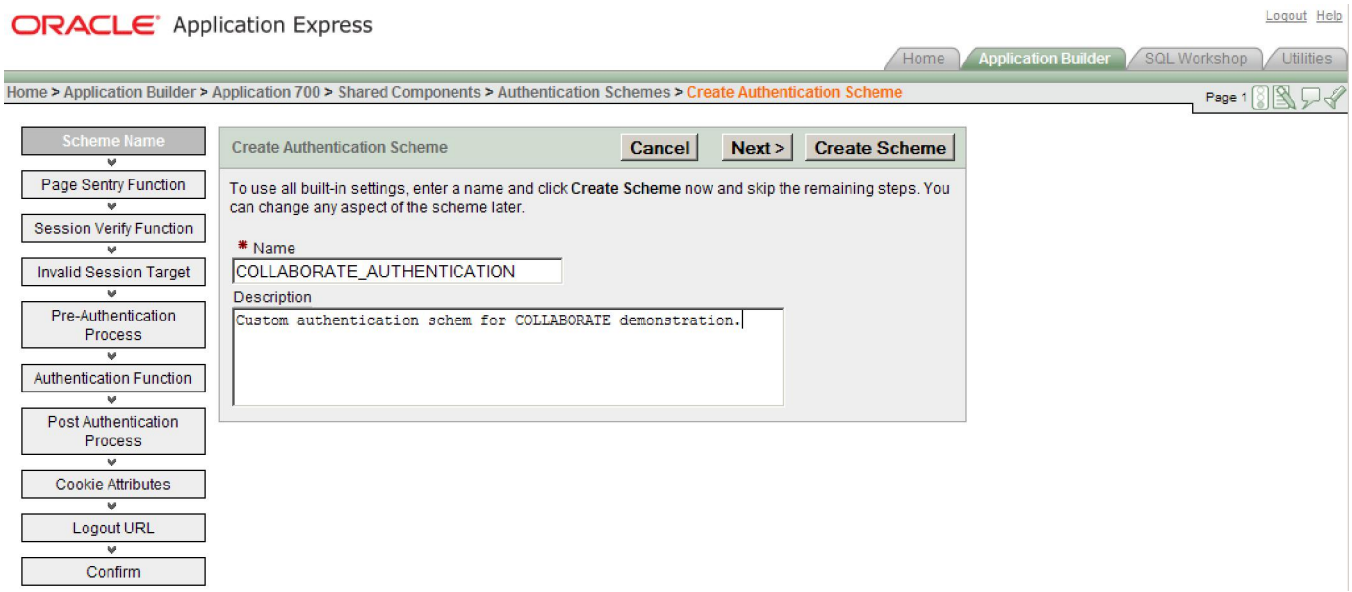

Figure 18 – Custom Authentication Scheme Name

For Steps 3, Sentry Function, and Step 4, Session Verify Function, it is recommended that one accept the default functionality. The Sentry Function checks before each page whether to proceed or redirect elsewhere. The Session Verify Function checks for a valid session. Use the default here, unless you need to set multiple cookies.

Step 5, Invalid Session Target - Specify where to go when a login attempt fails. This can be the login page, a custom page within this application, a URL, the Single Single-On login page, or none. This wizard step is illustrated in Figure 19.

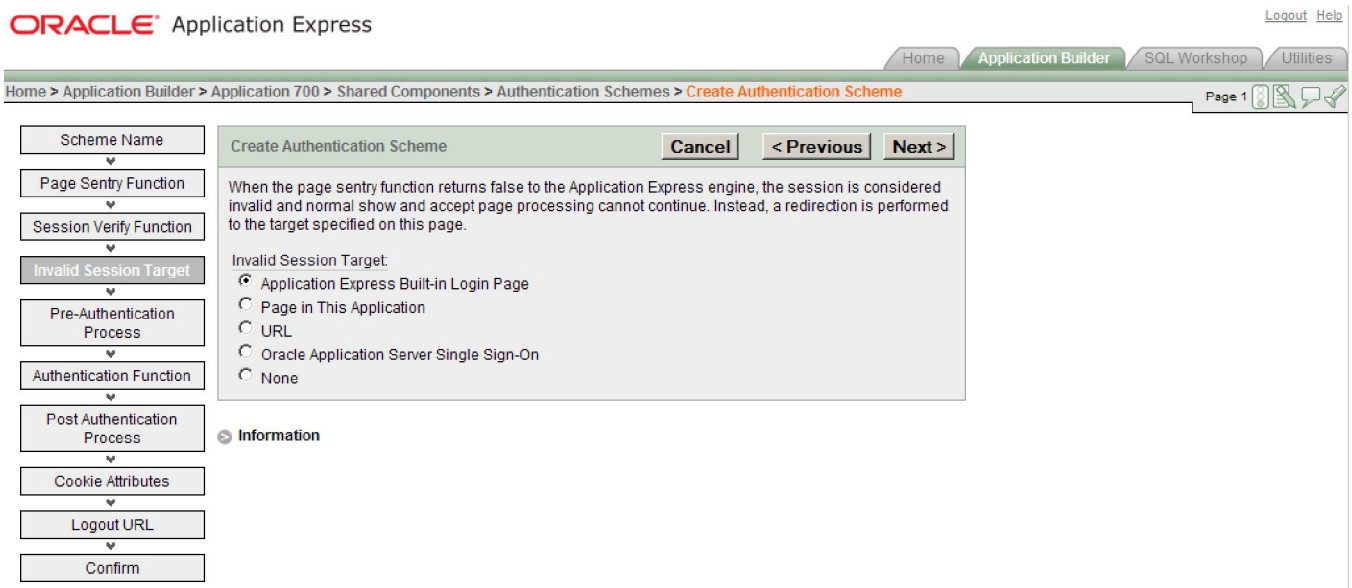

Figure 19 – Custom Authentication Scheme Invalid Session Target

Step 6, Pre-Authentication Process - Enter any code that needs to be executed before authentication here, such as setting cookies or creating audit records.

Step 7, Authentication Process - *This* is where the call to your authentication function goes. Check Use my custom function to authenticate, then enter your authentication function in the region. Your authentication function must have two parameters, username and password, and return a Boolean. However, **do NOT enter the username and password parameters in the authentication function element**. The username and password parameters are automatically inferred by the APEX engine. Use of a custom authentication function allows you to use your own user table, and not have to create an APEX user account for each of your application users.

- You can accept the defaults in most steps, enter your authentication function (minus the username and password parameters) in the Authentication Process step.
- Your authentication function must have the username and password parameters, whether or not it requires them as input.

Step 8, Post Authentication Process - Enter code to be executed after authentication, such as setting cookies or creating audit records. This step is optional.

Step 9, Cookie Attributes - Enter cookie attributes you will need in your application. This step is optional. The APEX default session cookie will still be created if you enter nothing in this step.

You're done!

The cheat sheet recap of the Custom Authentication process is:

- You can accept the defaults in most steps, enter your authentication function (minus the username and password parameters) in the Authentication Process step.
- Your authentication function must have the username and password parameters, whether or not it requires them as input.

While this custom authentication example is very simple, the custom authentication framework allows for incorporating role-based, java security-based authentication, or authentication against an Oracle e-Business Suite User Repository. See the APEX Studio site for more APEX custom authentication examples at http://htmldb.oracle.com/pls/otn/f?p=18326:1:::NO:::

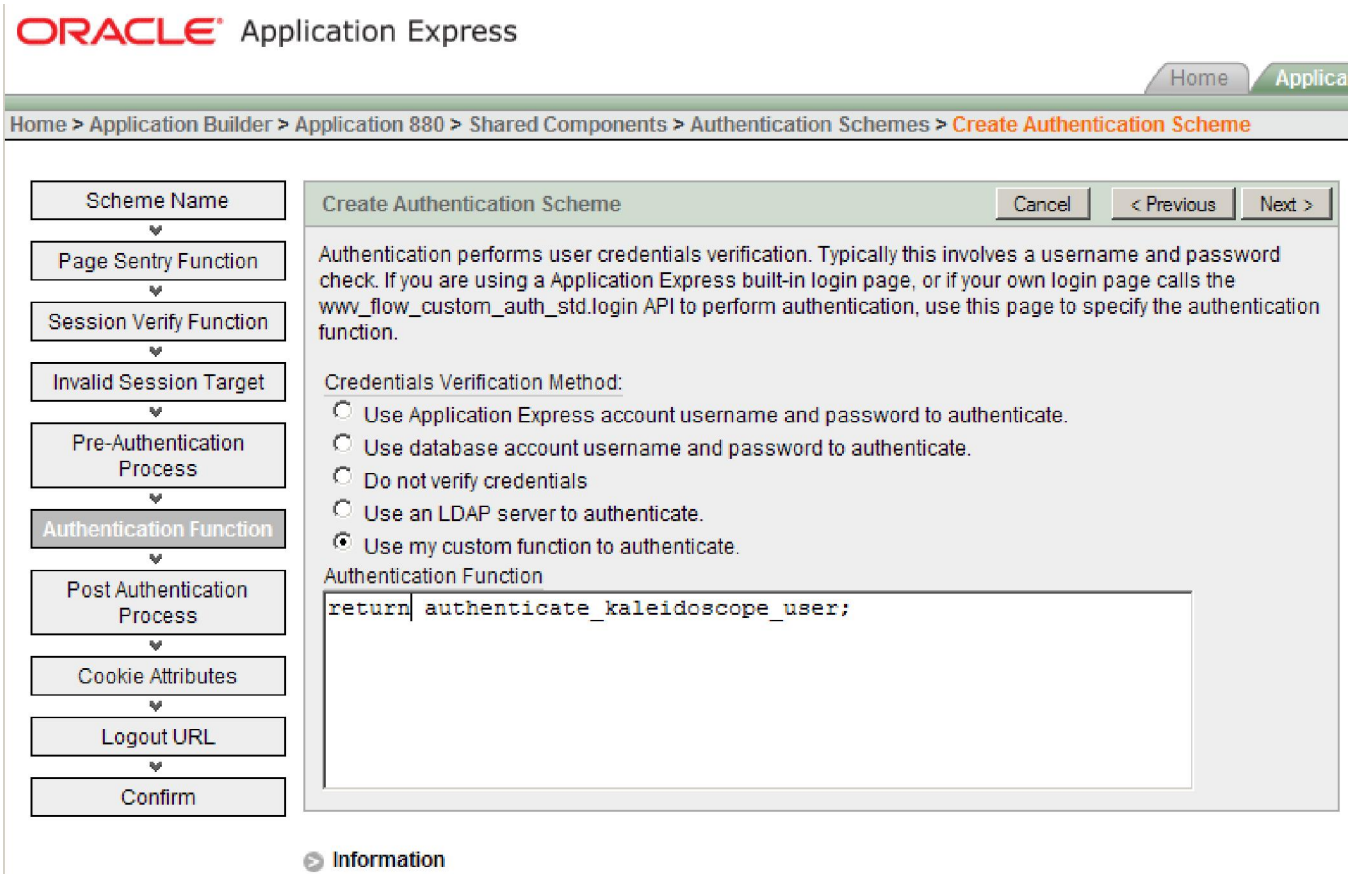

Figure20 – Custom Authentication Scheme Authentication Function

When All Else Fails … AJAX

There are times when one needs to step outside the usual HTML-JavaScript-full-page rendering pattern and incorporate AJAX to achieve the desired functionality. AJAX stands for Asynchronous JavaScript using XML. AJAX allows for dynamic refresh of a portion of your page, as opposed to regenerating the whole page. This is useful for example, as a workaround to the Select List limitation of 32K bytes. (Note that the 32K limitation on some item types is lifted in APEX version 3.1). When there is no other workaround, AJAX may be just the thing.

Before embarking of custom coded AJAX, see in the APEX 3.1 new features of

A detailed discussion of how to incorporate AJAX in your APEX application is beyond the scope of this article. To attempt a thirty-second AJAX lesson would do more harm than help. However, our APEX cheat sheet would not be complete without references to the best sources for complete information on incorporating AJAX into APEX. The most helpful articles on AJAX in APEX applications are:

Putting the Express Back Into Oracle Application Express with AJAX, Steve Karam, Burleson Consulting and Training, http://www.oraclealchemist.com/wp-content/uploads/2006/11/s281946.pdf.

AJAX Select List Code Generator, Scott Spendolini, Sumner Technologies, LLC, http://spendolini.blogspot.com/2005/10/ajax-select-list-code-generator.html.

There are also several free, online AJAX courses, for those who want a more complete lesson in AJAX outside of the APEX environment.

Note that AJAX introduces a new level of complexity to your code. If you do introduce AJAX, document your AJAX code thoroughly, for those that follow you, and for yourself when you go back to make a change months later.

Save Often

Some concepts are just too basic for even the cheat sheet. With APEX, a seeming innocuous change sometimes has a dramatic effect – so you need to proactive by saving often and knowing how to recover to the application as of a few minutes ago..

SAVE OFTEN – Get used to the **Apply Changes** button. Apply Changes *before* you navigate away from a page. This avoids having to redo the same changes.

- Export periodically. When all else fails, you can restore an application by importing the most recent export. This is nothing more that good programming practice. Backup you work as you go.

The Final Cheat Sheet

That's it – I cannot fit any more in today's cheat sheet article. The truth is I cannot fit all I need to know about APEX in one reference, and as I do more with APEX, my cheat sheet of essential features shifts and expands. It has evolved to a long, dynamic list of bookmarks, blogs and feeds. My hope is that some of the above has been useful as a foundation for developing one's own APEX toolbox. Maybe next year they'll let us fit all we want on a flash drive!

APEX References and Help Sources

Given that it is not possible to know it all, save it all on one page, or present it all in one article, it is important to know where to turn for references and additional APEX information. The APEX user community is active, helpful, responsive and growing. It is also useful to invest in learning JavaScript, HTML and CSS, as these technologies are integral to how APEX application render and display. The more one knows in these areas, the more flexibility one has in APEX development.

The following is a list of online and printed, Oracle and non-Oracle APEX resources. Use these resources for continual help on all levels. There will come a time when you have to think and act outside the wizard. Be creative, and go for it. There is plenty of APEX support out there.

Online APEX Resources

The APEX OTN Forum - http://forums.oracle.com/forums/forum.jspa?forumID=137 . The APEX user community is active and growing. It is your best source for APEX how-to's and answers for users of all abilities. Know it, Bookmark it. Use it. Contribute to it.

The APEX Studio - http://htmldb.oracle.com/pls/otn/f?p=18326:1:3795991895340045::NO::: As with JavaScript, there are a growing number of APEX applications out there that are free to download and use as is or modify for your own application. This is the Oracle-hosted APEX hosting site. It includes applications, utilities, themes, tips and tricks. Watch for early adopter versions of APEX.

The APEX OTN Technology Center – http://www.oracle.com/technology/products/database/application_express/index.html The Oracle Technology Network Application Express home page.

APEX on SourceForge.net - http://apex.oracle.com/pls/otn/f?p=apexsf. The Open APEX Application Development project. An open source site for sharing APEX applications.

Oracle MetaLink - APEX is a fully supported Oracle product, part of the 10g database and higher. As such, support through Metalink is possible. This author's experience is that help through the APEX forum is faster.

APEX-Related Blogs

Scott Spendolini: http://spendolini.blogspot.com and the older http://blogs.oracle/com/scott

Carl Backstrom: http://carlback.blogspot.com

Patrick Wolf – Inside APEX. http://inside-apex.blogspot.com and the ApexLib add-on at http://apexlib.sourceforge.net

APEX Blog Aggregator: http://www.apexblogs.info A collection of APEX-related blog material, compiled from a variety of APEX documentation, examples pages and blogs by APEX experts.

Textbooks

Oracle HTMLDB Handbook, *Develop and Deploy Fast, Secure Web Applications*, by Bradley Brown and Lawrence Linnemeyer, McGraw Hill/Oracle Press, 2006, $49.99, 486 pages.

Easy Oracle HTML-DB, *Create Dynamic Web Pages with Oracle*, by Michael Cunningham and Kent Crotty, Rampant Press, 2006, $39.95, 400 pages.

Pro Application Express, by John Scott, Scott Spendolini, APress, To be published soon in 2008.

About the Author

Karen Cannell is a Principal with Integra Technology Consulting and editor of the ODTUG Technical Journal. She has analyzed, designed developed, converted, enhanced and otherwise tweaked database applications for over 22 years, focused on Oracle since 1994. Recent focus includes design and development of ETL processes, Application Express, Oracle Business Intelligence and migration of legacy apps to web technologies, leveraging the Oracle 10g/11g suite of tools. She can be reached at kcannell@integratc.com.

Appendix A – APEX Built-In Substitution Strings

This entire section is an excerpt from the Oracle Database Application Express User's Guide, the Application Builder Concepts section. Refer to the Oracle APEX online technology center for the latest APEX User's Guide and substitution string information.

About Built-in Substitution Strings

Application Builder supports a number of built-in substitution strings. You may need to reference these values to achieve specific types of functionality.

The following sections describe these substitution strings; when to use them, and what supported syntax is currently available. Note that bind variable `:USER` has special meaning within the database. Also, the term Direct PL/SQL refers to PL/SQL that can be used in stored database objects such as procedures and functions.

Topics:

- DEBUG
- HOME_LINK
- LOGIN_URL
- IMAGE_PREFIX
- Application Express SCHEMA OWNER
- PRINTER_FRIENDLY
- LOGOUT_URL
- PROXY_SERVER
- PUBLIC_URL_PREFIX
- REQUEST
- SQLERRM
- SYSDATE_YYYYMMDD
- WORKSPACE_IMAGES

See Also:

- "Substitutions" for information about defining static substitution strings as an application attribute
- "Establishing User Identity Through Authentication" for information about authentication

Appendix B: JavaScript Sample – Format-As-You-Type Phone Number

The JavaScript will format a phone number as the user types the number in, adding open and close brackets and dash (,) and - characters. The main idea and bulk of this script was found online. Modifications were made to suit the case at hand so it would work in our application. It is included here as an example of what can be done with a bit of online research and incentive. There are no doubt many more clever and efficient ways to achieve the same result!

The following code goes in the Page Header region, or must be included in a referenced include file.

To call the JavaScript, place a call on the page item in the HTML Form Element Attributes field to the backspacerUP and backspacerDOWN functions.

```
var zChar = new Array(' ', '(', ')', '-', '.');
var maxphonelength = 13;
var phonevalue1;
var phonevalue2;
var cursorposition;

function ParseForNumber1(object){
      phonevalue1 = ParseChar(object.value, zChar);
}
function ParseForNumber2(object){
      phonevalue2 = ParseChar(object.value, zChar);
}
function backspacerUP(object,e) {
      if(e){
            e = e
      } else {
            e = window.event
      }
      if(e.which){
            var keycode = e.which
      } else {
            var keycode = e.keyCode
      }

      ParseForNumber1(object)

      if(keycode > 48){
            ValidatePhone(object)
      }
}
function backspacerDOWN(object,e) {
      if(e){
            e = e
      } else {
            e = window.event
      }
      if(e.which){
            var keycode = e.which
      } else {
            var keycode = e.keyCode
      }
      ParseForNumber2(object)
}

function GetCursorPosition(){

      var t1 = phonevalue1;
```

```
      var t2 = phonevalue2;
      var bool = false
    for (i=0; i<t1.length; i++)
    {
      if (t1.substring(i,1) != t2.substring(i,1)) {
            if(!bool) {
                  cursorposition=i
                  bool=true
            }
      }
    }
}
function ValidatePhone(object){

      var p = phonevalue1

      p = p.replace(/[^\d]*/gi,"")

      if (p.length < 3) {
            object.value=p
      } else if(p.length==3){
            pp=p;
            d4=p.indexOf('(')
            d5=p.indexOf(')')
            if(d4==-1){
                  pp="("+pp;
            }
            if(d5==-1){
                  pp=pp+")";
            }
            object.value = pp;
      } else if(p.length>3 && p.length < 7){
            p ="(" + p;
            l30=p.length;
            p30=p.substring(0,4);
            p30=p30+")"

            p31=p.substring(4,l30);
            pp=p30+p31;

            object.value = pp;

      } else if(p.length >= 7){
            p ="(" + p;
            l30=p.length;
            p30=p.substring(0,4);
            p30=p30+")"

            p31=p.substring(4,l30);
            pp=p30+p31;

            l40 = pp.length;
            p40 = pp.substring(0,8);
            p40 = p40 + "-"

            p41 = pp.substring(8,l40);
            ppp = p40 + p41;

            object.value = ppp.substring(0, maxphonelength);
      }
```

```
      GetCursorPosition()
      if(cursorposition >= 0){
            if (cursorposition == 0) {
                  cursorposition = 2
            } else if (cursorposition <= 2) {
                  cursorposition = cursorposition + 1
            } else if (cursorposition <= 5) {
                  cursorposition = cursorposition + 2
            } else if (cursorposition == 6) {
                  cursorposition = cursorposition + 2
            } else if (cursorposition == 7) {
                  cursorposition = cursorposition + 4
                  e1=object.value.indexOf(')')
                  e2=object.value.indexOf('-')
                  if (e1>-1 && e2>-1){
                        if (e2-e1 == 4) {
                              cursorposition = cursorposition - 1
                        }
                  }
            } else if (cursorposition < 11) {
                  cursorposition = cursorposition + 3
            } else if (cursorposition == 11) {
                  cursorposition = cursorposition + 1
            } else if (cursorposition >= 12) {
                  cursorposition = cursorposition
            }

        var txtRange = object.createTextRange();
        txtRange.moveStart( "character", cursorposition);
            txtRange.moveEnd( "character", cursorposition - object.value.length);
        txtRange.select();
    }

}
function ParseChar(sStr, sChar)
{
    if (sChar.length == null)
    {
        zChar = new Array(sChar);
    }
    else zChar = sChar;

    for (i=0; i<zChar.length; i++)
    {
        sNewStr = "";

        var iStart = 0;
        var iEnd = sStr.indexOf(sChar[i]);

        while (iEnd != -1)
        {
            sNewStr += sStr.substring(iStart, iEnd);
            iStart = iEnd + 1;
            iEnd = sStr.indexOf(sChar[i], iStart);
        }
        sNewStr += sStr.substring(sStr.lastIndexOf(sChar[i]) + 1, sStr.length);

        sStr = sNewStr;
    }
```

```
    return sNewStr;
}
```