SUNGARD SCT HIGHER EDUCATION

SCT Banner Technical Training
Web Programming Training Workbook

May 2005
Release 7.1

Prepared By: SunGard SCT
4 Country View Road
Malvern, Pennsylvania 19355
United States of America

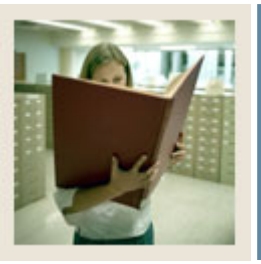


Table of Contents

Table of Contents (Continued)

Section A: Introduction

Lesson: Overview

Jump to TOC

Workbook goal

The purpose of the class is to teach participants how to modify, customize and program Self Service Banner (SSB) products.

Intended audience

Programmers, DBA's, and analysts who may teach others about SCT Banner tables and processes will benefit from the training.

To complete this workbook, you should have

- completed the SCT Education Practices computer-based training (CBT) tutorial "Banner 7 Fundamentals" or have equivalent experience navigating in the Banner system
- basic understanding of HTML tags, structures and syntax
- basic understanding of PL/SQL block structures, procedures and functions
- access to a Self Service Banner as a Web Tailor user
- a UNIX account on the database server with appropriate access to a training database OR SQLPLUS access to the training database.

Objectives

At the end of this course, participants will be able to:

- develop their own Self Service Banner applications by creating custom PLSQL programs
- integrate their applications with SCT Banner, utilizing the SCT Banner Web product libraries
- understand the role of the Oracle 9i Application Server in Self Service Banner
- enhance their applications using JavaScript and Style Sheets.

Client responsibilities

The client must complete several tasks before the training consultant arrives at the site to conduct the course.

Before you set up the current module, the following elements must be defined:

- Create training accounts in the training database.
- Under each account grant: create public synonym, and execute any procedure
- Under each account run the CT.sql script (provided by instructor) to create the training tables

If you are unable to complete these tasks before the course is scheduled, please contact your account consultant for assistance.

Section A: Introduction

Lesson: Overview (Continued)

Jump to TOC

Section contents

Section A: Introduction

Lesson: Introduction

Jump to TOC

Introduction

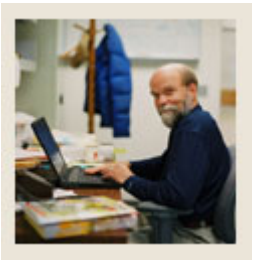
The purpose of the class is to teach participants how to modify, customize and program Self Service Banner (SSB) products.

Topics include:

- Why web-based application development
- Oracle 9ias architecture and components responsible for running SSB: Listener, DAD, PLSQL Gateway, Security
- HTML review
- PLSQL review
- Oracle PLSQL toolkit development suite
- Securing your applications
- The Banner Web Libraries of Web Tailor, Banner Web Security features and Product Standards

What is Self Service Banner?

- Add-on to Banner baseline products
- Advantages of Banner Web Design
    - Reduced network processing of data servers
    - Move from fat client to thin client on desktop
    - No middle tier forms, logic maintained on database server
- Uses Oracle PL/SQL Packages on Banner database
    - Static html pages, CSS and image files on Web Server



Section B: Oracle 9iAS Architecture

Lesson: Overview

Jump to TOC

Introduction

In this section, we discuss Oracle 9i Application Server as it relates to Self Service Banner (SSB).

Section contents

Section B: Oracle 9iAS Architecture

Lesson: Tiers

Jump to TOC

Tiers

SSB architecture has three tiers:

- **Web Browser**
  SSB is designed to work best with a browser which supports Java Scripting and Style Sheets
- **Oracle 9i Application Server**
  SSB only requires Oracle 9ias Core Edition and the modplsql component
- **Oracle Database Server**
  SSB requires the installation of Oracle's PLSQL toolkit. By default, the Oracle PLSQL toolkit is installed into the SYS schema

Section B: Oracle 9iAS Architecture

Lesson: Details

Jump to TOC

Configuration files

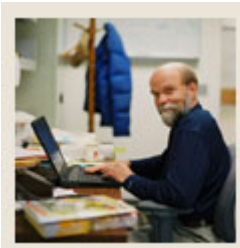
Path	Description
$ORACLE_HOME/Apache/Apache/conf/httpd.conf	Master Apache Listener file
$ORACLE_HOME/Apache/Apache/conf/oracle_apache.conf	Oracle specific Information
$ORACLE_HOME/Apache/modplsql/cfg/plsql.conf	PLSQL gateway configuration file
$ORACLE_HOME/Apache/modplsql/cfg/wdbsvr.app	DAD configuration file

Web Listener

- Listens for requests on pre-defined ports (use any higher than 1000)
- For maintenance, best to have separate listeners for each database
- Allows systems to be brought up and down independently
- Allows SSL to be used for web
- Defined by one file configuration file on 9ias
- $ORACLE_HOME/Apache/Apache/conf/httpd.conf

Data Access Descriptor

- Locate the remote database
- Establish a SQL*Net connection
- Login to the remote database as oracle user
- Execute requested package.procedure
- Define in flat file: ORACLE_HOME/Apache/modplsql/cfg/wdbsvr.app
- DADs can be modified, removed and added via web page
    - http://server.domain:port/pls/admin_/gateway.htm



Example httpd.conf File

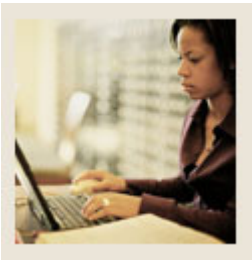
```
Port  9000
Listen 9000
<VirtualHost _default_:9000>
# Document Root the directory where static files are pulled for
banner web
DocumentRoot d:\sct\banweb
# Directory Index is the default page for the listener
DirectoryIndex trng.HTM
</VirtualHost>
```

Example plsql.conf File

```
<Location /banweb>
   SetHandler pls_handler
   Order deny,allow
   Allow from all
</Location>

 <LocationMatch "banweb(.*)admin_">
   AuthType Basic
   AuthName "Restricted Access"
AuthUserFile
/u01/app/oracle/product/ias1022/Apache/modplsql/cfg/userfile
AuthGroupFile
/u01/app/oracle/product/ias1022/Apache/modplsql/cfg/groupfile
require group webadmin
</LocationMatch>
```

Browser Requests

- The Oracle HTTP Server receives a PL/SQL Server Page request from a client browser
- The Oracle HTTP Server routes the request to the PL/SQL Gateway
- The request is forwarded by the PL/SQL Gateway to the Oracle9i Database. By using the configuration information stored in your DAD, the PL/SQL Gateway connects to the database
- The PL/SQL Gateway prepares the call parameters, and invokes the PL/SQL procedure in the application
- The PL/SQL procedure generates an HTML page using data and the PL/SQL Web Toolkit accessed from the database
- The response is returned to the PL/SQL Gateway
- The Oracle HTTP Server sends the response to the client browser



Section C: HTML Review

Lesson: Overview

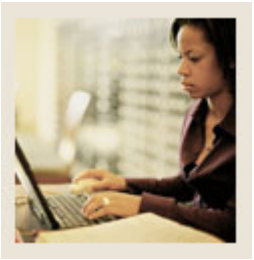
Jump to TOC

Introduction

In this section, we will discuss the use of HTML to create web applications.

- Basic HTML to create documents and format text
- Advanced HTML to create tables to format data and forms to process data

Section contents



Section C: HTML Review

Lesson: Details

Jump to TOC

Hyper Text Markup Language (HTML)

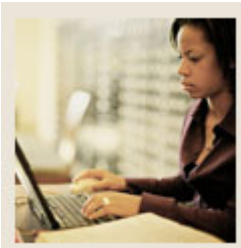
- language of the web
- instructs browser what to do
- series of tags in an ASCII/text document
- tags travel in pairs: **<bold>** hi **</bold>**
  with what you want to affect in between tags
- you can always 'view source' in browser

What an HTML Document Is

HTML documents are plain-text (also known as ASCII) files that can be created using any text editor (e.g., Emacs or vi on UNIX machines; SimpleText on a Macintosh; Notepad on a Windows machine). You can also use word-processing software if you remember to save your document as "text only with line breaks".

Sample HTML Document

```
<html>
<!-- comments go here -->
<head>
<title> Your Title appears at the very top of window bar </title>
</head>
<body>
<h1> Hello World </h1>
<p>This is our first web document!
</body>
</html>
```



Section C: HTML Review

Lesson: Details (Continued)

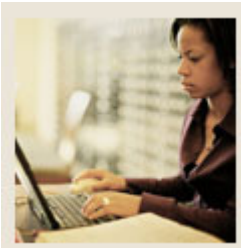
Jump to TOC

Basic HTML Tags

HTML tag	Purpose
`<html> </html>`	creates an html document
`<head> </head>`	info about the html document
`<title> </title>`	displays info in window title bar
`<body> </body>` `<body bgcolor=? text=? link=? vlink=? alink=?>`	visible portion of document

`<h1> </h1>`
`<h2> </h2>`
`<h3> </h3>`
`<h4> </h4>`
`<h5> </h5>`
`<h6> </h6>`
	header or headline

`<p> </p>`	paragraph
`<b> </b>`	bold
`<i> </i>`	italics
` `	inserts a line break
`<pre> </pre>`	preformatted text
`<font> </font>` `<font size=? color=?>`	set font attributes



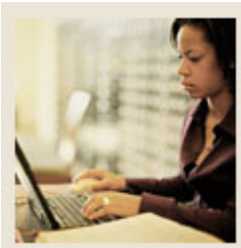
Section C: HTML Review

Lesson: Details (Continued)

Jump to TOC

Advanced HTML tags

HTML tag	Purpose
<img> </img> <img src="file?" Align=? Border=?>	image (picture) source
<a href> <a href="URL"> <a href="mailto:EMAIL">	hyperlink reference
<table> </table>	creates a table of data
<th> </th>	table header
<td> </td>	table data
<tr> </tr>	table row (where does it end)
<form> </form> <option> <input type="submit" value="CGI name"> <input type="reset"> <select multiple name="?" size=?> </select> <input type="radio" name="?" value="?"> <input type="checkbox" name="?"> <input type="text" name="?" size=?> <textarea name="?" cols=? Rows=?> </textarea>	creates a form to call a CGI (process)



Section C: HTML Review

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self-check activity.

Exercise 1

In this exercise, you will experiment with some basic components of HTML.

Step	Action
1	Use a text editor and create an HTML file.
2	Play with different tags we've seen (bold, italics, table & form).
3	View an HTML file with a Web browser.

Section D: PL/SQL Review

Lesson: Overview

Jump to TOC

Introduction

PL/SQL is the language of the Oracle database. All of the Banner Self Service logic is stored and processed PL/SQL program unit called packages and procedures. In order to complete the remaining exercises, the participant will need a strong knowledge of PL/SQL structure, code and syntax.

The following discussion is intended as review of basic PL/SQL coding technique. Participants should reference a PL/SQL manual for detailed explanations of object declarations and control structure.

Section contents

Section D: PL/SQL Review

Lesson: Procedural Language/Standard Query Language

Jump to TOC

Components

- Basic
- Variables
- Control Structures
- If-then-else, for-loop, while-loop, exit-when and goto
- Cursor & cursor FOR loops
- Modern
- Modularity; subprograms & packages
- Information Hiding & Data Encapsulation
- Overloading
- Exception Handling

PL/SQL Package Syntax

```
PACKAGE package_name
IS
  <declaration of public variables>
  <declaration of public cursors>
  <declaration of public functions & procedures>
END package_name;
PACKAGE BODY package_name
IS
  <declaration of public variables - again!>
  <declaration of public cursors - with SELECT>
  <declaration of public functions & procedures - BODY>
BEGIN
  ...
EXCEPTION
  ...
END package_name;
```

Section D: PL/SQL Review

Lesson: Procedural Language/Standard Query Language (Continued)

Jump to TOC

PL/SQL Procedure Syntax

```
CREATE OR REPLACE   PROCEDURE name [ (parameter list) ]
IS   <declaration section>
BEGIN
  <execution section>
EXCEPTION
END name;
```

PL/SQL Function Syntax

```
CREATE OR REPLACE
FUNCTION name [ (parameter list) ] RETURN return_datatype
IS
  <declaration section>
BEGIN
  <execution section>
EXCEPTION
  <exception section>
END name;
```

Cursor Example

```
DECLARE
    CURSOR emp_cursor(dnum NUMBER) IS
        SELECT sal, comm FROM emp WHERE deptno = dnum;
    total_wages   NUMBER(11,2) := 0;
    higher_comm   NUMBER(4) := 0;
BEGIN
    /* The number of iterations will equal the number *
     * of rows returned by emp_cursor.               */
    FOR emp_record IN emp_cursor(20) LOOP
        emp_record.comm := NVL(emp_record.comm, 0);
        total_wages := total_wages + emp_record.sal +
        emp_record.comm;
        IF emp_record.comm > emp_record.sal THEN
            higher_comm := higher_comm + 1;
        END IF;
    END LOOP;
    INSERT INTO temp VALUES (higher_comm,
        'Total Wages: ' || TO_CHAR(total_wages));
    COMMIT;
END;
/
```

Section D: PL/SQL Review

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self-check activity.

Exercise 1

In this exercise, you will create a package and procedure in the TRAINING database.

Step	Action
1	Log into UNIX account and set your ORACLE_SID to the training database. (Alternatively, open a SQLPLUS session on your local PC and use NOTEPAD.)
2	Using the VI, EMACS or NOTEPAD editor, create a package and procedure. No logic is required; simply create an empty procedure that does nothing. In the next exercise, you will use the package.procedure as a template for your applications. • SCRIPT NAME = hello_world_*yourname*.sql • PACKAGE NAME = hello_world_*yourname* • PROCEDURE NAME = P_DisplayHello
3	Run your script to create the package and procedure. Install the package under your assigned TRAINING account. Hint: type "show errors" to see why your object may have not compiled.

Section D: PL/SQL Review

Lesson: PL/SQL Toolkit

Jump to TOC

PL/SQL Toolkit overview

Oracle developed the PL/SQL toolkit to allow web programmers to create dynamic web applications using PL/SQL. The PL/SQL Toolkit is a series of packages, owned by SYS, which generate dynamic HTML tags and perform internet-related functions.

The primary packages used to create HTML tags are the HTP and HTF packages.

- The HTP package contains *procedures* which generate and send HTML tags to the browser.
- The HTF package contains *functions* which generate and return HTML tags to PL/SQL statements.

Other PL/SQL toolkit packages are available which manipulate string data, process cookies, handle authentication and interpret browser clicks on images.

HTP and HTF Packages

The htp (hypertext procedures) and htf (hypertext functions) packages generate HTML tags. For instance, the htp.anchor procedure generates the HTML anchor tag, <A>.

HTP example

The following commands generate a simple HTML document:

create or replace procedure hello AS

```
BEGIN
  htp.htmlopen; -- generates <HTML>
  htp.headopen; -- generates <HEAD>
  htp.title('Hello'); -- generates <TITLE>Hello</TITLE>
  htp.headclose; -- generates </HEAD>
  htp.bodyopen; -- generates <BODY>
  htp.header(1, 'Hello'); -- generates <H1>Hello</H1>
  htp.bodyclose; -- generates </BODY>
  htp.htmlclose; -- generates </HTML>
END;
```

Corresponding HTF functions

For every htp procedure that generates HTML tags, there is a corresponding htf function with identical parameters.

The function versions do not directly generate output in your web page. Instead, they pass their output as return values to the statements that invoked them. Use these functions when you need to nest calls.

Looking up HTF functions

To look up htf functions, see the entry for the corresponding htp procedures.

```
HTP.P('Good Morning');
      generates  Good Morning
HTP.P(HTF.BOLD('Good Afternoon'));
      generates  Good Afternoon
```

OWA Packages

- OWA_COOKIE: sub-programs send and retrieve HTTP cookies from browsers
- OWA_IMAGE: sub-programs retrieve coordinates where a user clicks on an image
- OWA_SEC: sub-programs define authentication functions; also retrieve password, username and host information for user authentication
- OWA_PATTERN, OWA_TEXT: subprograms manipulate string data
- OWA_UTIL: subprograms used to display and retrieve sets of data; handle CGI environment variables and run dynamic SQL

Section D: PL/SQL Review

Lesson: PL/SQL Toolkit (Continued)

Jump to TOC

Documentation

- Refer to the "PL/SQL Web Toolkit" topics in Section H
- View Oracle 9i Application PL/SQL Toolkit Documentation **http://download-west.oracle.com/docs/cd/A97335_01/index.htm**
- View 10g PL/SQL Reference Guide:
  - HTP: http://download-east.oracle.com/docs/cd/B19306_01/appdev.102/b14258/w_htp.htm#i1058734
  - HTF: http://download-east.oracle.com/docs/cd/B19306_01/appdev.102/b14258/w_htf.htm#i1011713

"Hello World" Example

```
CREATE OR REPLACE PACKAGE Hello_World
IS   Procedure P_DisplayHello;
END Hello_World;
/
CREATE OR REPLACE PACKAGE BODY Hello_World
IS
  Procedure P_DisplayHello IS
    BEGIN
           HTP.P('Hello World');
  END P_DisplayHello;
END Hello_World;
/
```

"Hello World" HTF Example

```
CREATE OR REPLACE PACKAGE Hello_World
IS
  Procedure P_DisplayHello;
END Hello_World;
/
CREATE OR REPLACE PACKAGE BODY Hello_World
IS
  Procedure P_DisplayHello IS
   BEGIN
           HTP.P(HTF.Bold('Hello World'));
     END P_DisplayHello;
END Hello_World;
/
```

HTP.IMG

```
htp.img (
curl in varchar2 DEFAULT NULL
calign in varchar2 DEFAULT NULL
calt in varchar2 DEFAULT NULL
cismap in varchar2 DEFAULT NULL
cattributes in varchar2 DEFAULT NULL);

htp.img('/images/logo.gif','CENTER','LOGO');
<IMG SRC="/images/logo.gif" ALIGN="CENTER" ALT="Logo">
```

HTP.ANCHOR

```
htp.anchor (
curl in varchar2
ctext in varchar2
cname in varchar2 DEFAULT NULL
cattributes in varchar2 DEFAULT NULL);
htp.anchor('http://www.oracle.com','Oracle Web Site');
  <A HREF = "http://www.oracle.com"> Oracle Web Site </A>
```

HTP.HEADER

```
htp.header (
nsize in integer
cheader in varchar2
calign in varchar2 DEFAULT NULL
cnowrap in varchar2 DEFAULT NULL
cclear in varchar2 DEFAULT NULL
cattributes in varchar2 DEFAULT NULL);

  htp.header(4,'Good Morning','center');
  <H4 align='center'>Good Morning</H4>
```

Section D: PL/SQL Review

Lesson: Common Tags

Jump to TOC

Head tags

- HTP.title('title text')
- HTP.meta('http-equiv','name','content');
- HTP.script('script_text','language');
- HTP.style('style_text');

Body tags

- HTP.p('text'); print
- HTP.hr; horiz rule
- HTP.br; line break
- HTP.fontOpen('color','face','size') font
- HTP.comment; comment
- HTP.para paragraph

Formatting tags

- HTP.bold('text','attributes');
- HTP.italic('text','attributes');
- HTP.underline('text','attributes');
- HTP.header('size','text',…'attributes');
- HTP.center('text');

List tags

- HTP.ulistOpen('clear','wrap','bullet');
- HTP.olistOpen('clear','wrap','attributes');
- HTP.dlistOpen('clear','attributes');
- HTP.menulistOpen;
- HTP.dirlistOpen;
- HTP.listHeader('text','attributes');
- HTP.listItem('text','clear','bullet');
- HTP.dlistDef('text','clear','attributes');
- HTP.dlistTerm('text','clear','attributes');

List Example

```
...
HTP.ulistOpen;
    HTP.listHeader('Registration Checklist');
    FOR my_rec IN my_cursor LOOP
          EXIT WHEN my_cursor%NOTFOUND;
          HTP.listItem(my_rec.checklist_desc);
    END LOOP;
HTP.ulistClose;
```

Section D: PL/SQL Review

Lesson: Tables

Jump to TOC

Table syntax

HTML	HTP
<TABLE>	HTP.tableOpen('border','align','wrap,'clear', 'attributes')
</TABLE>	HTP.tableClose;
<TR>	HTP.tableRowOpen('align','vertical','dp','wrap', 'attributes');
</TR>	HTP.tableRowClose;
<TH>	HTP.tableHeader('text','align','dp','wrap','rowspan', 'colspan','attributes');
<TD>	HTP.tableData('text','align','dp','wrap','rowspan', 'colspan','attributes');

Table example 1

```
...
HTP.tableOpen;
    HTP.tableRowOpen;
         HTP.tableHeader('Student ID');
         HTP.tableHeader('Grade in Class');
    HTP.tableRowClose;
    FOR my_rec IN my_cursor LOOP
      EXIT WHEN my_cursor%NOTFOUND;
      HTP.tableRowOpen;
          HTP.tableData(my_rec.stu_id);
          HTP.tableData(my_rec.class_grd);
      HTP.tableRowClose;
    END LOOP;
HTP.tableClose;
```

Table example 2

```
HTP.tableOpen('border','center', null, null,
                'BGCOLOR=gray');
     HTP.tableRowOpen('left', null, null,
                 null, 'BGCOLOR=yellow');
            HTP.tableHeader('Account');
            HTP.tableHeader('Balance');
     HTP.tableRowClose;
     HTP.tableRowOpen;
             HTP.tableData(htf.italic(acct_num));
      HTP.tableData(htf.italic(acct_bal));
     HTP.tableRowClose;
HTP.tableClose
```

Section D: PL/SQL Review

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self-check activity.

Exercise 1

In this exercise, you will modify the Hello_world_*your_name* package to display "Hello World Name" to a browser.

Step	Action
1	Open the package hello_world_*yourname*.sql you created in a previous exercise.
2	In your package, make sure all instances of *package_name* are changed to Hello_World_*youruserid*.
3	Name the Procedure *P_DisplayHello*. Remember, the name of your PACKAGE should contain your userid to make it unique (i.e. Hello_World_TRAIN22).

4	Next, here is the code to paint a Hello World message on a web page:
	`Begin`
	`  HTP.p('Hello World');`
	`End P_DisplayHello;`
	
	Save it in your directory and compile it in SQL*Plus.
	
	In SQL*Plus:
	`SQL>  @[Your directory]Hello_World.SQL`

5	If you get any compilation errors, fix them and repeat the previous steps until your package compiles without errors.
6	Grant execute on hello_world_*yourname* to www_user (web user).
7	Create public synonym hello_world_*yourname* for hello_world_*yourname*. Note: The instructor will demonstrate how to register a page in Web Tailor before completing Step 8.

Exercise 1 (cont.)

Step	Action
8	Load your package from a URL. Before you can do this, you must answer a few questions for yourself: • What is the address and port of the web server for this class? • What is the default-gateway/DAD used to connect to the database I have compiled my code for? • What is the name of the package.procedure combination that will return an HTML page? When you answer these questions, you should be able to compose an URL that looks something like this: **`http://granny.weber.edu:7777/pls/trn6dad/Hello_World_`*`yourname`*`.P_DisplayHello`** Did it work for you? If it did, you have successfully installed an object in the database that paints an HTML page, and requested a dynamic retrieval of that page from a browser. If it did not work, keep trying...

Exercise 2

Use your existing package to create a simple unordered list of street addresses from the SWBADDR table.

Step	Action
1	Open the package hello_world_*yourname*.sql you created in a previous exercise.
2	Add code to your P_DisplayHello procedure.
3	Use a cursor that selects from the SWBADDR table.
4	Use an implicit cursor to loop through each record and display the contents as unordered items.
5	Call P_DisplayHello from a browser.

Section D: PL/SQL Review

Lesson: Forms

Jump to TOC

HTP procedures

HTML Forms allow users to submit data. The PL/SQL Toolkit delivers a set of HTP procedures to display form tags.

The PL/SQL Toolkit delivers a set of HTP procedures to display form tags. In PLSQL, each html form tag is represented by a toolkit procedure.

To develop a form in PLSQL, you must create two procedures:

- **Procedure to Display Form**
  This procedure generates HTML output to display form inputs to the user. This procedure includes at least a form open tag, form close tag and any form input tags, which may be radio, checkbox, select-drop down, text box, text area or password.
- **Procedure to Process Form Input**
  When the user clicks "Submit," the data will be sent to this procedure. Therefore, each parameter will match an input filled in by a user. This form is also responsible for creating the next web page that a user sees after they click "Submit".

NAME attributes

Every form input has a NAME attribute.

When a user fills out a form, each NAME attribute stores a VALUE (or more than one value for checkbox and drop downs).

When the user clicks on Submit, the NAME/VALUE pair is passed to the processing procedure as a parameter. The processing procedure will match the NAME to a PARAMETER by spelling.

```
HTP.FORMTEXT('CITY','30',cvalue => 'Oregon');
<INPUT TYPE="text" NAME="City" Value="Oregon");
```

Oracle PL/SQL Toolkit procedures

- htp.FormOpen
- htp.FormClose
- htp.FormSubmit
- htp.FormImage
- htp.FormReset
- htp.FormHidden
- htp.FormText
- htp.FormRadio
- htp.FormSelectOpen
- twbkfrmt.FormSelectOption
- htp.FormSelectClose
- htp.FormTextArea
- htp.FormCheckBox
- htp.FormPassword

htp.FormOpen

This generates the <FORM> and </FORM> tags, which create a form section in an HTML document.

htp.formOpen(curl, cmethod, ctarget, cenctype, cattributes) ;
generates:
<FORM ACTION="curl" METHOD="cmethod" TARGET="ctarget" ENCTYPE="cenctype" cattributes>

Parameter	Description
Curl	the URL of the WRB or CGI script where the contents of the form is sent. This parameter is required.
Cmethod	the value for the METHOD attribute. The value can be "GET" or "POST".
Ctarget	the value for the TARGET attribute.
Cenctype	the value for the ENCTYPE attribute.
Cattributes	other attributes to be included

GET and POST

The GET method sends data in the URL text string. Therefore, input is limited to 256 characters in older browsers.

The POST method sends data in the body or the request. Therefore, the amount of input is not limited for POST.The ACTION tag defines the gateway, data access descriptor and package.procedure. When the user clicks on submit, the data is sent to the Oracle procedure instead of a CGI script.

```
htp.FormOpen('/pls/trng/my_package.p_process',POST);
<FORM METHOD=POST ENCTYPE="text/plain"
ACTION="/pls/trng/my_package.p_process">
htp.FormClose
```

This generates the form close tag.

Htp.formclose;
generates:
</FORM>

There are no parameters for htp.formclose. Each form open tag needs a closing tag.

htp.FormSubmit

This generates the <INPUT> tag with TYPE="submit", which creates a button that, when clicked, submits the form. If the button has a NAME attribute, the button contributes a name/value pair to the submitted data.

```
htp.formSubmit (
   cname          in       varchar2   DEFAULT NULL
   cvalue         in       varchar2   DEFAULT 'Submit'
   cattributes    in       varchar2   DEFAULT NULL);
```

generates:

```
<INPUT TYPE="submit" NAME="cname" VALUE="cvalue" cattributes>
```

Parameter	Description
Cname	value for the NAME attribute
Cvalue	value for the VALUE attribute
Cattributes	other attributes to be included

htp.formImage

This generates the <INPUT> tag with TYPE="image", which creates an image field that the user clicks to submit the form immediately.

The coordinates of the selected point are measured in pixels, and returned (along with other contents of the form) in two name/value pairs. The x coordinate is submitted under the name of the field with .x appended, and the y coordinate with .y appended. Any VALUE attribute is ignored.

```
htp.formImage (
   cname          in       varchar2
   csrc           in       varchar2
   calign         in       varchar2   DEFAULT NULL
   cattributes    in       varchar2   DEFAULT NULL);
```

generates:

```
<INPUT TYPE="image" NAME="cname" SRC="csrc" ALIGN="calign"
cattributes>
```

Parameter	Description
Cname	value for the NAME attribute
Csrc	value for the SRC attribute, which specifies the image file
Calign	value for the align attribute
Cattributes	other attributes to be included

htp.FormReset

This generates the <INPUT> tag with TYPE="reset", which creates a button that, when selected, resets the form fields to their initial values.

```
htp.formReset (
   cvalue          in       varchar2   DEFAULT 'Reset'
   cattributes     in       varchar2   DEFAULT NULL);
```

generates:

```
<INPUT TYPE="reset" VALUE="cvalue" cattributes>
```

Parameter	Description
Cname	value for the NAME attribute
Cvalue	value for the VALUE attribute
Cattributes	other attributes to be included

htp. FormHidden

This generates the <INPUT> tag with TYPE="hidden", which inserts a hidden form element. This element is not seen by the user. It submits additional values to the script.

```
htp.formHidden (
   cname           in       varchar2
   cvalue          in       varchar2   DEFAULT NULL
   cattributes     in       varchar2   DEFAULT NULL);
```

generates:

```
<INPUT TYPE="hidden" NAME="cname" VALUE="cvalue" cattributes>
```

Parameter	Description
Cname	value for the NAME attribute
Cvalue	value for the VALUE attribute
Cattributes	other attributes to be included

htp.FormText

This generates the <INPUT> tag with TYPE="text", which creates a field for a single line of text.

```
htp.formText (
   cname          in       varchar2
   csize          in       varchar2   DEFAULT NULL
   cmaxlength     in       varchar2   DEFAULT NULL
   cvalue         in       varchar2   DEFAULT NULL
   cattributes    in       varchar2   DEFAULT NULL);
```

generates:

```
<INPUT TYPE="text" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength"
VALUE="cvalue" cattributes>
```

Parameter	Description
Cname	value for the NAME attribute
Csize	value for the SIZE attribute
Cmaxlength	value for the MAXLENGTH attribute
Cvalue	value for the VALUE attribute
Cattributes	other attributes to be included

htp.FormRadio

This generates the <INPUT> tag with TYPE="radio", which creates a radio button on the HTML form. Within a set of radio buttons, the user selects only one. Each radio button in the same set has the same name, but different values. The selected radio button generates a name/value pair.

```
htp.formRadio (
   cname          in       varchar2
   cvalue         in       varchar2
   cchecked       in       varchar2   DEFAULT NULL
   cattributes    in       varchar2   DEFAULT NULL);
```

generates:

```
<INPUT TYPE="radio" NAME="cname" VALUE="cvalue" CHECKED
cattributes>
```

Parameter	Description
Cname	value for the NAME attribute.
Cchecked	if the value for this parameter is not NULL, the CHECKED attribute is added to the tag.
Cvalue	value for the VALUE attribute.
Cattributes	other attributes to be included

Example

```
htp.formradio('fav_band','Beatles');
htp.formradio('fav_band','Zeppelin','X');
```

generates:

```
<input type="radio" name="fav_band" value="Beatles">
<input type="radio" name="fav_band" value="Zeppelin" checked>
```

htp.FormSelectOpen

This generates the <SELECT> and </SELECT> tags, which creates a Select form element. A Select form element is a listbox where the user selects one or more values. The values are inserted using htp.FormSelectOption.

```
htp.formSelectOpen (
   cname          in       varchar2
   cprompt        in       varchar2   DEFAULT NULL
   nsize          in       integer    DEFAULT NULL
   cattributes    in       varchar2   DEFAULT NULL);
```

generates:

```
<SELECT NAME="cname" SIZE="nsize" cattributes>
```

Parameter	Description
Cname	value for the NAME attribute.
Cprompt	string preceding the list box.
Nsize	value for the SIZE attribute.
Cattributes	other attributes to be included

Note

Use **twbkwbis.p_FormSelectOption** to display options in a select list, not htp.FormSelectOption.

The Banner standardized procedure creates a display value and a pass value. For example, you may display a description like "Post Graduate" but pass the validation code "PG" when the form gets submitted.

twbkwbis.p_ FormSelect Option

This generates the <OPTION> tag, which represents one choice in a Select element.

```
twbkwbis.p_formSelectOption (
      cdispvalue         in   varchar2,
      cpassvalue         in   varchar2 default null,
      cselected          in   varchar2 default null,
      cattributes        in   varchar2 default null
   );
```

generates:

```
<OPTION VALUE="Cpassvalue" SELECTED cattributes>cdispvalue
```

Parameter	Description
Cdispvalue	text for the option.
Cpassvalue	value passed to the database
Cselected	If the value for this parameter is not NULL, the SELECTED attribute is added to the tag.
Cattributes	other attributes to be included

Example

```
   htp.p_formSelectOpen('flavor');
   twbkwbis.p_formSelectOption('Rocky Road Chocolate', cpassvalue =>
'R');
   twbkwbis.p_formSelectOption('Strawberry', cpassvalue => 'S');
   twbkwbis.p_formSelectOption('Vanilla', cpassvalue => 'V', cselected
=> 'X' );.
   htp.formSelectClose;
```

generates:

```
<SELECT NAME="flavor">
<OPTION VALUE=" R"> Rocky Road Chocolate
<OPTION VALUE=" S"> Strawberry
<OPTION VALUE=" V" SELECTED> Vanilla
</SELECT>
```

htp.FormText Area

This generates the <INPUT> tag with TYPE="text", which creates a field for a single line of text.

```
htp.formTextarea (
   cname          in       varchar2
   nrows          in       integer
   ncolumns       in       integer
   calign         in       varchar2   DEFAULT NULL
   cattributes    in       varchar2   DEFAULT NULL);
```

generates:

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign"
cattributes></TEXTAREA>
```

Parameter	Description
Cname	value for the NAME attribute.
Nrows	value for the ROWS attribute. This is an integer.
Ncolumns	value for the COLS attribute. This is an integer.
Calign	value for the ALIGN attribute.
Cattributes	other attributes to be included

htp.FormCheckBox

This generates the <INPUT> tag with TYPE="checkbox", which inserts a checkbox element in a form. A checkbox element is a button that the user toggles on or off.

```
htp.formCheckbox (
   cname          in       varchar2
   cvalue         in       varchar2   DEFAULT 'on'
   cchecked       in       varchar2   DEFAULT NULL
   cattributes    in      varchar2   DEFAULT NULL);
```

generates:

```
<INPUT TYPE="checkbox" NAME="cname" VALUE="cvalue" CHECKED
cattributes>
```

Parameter	Description
Cname	value for the NAME attribute.
Cvalue	value for the VALUE attribute.
Cchecked	if the value for this parameter is not NULL, the CHECKED attribute is added to the tag.
Cattributes	other attributes to be included

htp.Form Password

This generates the <INPUT> tag with TYPE="password", which creates a single-line text entry field. When the user enters text in the field, each character is represented by one asterisk. This is used for entering passwords.

htp.formPassword (
  cname in varchar2
  csize in varchar2
  cmaxlength in varchar2 DEFAULT NULL
  cvalue in varchar2 DEFAULT NULL
  cattributes in varchar2 DEFAULT NULL);

generates:

```
<INPUT TYPE="password" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength"
VALUE="cvalue" cattributes>
```

Parameter	Description
Cname	value for the NAME attribute.
Csize	value for the SIZE attribute.
Cmaxlength	value for the MAXLENGTH attribute.
Cvalue	value for the VALUE attribute.
Cattributes	other attributes to be included

Encapsulation

To ensure that form inputs line up properly in Self Service Banner, inputs are encapsulated in an HTML form.

Each form input is placed in an HTML data cell. Optionally, a label may be placed to the left of the data cell.

Example

```
htp.tablerowopen;

htp.tabledata('<LABEL for='|| email_ind_id||'>'||'Email
Indicator'||'</Label>');

htp.tabledata (
           HTF.formcheckbox (
              cname => 'email_ind_in',
              cvalue=> 'Y',
              cchecked=> 'CHECKED',
              cattributes=> 'ID="email_ind_id"' ));
htp.tablerowclose;
```

Section D: PL/SQL Review

Lesson: Labels

Jump to TOC

Introduction

Labels are used to describe a form input. They are tied to the form input using an ID.

Only one ID can be defined on a page. An ID attribute is a unique value that can be referenced in the style sheet.

Labels are optional. However, they are commonly used in Banner.

Example 1

<Label for="idname"> Last Name: </Label>
<INPUT type="text" name=last_name_in id="idname">

Example 2

Notice how the label and form input are encased as table elements.

```
htp.tablerowopen;
htp.tabledata('<LABEL for='|| email_ind_id||'>'||'Email
  Indicator'||'</Label>');
htp.tabledata (
           HTF.formcheckbox (
              cname => 'email_ind_in',
              cvalue=> 'Y',
              cchecked=> 'CHECKED',
              cattributes=> 'ID="email_ind_id"' ));
Htp.tablerowclose;
```

Section D: PL/SQL Review

Lesson: Complete Form Examples: Text Input

Jump to TOC

Display Text Input

```
Htp.FormOpen('/pls/trng/mypackage.P_ProcessTerm',
        'POST');
htp.tableOpen;
  htp.tableRowOpen;
  htp.tableData(htf.bold('Select Term:'));
  htp.tableData(htf.formText('term','6'));
  htp.tableRowClose;
htp.tableClose;
htp.formSubmit(null,'Submit Term');
htp.formClose;
```

HTML Output

```
<FORM ACTION="/pls/trn6dad/mypackage.P_ProcessTerm",
  METHOD=POST>
<TABLE>
  <TR>
        <TD><B>Select a term:</B></TD>
        <TD><INPUT TYPE="Text" NAME=term SIZE=6>
        </TD>
  </TR>
</TABLE>
<INPUT TYPE="Submit" NAME="Submit Term">
</FORM>
```

Accept Input

```
PROCEDURE P_ProcessTerm(term VARCHAR2)
IS
BEGIN

OPEN valid_term_cursor(term);
  FETCH valid_term_cursor INTO valid_term_row;
  IF valid_term_cursor%notfound THEN
        raise invalid_term_entered;
  END IF;
  CLOSE valid_term_cursor;

  /* valid term entered, go to schedule */

htp.p('Thank you for submitting the term');
   End p_processterm;
```

Display Drop Downs Input

```
   Htp.formOpen ('/pls/CON6/bwzkwpay.p_calc_refund');

   htp.formHidden('pidm','482911');

   htp.formSelectOpen('refund_class', 'Pick class to request a
refund:');
   twbkwbis.P_formSelectOption('English 410', '410', cselected =>
'X');
   twbkwbis.p_formSelectOption('Biology 305', '305');
   twbkwbis.p_formSelectOption('Physics 400', '305');
   twbkwbis.p_formSelectOption('Mathematics 320', '320');
   htp.formSelectClose;

   htp.formSubmit (NULL, ' Press here ');
   htp.formClose;
```

HTML Output

```
   <FORM ACTION="/pls/CON6/bwzkwpay.p_calc_refund" method="POST">

   <input type="hidden" name="pidm" value="482911">

   Pick class to request a refund:
   <select name="refund_class">
     <Option value="410"> English 410
     <Option value="305"> Biology 305
     <Option value="400"> Physics 400
     <Option value="320"> Mathematics 320
   </select>

   <input type="submit" value=" Press here ">
   </form>
```

Accept Input

```
    PROCEDURE p_calc_refund(pidm VARCHAR2, refund_class VARCHAR2)
    IS
      pidm_v NUMBER := to_number(pidm);
     account NUMBER;
    BEGIN
       SELECT account_amount into account  FROM account WHERE
account_pidm = pidm_v AND account_refund_class = refund_class;

    htp.p('You will be refunded $'||account  );
    P_peformrefund(pidm_v, refund_class);
    …
    END p_calc_refund;
```

Section D: PL/SQL Review

Lesson: Self Check

Jump to TOC

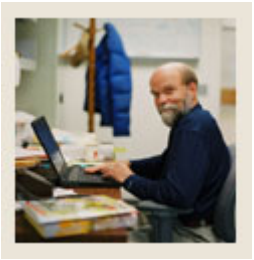
Directions

Use the information you have learned in this workbook to complete this self-check activity.

Exercise 1

Create a form to prompt a user's favorite vacation destination and display the input.

Step	Action
1	Open your existing hello_world_*yourname.sql* and create a new procedure to display a form. Prompt a user to enter their favorite vacation destination into a text field.
2	In your hello_world_yourname package, create a procedure to process the form and display the input back to the user.
3	Open a browser and test your form.



Section E: Security

Lesson: Overview

Jump to TOC

Introduction

Self Service Banner relies on application logic, network and database security to ensure data integrity and privacy. The purpose of this discussion is to identify how programmers can integrate Banner security into their applications.

Section Contents

Section E: Security

Lesson: Security

Jump to TOC

Security Tables

Third party access table, GOBTPAC, stores PIN for each web user in Banner. Users cannot get access to Self Service Banner unless they have a record in this table.

General role view, GOVROLE, is based on the module specific master tables. Defines the valid roles for each user. View determines which menus are available upon login. Additional web specific roles are specified in the web tailor roles table, TWGRROLE.

Web Tailor Web ID table, TWGBWSES, stores unique web ID for each user session. User sessions are tracked by unique web IDs. Web IDs are regenerated and downloaded as cookies each time a user clicks on a new page. Only one browser session is allowed per user.

Security Calls

Procedures, which display data to end users, call a Banner security procedure **twbkwbis.F_ValidUser(pidm number)**.

To integrate your custom package into Banner Security, simply add the procedure call at the beginning of your code. The PIDM is actually an output variable, determined by the value in TWGBWSES.

GOVROLE and TWGRROLE roles are fetched into PLSQL table, role_table, by **twbkslib.p_fetchroles (pidm number)**

Security Example

Create or replace package body Hello_Chris

```
    IS
      Procedure P_DisplayHello IS
             pidm number;
      BEGIN
            IF NOT twbkwbis.F_ValidUser(pidm) then
               return;
            END IF;
    END;
```

Section E: Security

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self-check activity.

Exercise 1

Add a security call to your P_DisplayHello procedure.

Step	Action
1	Open your existing hello_world_*yourname.sql* and modify your existing *P_DisplayHello* Procedure by adding a call to *twbkwbis.F_ValidUser*.
2	Open a browser and request *P_DisplayHello*. What are the results?
3	In the same browser, open and log into SSB on the same database. Now request *P_DisplayHello*. What are the results?

Section F: Development and Standards

Lesson: Overview

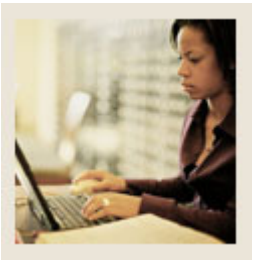
Jump to TOC

Introduction

Web Tailor provides a GUI interface to add custom menus and application pages to the Self Service product. SCT recommends that programmers use the GUI interface, rather than table inserts, to register procedures with Web Tailor.

To simplify and standardize the programming process, SCT provides several Web Tailor procedures to display the TOP and BOTTOM of an application page.

Section Contents



Web Tailor tables

Underlying Web Tailor tables define the title, page name, menu and help links, and information text displayed at the TOP of a document. Underlying Tailor tables also define the logo and homepage link displayed at the bottom of a document.

Steps to Integrate

- Add Banner open document code, information text code and close document code
- Compile your package
- Create links to access the package in Menu
- Register the packages with Web Tailor
- Login into SSB and test your link to your application page

P_OPENDOC

TWBKWBIS.P_OPENDOC('Your Procedure');

- Creates the <HTML> <HEAD> <BODY> tags
- Name of procedure defined in Web Tailor passed as parameter (case sensitive)
- Calls style sheet defined in Web Tailor. Defaults to web_defaultapp.css
- Displays title, page name, menu bar and help links defined in Web Tailor

P_CLOSEDOC

TWBKWBIS.P_CLOSEDOC('Version Number');

- Creates the </HTML> </BODY> tags
- Name of procedure defined in Web Tailor passed as parameter (case sensitive)
- Displays "Powered by SCT logo" and version number

P_DISPINFO

TWBKWBIS.P_DISPINFO('Your Procedure','Label');

- Displays information text associated with procedure in Web Tailor
- Name of procedure defined in Web Tailor passed as first parameter (case sensitive)
- Label of information text passed as second parameter. Defaults to "DEFAULT"

Section F: Development and Standards

Lesson: Document Example

Jump to TOC

Example code

```
CREATE OR REPLACE PACKAGE BODY Hello_Chris
IS Procedure P_DisplayHello IS
     pidm number;
BEGIN
    if not twbkwbis.F_ValidUser(pidm)
    then  return;
    end if;
  twbkwbis.P_OpenDoc('Hello_Chris.P_DisplayHello');
twbkwbis.P_DispInfo ('Hello_Chris.P_DisplayHello','DEFAULT');
- -
  - - application code goes here
- -
twbkwbis.P_CloseDoc('5.1');
```

Section F: Development and Standards

Lesson: Adding a Procedure to Web Tailor

Jump to TOC

Steps

Choose the new Web Tailor menu. Remember, your new Web Tailor records will all be "LOCAL."

Step	Action
1	Create the package under your training account. Create a public synonym for your package and grant execute on the package to WWW_USER (the DAD Oracle account - not to PUBLIC).
2	From the Web Tailor Main Menu, select **(Customize a) Web Menu or Procedure:** Then, click on the **Create** button to register/create your procedure(s). If necessary, use an existing procedure as a guideline of what to enter in the fields.
3	To add this new procedure/page to an existing menu page, choose the existing menu under (Customize a Set of) **Menu Items** (e.g. choose bmenu.P_GenMnu for the Personal Information Page), add the new procedure/page as a new menu item, and click both the "Enabled Indicator" and "Database Procedure". **If necessary, first copy Baseline to Local!**
4	To display procedures/pages as menu items on existing Application pages (rather than a menu page) these will appear as bottom menu links in the footer section of the Application page. To do so, choose an existing Application page under (Customize a Set of) **Menu Items** (e.g. choose twbkwbis.P_ChangePin for the SSB change of Pin Page which is a menu item on the Personal Information page). Select "Add a new menu item" and add your procedure/page. Make sure to include link text and select both the "Enabled Indicator" and "Submenu Indicator".
5	If you have a static html help text file for the menu/procedure(s), place the file in the appropriate help directory on your web server, and enter the URL (e.g. /genhelp/textfile.htm) in the "Help Link (URL)" field for your menu/procedure.
6	If you want Info text displayed for the menu/procedure(s), there is a "Customize Information Text" button at the bottom of the "Customize a Web Menu or Procedure" form, as well as a separate Web Tailor menu item "Customize a Set of Information Text". (Your procedure must have a call to the "twbkwbis.P_DispInfo" procedure in order for the info text to be displayed.)

Section F: Development and Standards

Lesson: Printing Menu Items

Jump to TOC

Introduction

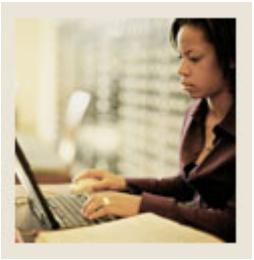
You may want to print menu items in vertical or horizontal alignment. The Self Service Banner formatting package provides a procedure to read TWGRMENU and print links.

Twbkfrmt.P_ PrintMenu

Prints a menu with items from TWGRMENU.

```
twbkfrmt.P_PrintMenu (
   name              IN   VARCHAR2,
   display_type      IN   VARCHAR2 DEFAULT NULL,
   class_in          IN   VARCHAR2 DEFAULT NULL,
   num_in_row        IN   VARCHAR2 DEFAULT NULL,
   validate_links    IN   BOOLEAN DEFAULT TRUE,
   map_title         IN   VARCHAR2 DEFAULT NULL,
   ccaption          IN   VARCHAR2 DEFAULT NULL
);
```

Parameter	Description
Name	Name of the menu, from TWGRMENU_NAME
display_type	'F' - Footer, or bottom, links. 'B' - Body, or full-page, menu.
class_in	If the size of the links should be different than the default, send the new size here.
num_in_row	By default, for a "Footer" menu, there are 3 links on each row. This is the override for that number
validate_links	By default, the twbkwbis.F_ValidLink call is made before printing a menu item. If the menu is generated within the Student Admissions module, the user will not be authenticated with normal WebTailor security. When this flag is FALSE, links are printed as-is.



Section F: Development and Standards

Lesson: Self Check

Jump to TOC

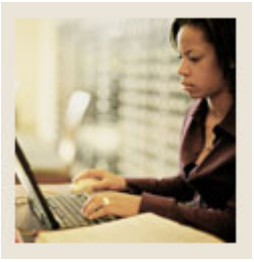
Directions

Use the information you have learned in this workbook to complete this self-check activity.

Exercise 1

Incorporate your hello_world procedure into Web Tailor by adding procedure calls to your P_DisplayHello procedure.

Step	Action
1	Open your existing hello_world_*yourname.sql* and modify your existing P_DisplayHello to include the standard OPEN and CLOSE procedure calls.
2	Compile your package.
3	Login to Web Tailor and create your procedure under new Web Tailor.
4	Add a link under the "Personal Information menu" to your procedure.
5	Access the new link to P_DisplayHello. Did the TOP and BOTTOM of the document display? (View and analyze the source from your browser.)
6	Open your existing hello_world_*yourname.sql* and modify your existing P_DisplayHello to include Information Text.
7	Compile your package.
8	Login to Web Tailor and associate information text to P_DisplayHello.
9	Access the new link to P_DisplayHello. Did the information text display?



Section F: Development and Standards

Lesson: Development and Standards - Tables, Text and Links

Jump to TOC

Overview

SCT provides a set of standard Banner formatting procedures to replace the direct use of HTP and HTF procedures and functions in SSB code. Instead, SCT supplies a set of table, list, anchor, images and fonts procedures, developed to create a uniform look and feel. These procedures infuse each respective TAG with SCT specific attributes and styles.

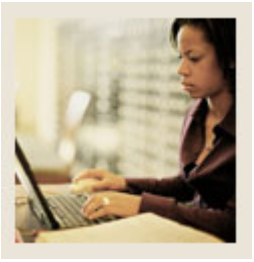
There are several distinct advantages to coding with SCT provided procedures.

- First, the developer eliminates the need to create and copy lengthy attribute parameters for each HTP or HTF procedure.
- Second, the developer ensures that the same HTML elements will look the same throughout the application.
- Third, only core application display needs to be developed.
- Finally, the use of SCT formatting procedures ensures the developer follows WC3 standards.

Full documentation for SCT guidelines and standards is available Self Service Banner Methodology Handbook. The twbkFRMT package contains SCT procedures/functions used for formatting.

Note

This section is not intended as replacement for SCT documentation. Rather, it is intended as a guide to understanding important coding techniques developers use to develop your SSB applications.



Section F: Development and Standards

Lesson: Text

Jump to TOC

Printing text

If you want to do this . . .	Use this call....	Call Type
Print section headers	twbkfrmt.p(f)_printheader	Web Tailor
Print quotes	htp(f).blockquoteopen htp(f).blockquoteclose	PLSQL Toolkit
Start new paragraph	htp(f).para, htp(f).paragraph	PLSQL Toolkit
Print normal text	twbkfrmt.p(f)_printtext	Web Tailor
Print bold, big, small, centered, emphasized, italicized, strikethrough, subscripted, superscripted, teletype, or underlined text. Print text with a certain font, color, borders or line spacing. • Classes will be based on styles (e.g.,Info Text, page title text, etc.), not on individual style elements (bold, big, small, etc.) . • Refer to web_defaultapp.css for SCT provided text styles	twbkfrmt.p(f)_printtext('my text', class_in=> ?)	Web Tailor

National Language Support

The G$_NLS package provides language translation support. Specifically, the GET function of G$_NLS returns a message in the target language. If no translation is found, the function returns the default message.

The GET function accepts 32 parameters. The first two parameters specify the message number and message type. The third parameter accepts the default message. The final 29 parameters are for substitution values. If no translation for p_msg_id_in and p_msg_type_in is found then the default message, p_msg_txt_in, is returned.

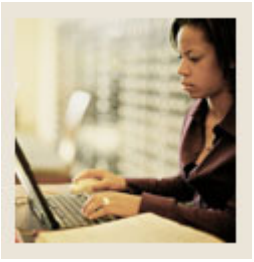
```
GET(
  p_msg_id_in     IN   VARCHAR2, -- Message number, 'x' means no
  message
p_msg_type_in      IN   VARCHAR2, -- 'SQL'
p_msg_txt_in       IN   VARCHAR2, -- Text of message
var01              IN   VARCHAR2, -- value to substitue for %01% etc
  in p_m
              ….
  var32            IN   VARCHAR2)
```

Example of Language Support

```
   HTP.p ('Number of cookie sections found: '
||twbkauth.cp_num_cookie_vals);

     Should be

   HTP.p (G$_NLS.Get('TWBKWBI1-0031', 'SQL','Number of cookie sections
found: %01%', twbkauth.cp_num_cookie_vals) );
```

In this case, the function replaces the %01% with the value twbkauth.cp_num_cookie_vals. The output might look like this:

Total number of cookie sections found: 4



Section F: Development and Standards

Lesson: Tables

Jump to TOC

Data Display

- Format <TABLE> tag to display uniform data

```
twbkfrmt.P(F)_TableOpen('DATADISPLAY');
```

Data Entry

- Format <TABLE> display input for forms

```
twbkfrmt.P(F)_TableOpen('DATAENTRY');
```

Plan Display

- Format <TABLE> display items, links or images

```
twbkfrmt.P(F)_TableOpen('PLAIN');
```

For a listing of table procedures, refer to Using UI Conversion Methodology with SCT Banner Web.

Table Examples

```
   twbkfrmt.P_TableOpen('DATADISPLAY' ccaption =>
                     'Your E-mail Address');
   twbkfrmt.P_TableRowOpen;
   twbkfrmt.p_tabledataheader ('E-mail Address');
twbkfrmt.p_tabledatalabel(' Joe Smith');
      twbkfrmt.P_TableData ('jsmith@yahoo.com');
   twbkfrmt.p_tablerowclose;
   twbkfrmt.p_tableclose;
```

Tabular vs. Non-Tabular

For non-tabular data (images, forms and menu links)

```
   P_TableDataOpen(datatype=>'nontabular')
   twbkfrmt.P_PrintImage (twbklibs.twgbwrul_rec.twgbwrul_error_gif);
   P_TableClose;

   For tabular (data from SCT Banner tables)

   P_TableData('Text Data from Banner');
```

Handling special characters

Certain characters require substitution when used in a browser URL. Therefore, the formatting package has a function, twbkfrmt.F_Encode, to replace subset of these characters with the corresponding code.

```
FUNCTION f_encode (string_in IN VARCHAR2)
RETURN VARCHAR2 IS
temp_string VARCHAR2 (1300) := string_in;
BEGIN
      temp_string := REPLACE (temp_string, '%', '%25');
      temp_string := REPLACE (temp_string, '+', '%2B');
      temp_string := REPLACE (temp_string, ' ', '+');
      temp_string := REPLACE (temp_string, '/', '%2F');
      temp_string := REPLACE (temp_string, ':', '%3A');
      temp_string := REPLACE (temp_string, ';', '%3B');
      temp_string := REPLACE (temp_string, '@', '%40');
      temp_string := REPLACE (temp_string, '&', '%26');
      temp_string := REPLACE (temp_string, '=', '%3D');
      temp_string := REPLACE (temp_string, '?', '%3F');
      temp_string := REPLACE (temp_string, '''', '%27');
      RETURN temp_string;
  END f_encode;
```

Example of twbkfrmt. F_Encode

```
twbkfrmt.f_encodeurl('bwrktrkr.P_DispTrkReq?aidy_in='
|| twbkfrmt.f_encode(aidy)
||'&calling_proc_name='||'bwrksumm.P_DispSumm')
```

Section F: Development and Standards

Lesson: Anchors

Jump to TOC

Introduction

Use the print anchor procedure, twbkfrmt.p_printanchor to display links to other pages.

Syntax

```
   twbkfrmt.P_PRINTANCHOR('/pls/dad/hello_world_chris.p_displayhello',
   ctext =>   'hello world chris');

   <A HREF= '/pls/dad/hello_world_chris.p_displayhello'> hello world
chris </A>
```

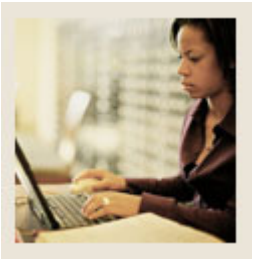
Characteristics

- Pages that display in the browser URL need to be registered in Web Tailor
- Use the twbkfrmt.F_EncodeURL function to handle special characters
- Use the twbkwbis.f_cgibin function to return values /pls/*dad/*

You may need to move your code from one database to another. To do this, your links must reference the correct Database Access Descriptor. The function f_cgibin automatically detects the correct string.

Anchor Example

```
   twbkfrmt.p_printanchor (
            curl    => twbkfrmt.f_encodeurl (
                       twbkwbis.f_cgibin ||
                       'bwrkolib.P_SelDefAidy?aidy=' ||
                       twbkfrmt.f_encode(aidy) ||
                       '&' ||
                       'calling_proc_name=' ||
                       'bwrksumm.P_DispSumm'),
            ctext   => g$_nls.get (
                       'BWRKSUM1-0041',
                       'SQL',
                       'Select Another Aid Year'
                        ),
      cattributes   => ' class = "whitespacelink"'
   );
```



Section F: Development and Standards

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self-check activity.

Exercise 1

Create a procedure to display the content of the course validation table, SVWCRSE (or any validation table you choose) in an HTML table. Display the course number, course description, credit hours and activity date.

Step	Action

1	Use a UNIX editor to open up a new file and create a new package.procedure:
	`SCRIPT NAME = SSB_`*`yourname`*`.sql`
	`PACKAGE NAME = SSB_`*`yourname`*
	`PROCEDURE NAME = P_DisplayCourse`

2	Create a cursor in the SSB_*yourname* to select all rows from the SWVCRSE table.
3	Use a LOOP statement in the P_DisplayCourse to display the contents of SWVCRSE cursor. Display the content using an HTML table. Include headers for each column. Hint: use the DATADISPLAY option of P_TableOpen.
4	Add security and Web Tailor procedure calls to authenticate and present the TOP and BOTTOM of the document, respectively.
5	Register your procedure with Web Tailor. Include a link to the procedure in the "Personal Information Menu."
6	Test your procedure. Did the content of the table display as you expected?

Exercise 2

Create a link in your Hello World P_DisplayHello to your SWVCRSE (ssb_*yourname*.p_displaycourse) procedure.

Step	Action
1	Modify your Hello_World.P_DisplayHello to include a link to SSB_*yourname*.P_DisplayCourse using the P_PRINTANCHOR procedure.
2	Login to SSB and test the link.

Section F: Development and Standards

Lesson: Development and Standards: Data Entry

Jump to TOC

Overview

All data entry in Self Service Banner is done using HTML forms. Whether a student is searching for an open class or an employee is filling out a survey, SCT presents and processes the data using a form. This topic will review the steps necessary to create a form in SSB.

Forms and procedures

Each Banner form should have at least two procedures.

The first procedure paints the HTML form and input types (checkboxes, radio buttons, text boxes, select lists, etc.).

The second procedure processes the form data as input parameters.

SCT does not provide formatting procedures/functions for HTML forms. Instead, SCT uses the PL/SQL Toolkit HTP and HTF functions/procedures to display the form.

Implementing a form

To implement a form, SCT uses an HTML table to format the input.

twbkFRMT.P_TABLEOPEN('DATAENTRY') is used specifically for form input.

Form Example

Create a form to allow a student to change his/her address.

- The student should see a drop-down listing of all address types they have records for… perhaps a business address (BA) and a personal address (PA)
- By choosing an address type from a drop-down and clicking on "submit", the correct address will display

Form procedure and cursor

```
CREATE OR REPLACE PACKAGE BODY Hello_Chris
IS
Procedure P_Address_Types IS
         pidm number;
         cursor address_cursor(pidm_v number) is
            select * from saturn.spraddr where
            spraddr_pidm  = pidm_v;
BEGIN
```

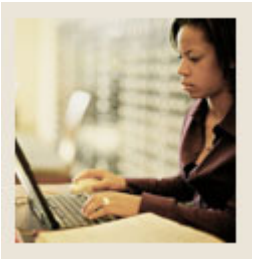
Display the form

… In procedure P_Address_Types

```
    htp.formOpen(twbkwbis.f_cgibin||'/hello_chris.Process','post');
    twbkfrmt.P_TableOpen('DATAENTRY');
       twbkfrmt.P_TableRowOpen;
          twbkfrmt.P_TableDataLabel('Your Address Types', 'left');
          twbkfrmt.P_TableDataOpen;
          htp.formSelectOpen('atype', 'pick the address type');
            FOR address_rec in address_cursor(pidm)
            LOOP
              IF address_rec.spraddr_atyp_code = 'PA'
              THEN

twbkwbis.P_FormSelectOption(address_rec.spraddr_atyp_code,
address_rec.spraddr_atyp_code, 'SELECTED');
            ELSE

twbkwbis.P_FormSelectOption(address_rec.spraddr_atyp_code,
address_rec.spraddr_atyp_code);
             END IF;
             END LOOP;
             twbkfrmt.P_TableDataClose;
         twbkfrmt.P_TableRowClose;
      twbkfrmt.P_TableClose;
    htp.formSubmit(null,'Submit Term');
      htp.FormClose;
```

Process the Form

```
CREATE OR REPLACE PROCEDURE
display_address_your_name_Process (atype IN varchar2 )
 pidm number;
         cursor spraddr_cursor(pidm_v pidm, atyp_v varchar2)  is
            select * from saturn. where
            spraddr_pidm  = pidm_v
            and spraddr_atyp_code = atyp_v;
BEGIN   - -  security calls to collect PIDM value
 for address_rec in spraddr_cursor(pidm, atype)  loop
          twbkfrmt.P_TableOpen('DATADISPLAY');
          twbkfrmt.P_TableRowOpen;
          twbkfrmt.P_TDLabel('Address', 'Left');
          twbkfrmt.P_TableDataOpen;
          twbkfrmt.P_PrintText(address_rec.SPRADDR_STREET_LINE1);
          twbkfrmt.P_TableDataClose;
          twbkfrmt.P_TableRowClose;
           twbkfrmt.P_TableClose;
    end loop;
END;
```



Section F: Development and Standards

Lesson: Storing Parameters

Jump to TOC

Introduction

Any PIDM based value can be stored in the TWGRWPRM table. For example, you can store the term code while a student registers for classes. At time of execution, the PIDM is returned by security function. Given the PIDM, a programmer defined PIDM can be inserted and fetched.

Example

```
sqlplus>desc TWGRWPRM
 Name                           Null?    Type
 ------------------------------ -------- -------------
 TWGRWPRM_PIDM                      NOT NULL NUMBER
 TWGRWPRM_PARAM_NAME            NOT NULL VARCHAR2(30)
 TWGRWPRM_PARAM_VALUE           NOT NULL VARCHAR2(255)
 TWGRWPRM_ACTIVITY_DATE         NOT NULL DATE
```

Insert a Parameter

Procedure to store the value of a parameter in the twgrwprm table

PROCEDURE twbkwbis.P_SetParam(pidm number, name varchar2, val varchar2);

Parameter	Description
Pidm	PIDM of the user logged in
Name	Programmer defined name – required to fetch value at later time
Value	Stored Character string, max 255 char

Fetch a Parameter

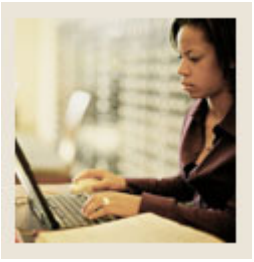
Function to return the parameter value for a pidm and parameter name

FUNCTION twbkwbis.F_GetParam(pidm number, name varchar2) return varchar2;

Parameter	Description
Pidm	PIDM of the user logged in
Name	Programmer defined name – required to fetch value at later time

Example

```
    pidm number;
    last_name_v spriden.spriden_last_name%type;
    Begin
     if not twgkwbis.F_ValidUser(pidm) then
     return;
         end if;
      --
         twgkwbis.P_OpenDoc('Hello_Chris.P_DisplayHello');

    -- insert a parameter:
    Select spriden_last into last_name_v where spriden_pidm = pidm and
    spriden_change_ind is null;
    P_SetParam(pidm, 'Last_Name', last_name_v);
    -- print Last name from TWGBPARM
    Htp.p(F_GetParam(pidm, 'Last_Name'));
..
```



Section F: Development and Standards

Lesson: Validation

Jump to TOC

Introduction

Basic validation of Form elements is available in the Self Service Banner package twbkvald. The use of Javascript means that it will not work on all browsers.

Declare validation variables

```
      validate_tab           twbklibs.varchar2_tabtype;
  -- Table for storing the fields that need to be checked for null values

      num_validate           NUMBER (8)                := 0;
```

Open the form using a special procedure

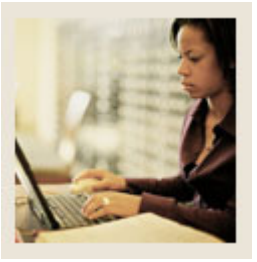
```
   twbkvald.p_validateformopen (
       twbkwbis.f_cgibin || 'bwkkwadm.p_saveelement');
```

Add the form element and validation procedure

```
HTP.formtext (
              cname       => 'last_name_in',
              csize       => '3',
              cmaxlength  => '3',
              cvalue      =>  125,
              cattributes => 'ID="last_name_id"'
              );

         twbkvald.p_addvalidatefield (
            'last_name_in', -- This is a required field
            'I',
            g$_nls.get ('BWKKWADM-0022', 'SQL', 'Label Width in
                         Pixels'),
            validate_tab,
            num_validate
            );

   --  check_format:  'Y'='I'=Integer, 'C'=Currency, 'A'=Alpha,
   --                      'O'=Alphanumeric, 'N'=No check
```

Use a special submit button to test validation

```
twbkvald.p_validatesubmitbutton (
            validate_tab=> validate_tab,
            num_validate=> num_validate,
            cname => 'SUBMITBUTTON'
          );
   twbkvald.p_validateresetbutton; -- Print a reset button
   HTP.formclose;
```



Section F: Development and Standards

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self-check activity.

Exercise 1

Create a form to display a student's grades for a term. After a student selects a valid term from a drop down and clicks submit, a page displays each course number and corresponding GPA for that term.

The PLSQL generated FORM will require two procedures:

- SSB_*yourname*.P_PromptTerm: To display the FORM
- SSB_*yourname*.P_DisplayGrades: To process the FORM

Table: SWRREGS: pidm, term, grade, crn and activity date

Hints: In both procedures, you do not need to prompt the student for his name or PIDM. By calling the security procedure twbkwbis.F_ValidUser(pidm number), you can retrieve a student's PIDM.

Step	Action
1	Open SSB_*yourname*.sql and add a new procedure in the SSB_*yourname* package named P_PromptTerm.

2	In P_PromptTerm:
	• Create a cursor on SWRREGS to return TERM codes for one student (hint: use the DISTINCT clause to return unique TERM codes)
	  • `Cursor c1 is select distinct term_code from swrregs where pidm = pidm_v;`
	• Create a cursor, C2, on SWRREGS to select course number, grade and activity date for a one term code and one pidm
	  • `Cursor c2 is select crn, gpa, activity_date from swrregs where pidm = pidm_v and term_code = term_code_v;`
	• Display valid TERMs in drop down HTML FORM Select List
	• Include standard Web Tailor open, close and security calls
	• Register SSB_*yourname*.P_PromptTerm in Web Tailor

Exercise 1 (cont.)

Step	Action
3	Open SSB_*yourname*.sql and add a new procedure in the SSB_*yourname* package named P_DisplayGrades.
4	In P_DisplayGrades: • Add a parameter to accept the form input, term • Use Web Tailor formatting packages to create a HTML TABLE which displays a student grades for the specified term. Use cursor, C2, to loop and display the CRN and GPA. • In the grade HTML table, label the CRN and GPA, Course Number and Grade, respectively • Include standard Web Tailor open, close and security calls • Register SSB_*yourname*.P_DisplayGrades in Web Tailor.
5	To test your form, you will need to login as a person who is in both the SWRREGS table and GOBTPAC. • Use SQLPLUS to find a valid PIDM in the SWRREGS tables • Use SQLPLUS to insert into GOBTPAC and SPRIDEN records for the specified PIDM (ask your instructor for help if you do not have permissions; one valid PIDM should be sufficient for all participants to test) • Login to SSB as a web tailor administrator and create a link under the personal information menu to SSB_*yourname*.P_PromptTerm • Logout and log back into SSB as the test PIDM [from above] and test your new grade form

Section G: Advanced Topics

Lesson: Self Check

Jump to TOC

Introduction

Fully realizing the potential of your Banner web application may involve exploring:

- **PLSQL Tables**
  Allow sets of form data to be sent to a single parameter. The values can then be processed sequentially. In Self Service Banner, PL/SQL tables are used to process form checkbox entries.
- **JavaScript**
  A flexible and powerful scripting language which can enhance the features of your web application. In Self Service Banner, the use of Javascript is limited to avoid compatibility issues between different browsers.
- **Styles and Style Sheets**
  Manage the appearance of your web pages. In Self Service Banner, we provide a default set of style sheets to create a uniform look for all web pages. Style sheets are easy to manipulate and require no programming expertise.

Section Contents

Section G: Advanced Topics

Lesson: PL/SQL Tables

Jump to TOC

Introduction

PL/SQL Tables are logical two-column table structures created in memory during the execution of a PL/SQL program. They can be populated and manipulated in PGA as long as they remain in scope.

For purposes of this discussion, this workbook will cover only the basic syntax and uses of PL/SQL as they apply to Self Service Banner. Specifically, PL/SQL tables are used as variables to capture data from Forms containing check boxes.

Structure

In the example below, only four rows are defined in the PL/SQL table. Since PL/SQL tables are non-continuous memory structures, only four rows are required in memory (unlike C arrays, etc.)

Index	Value
-999	Blue
-1	Red
10	NULL
1000	Green

Syntax

In order to declare a PLSQL table, you first need to define the table type, and then you declare a variable of this type. The index type must be binary_integer.

Example:

```
DECLARE
TYPE  T_CharTab IS TABLE OF varchar2(10) INDEX BY binary_integer;
V_NameTab T_CharTab;
BEGIN
..
```

Once the type and the variable are declared you can refer to an individual element in the PLSQL table by using the syntax
Tablename(index)

Example:

```
V_NameTab(-999) := 'Bob';
A_var        := V_NameTab(-999);
```

The reference can be on either side of an assignment statement. An assignment to an element in a PL/SQL table actually creates this element. If an element is referenced before is has been created, the PL/SQL engine will return the error ORA-1403: no data found.

Table Attributes

To manipulate a PLSQL table, Oracle provides a number of attributes. The syntax is: table.attribute

Attribute	Description
COUNT	Return the number of rows in the table. COUNT
DELETE	Deletes rows in a table. DELETE, DELETE(i), DELETE(i,j)
EXISTS	Returns TRUE if entry exists in table. EXISTS(i)
NEXT	Returns the index of the next existing row in the table. NEXT(i)
PRIOR	Returns the index of the previous row in the table. PRIOR(i)
FIRST	Returns the index of the last row in the table. FIRST
LAST	Returns the index of the last row in the table. LAST

HTML checkboxes

In HTML forms, checkboxes return more than one value per NAME. For example, when asked which states you have visited recently, you may have visisted more than one!

```
<HTML>
<HEAD><TITLE>Form Example</TITLE> </HEAD>
<BODY>
<H2> <font color="blue"> States you have visited </font>  </BR>
<FORM Method=Post ENCTYPE="text/plain"
Action="/pls/trng/process_states">
<input type="checkbox" name="State" value="California"> California
<input type="checkbox" name="State" value="New York">   New York
<input type="checkbox" name="State" value="Indiana">    Indiana
</h2>
</BODY>
</HTML>
```

PLSQL Procedure

To capture data from the FORM, a procedure named *process_states* needs to be created.

```
PROCEDURE process_states
  ( states in twbklibs.varchar2_tabtype)
 IS
   i number;
BEGIN
    FOR i IN states.FIRST TO states.LAST
   LOOP
      IF states.EXISTS(i)
     THEN
        htp.p('You visited ');
        htp.p(states(i));
        htp.p('<BR>');
       END IF;
    END LOOP;
END process_states;
```

Note: Self Service Banner already has the PLSQL type **twbklibs.varchar2_tabtype** defined.

Section G: Advanced Topics

Lesson: JavaScript

Jump to TOC

Introduction

Scripts are programs that add interactivity to your page. Most common scripts are written in JavaScript.

Self Service Banner uses limited Javascript. You can write scripts to add alerts box or bit of text to your page or dynamically change the content of the page as a user interacts with it. Scripting programs can be simple calls to native JavaScript functions or they can be complicated programs stored in the HEAD.

Script types

Two types of scripts:

- **Automated scripts**
  Scripts which execute automatically without user interaction. To fire an "automatic" script, use the <SCRIPT> tag. Since, Javascript is the native scripting language of most browsers, you can omit the <SCRIPT> tag, and directly place Javascript code in your HTML document.
- **Triggered scripts**
  Scripts which execute due to a triggering event. These events are referred to as *intrinsic events.* To fire the script according to an event, one of 18 pre-defined intrinsic events must be called.

Common Intrinsic Events

- ONLOAD
- ONCLICK
- ONMOUSEUP
- ONMOUSEOVER
- ONMOUSEOUT
- ONSELECT
- ONCHANGE

Example 1: Automatic Script

```
<HTML>
<HEAD>
<TITLE>  Using Scripts </TITLE>
</HEAD>
<BODY>
<!- - an automatic script >
  <SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
  document.write('This script works!');
  </SCRIPT>
</BODY>
</HTML>
```

Example 2: Triggered Script (ONCLICK)

```
<HTML>
<HEAD>
<TITLE>  Using Scripts </TITLE>
</HEAD>
<BODY>
<! - - a triggered script>
  <A HREF="time.html" ONCLICK="alert('Today is '+ Date())">time</A>
is it?
</BODY>
</HTML>
```

Example 3: Triggered Script (ONCLICK)

```
<HTML>
<HEAD>
<TITLE>  Using Scripts </TITLE>
</HEAD>
<BODY>
<! - - a triggered script>
 <A HREF ="http://otn.oracle.com"  ONMOUSEOVER="window.status='Click
here to access Oracle's Technology Network"> OTN Web Site </A>

</BODY>
</HTML>
```

Section G: Advanced Topics

Lesson: JavaScript (Continued)

Jump to TOC

Scripting in PL/SQL

Oracle provides a simple PL/SQL toolkit procedure to create the <SCRIPT> tag.

```
   htp.script(  cscript    in    varchar2,
clanguage       in    varchar2 default null);
```

Generates:

```
   <Script Language=clanguage>cscript</Script>
```

Note: This procedure is not necessary because JavaScript is the native scripting language of most browsers. Therefore, you can place JavaScript directly into your HTML document with the htp.p procedure.

Example 1

```
   twbkfrmt.p_printanchor (
            curl          => 'javascript:window.close()',
            ctext         => 'Exit Help',
            cattributes   => 'onMouseOver="window.status=
          '''Close your browser'''; return true" ' );
```

Example 2

```
   twbkfrmt.p_printanchor
        (curl  => skrwurl_rec.skrwurl_content,
         ctext => skrwurl_rec.skrwurl_url_text,
         cattributes      => ' onClick="window.open(' ||
                   CHR (39) ||'http://www.cnn.com'
                   ||CHR (39) ||'); ' ||'return false" '
     );
```

Section G: Advanced Topics

Lesson: Styles and Style Sheets

Jump to TOC

Introduction

Styles are used to change the appearance of your web elements. Before style sheets, visual attributes needed to be managed at an element level. Today, your HTML should just be content – visual effects live in an external style sheet.

Styles can be defined externally, at the document level, or a tag level. (SSB uses externally defined styles.)

You can apply a uniform style to all instances of the one tag in a document. For example, all <P> (paragraph) tags could be left-justified.

Alternatively, you can apply a style to a subset of a tags. For example, every other <P> (paragraph) could be right-justified.

Style Syntax

selector {property: value;}

Parameter	Description
Selector	HTML/element tag
Property	Attribute for HTML tag
Value	Value for property

Example 1

In this example, the BODY element inherits the background color attribute blue.

```
body {bgcolor: blue;}
```

Example 2

In this example, the font-family and color are applied to h1, h2, and h3 headings.

```
h1,h2,h3 {font-family:"sans serif ";color:red}
```

Note: Use commas to group selectors. Use quotes for multiple word values. Use semicolons for more than one property/value style.

Example 3

In this example, the selector has context. The style will only be applied to the *em* elements within a paragraph <P> tag.

```
para em {color: blue;}
```

Example 4

In this example, the selector applies to a class *class1*. All *em* elements belonging to the class *class1* will be affected.

```
em.class1{ color: blue;}
```

Example 5

In this example, the selector applies to only one divide element - identified by the id *Picasso*. Since ids are unique in an HTML document, only one element is affected.

```
A#picasso {color:blue}
```

Example 6

In this example, we include classes in our style sheet to control how elements are aligned. Notice how the class *center* can be applied to any element.

My_style_sheet.css

```
p.right {text-align: right}
p.left {text-align: left}
.center {text-align: center}
```

-- omit the selector clause to allow the style to be called by any tag

My_web_page.html

```
<p class="right"> This paragraph will be right-aligned. </p>
<p class="left"> This paragraph will be left-aligned. </p>
<h1 class="center"> This heading will be center-aligned </h1>
```

Reference a Style Sheet

The external sheet used by a web page is defined by Web Tailor parameter associated with the procedure. That means you do not need to call a specific style sheet in your code.

Default style sheets are called if no style sheet is explicitly associated with a procedure in Web Tailor.

```
<LINK REL="stylesheet" HREF="/css/web_defaulthome.css"
TYPE="text/css">
<LINK REL="stylesheet" HREF="/css/web_defaultprint.css"
TYPE="text/css" media="print">
```

A class may differ based on the state of the HTML object. For example, the submenulinktext2 class is used for links (<A> tags) For example, if a user has clicked on the link then the color changes:

```
A.submenulinktext2 {
color:       #1E2B83;
text-decoration: none;}
A.submenulinktext2:visited {
color: #800080;
text-decoration: none;}
```

Specifying Styles in Banner

The P_PrintStrong, P_PrintMenu, P_PrintText procedures have a "class_in" parameter which allow you to specify a new style class for a tag. The rest of the formatting procedures call a style sheet internally. Therefore, you can only change the style properties by modifying the existing style.

twbkfrmt.p_ PrintText

This generates text with a specified class. Otherwise, the default font size will be 3.

```
Twbkfrmt.P_PrintText (
      text       IN   VARCHAR2 DEFAULT NULL,
      class_in   IN   VARCHAR2 DEFAULT NULL
   );
```

Parameter	Description
text	Character string to display.
class_in	Optional class –defined in referenced style sheet.

twbkfrmt.p_ PrintStrong

This generates bolded text with a specified class using the <STRONG> element.

```
Twbkfrmt.P_PrintText (
      text       IN   VARCHAR2 DEFAULT NULL,
      class_in   IN   VARCHAR2 DEFAULT NULL
   );
```

Parameter	Description
text	Character string to display.
Class_in	Optional class – defined in referenced style sheet.

Example

```
web_defaultapp.css
.fieldOrangetextbold { color: ORANGE; font-family: Arial Narrow, verdana,
helvetica, sans-serif; font-weight: bold; font-size: 100%; font-style:
normal; text-align: left;}

your_code.sql

 twbkfrmt.P_TableDataHeader
     (twbkfrmt.f_printtext (
               robinst_rec.robinst_aidy_desc,
               class_in => 'fieldOrangetextbold'
               )
      ,ccolspan=>'10'
      );
```

Section G: Advanced Topics

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self-check activity.

Exercise 1: Checkbox input

Modify SSB_*yourname*.P_PromptTerm procedure to use checkboxes, not a drop down, to display each **term**.

Modify SSB_*yourname*.P_DisplayGrades to use a PL/SQL Table to capture more than one term submitted by the form.

Hint: Use twbklibs.varchar2_tabtype as the data type for the term parameter.

Exercise 2: JavaScript

Modify SSB_*yourname*.P_DisplayGrades to make the Course Number column a link to a new window displaying a description of the course.

Hint: Use the JavaScript function Window.Open() to launch the window "on click."

Exercise 3: Style sheet

Create your own copy of web_defaulthelp.css in the web server document root directory (you will need permission and a login from your DBA).

Name the new style sheet web_*yourname.css.*

Associate your new Course Description page (created in the previous exercise) with this style sheet.

Alter the appearance to your liking by making changes to web_*yourname.css* on the web server. For example, try changing the background color of page and re-size the window to a better size.

Hint: To associate your new window with a style sheet, you can *register* your new procedure in Web Tailor and define the location of the style sheet.

Section H: PL/SQL Web Toolkit Reference / Sample Packages

Lesson: Overview

Jump to TOC

Overview

This section contains references for the PL/SQL Web Toolkit, along with a sample specification and body for HELLO_WORLD and SSB_YOURNAME.

Section Contents

HTML, HEAD, and BODY Tags

```
htp.htmlOpen, htp.htmlClose - generate <HTML> and </HTML>
htp.headOpen, htp.headClose - generate <HEAD> and </HEAD>
htp.bodyOpen, htp.bodyClose - generate <BODY> and </BODY>
```

Comment Tag

```
htp.comment - generates <!-- and -->
```

Tags in the <HEAD> Area

```
htp.base    - generates <BASE>
htp.linkRel - generates <LINK> with the REL attribute
htp.linkRev - generates <LINK> with the REV attribute
htp.title   - generates <TITLE>
htp.meta    - generates <META>
htp.script  - generates <SCRIPT>
htp.style   – generates <STYLE>
htp.isindex - generates <ISINDEX>
```

Applet Tags

```
htp.appletopen, htp.appletclose
       - generate <APPLET> and </APPLET>
htp.param - generates <PARAM>
```

List Tags

```
htp.olistOpen, htp.olistClose - generate <OL> and </OL>
htp.ulistOpen, htp.ulistClose - generate <UL> and </UL>
htp.dlistOpen, htp.dlistClose - generate <DL> and </DL>
htp.dlistTerm                 - generates <DT>
htp.dlistDef                  - generates <DD>
htp.dirlistOpen, htp.dirlistClose
                            - generate <DIR> and </DIR>
htp.listHeader                - generates <LH>
htp.listingOpen, htp.listingClose
                            - generate <LISTING>
                              and </LISTING>
htp.menulistOpen, htp.menulistClose
                            - generate <MENU>
                              and </MENU>
htp.listItem                  - generates <LI>
```

Form Tags

```
htp.formOpen, htp.formClose
              - generate <FORM> and </FORM>
htp.formCheckbox – generates <INPUT TYPE="CHECKBOX">
htp.formHidden   - generates <INPUT TYPE="HIDDEN">
htp.formImage    – generates <INPUT TYPE="IMAGE">
htp.formPassword - generates <INPUT TYPE="PASSWORD">
htp.formRadio    – generates <INPUT TYPE="RADIO">
htp.formSelectOpen, htp.formSelectClose
              - generate <SELECT> and </SELECT>
htp.formSelectOption - generates <OPTION>
htp.formText     – generates <INPUT TYPE="TEXT">
htp.formTextarea, htp.formTextarea2
              – generate <TEXTAREA>
htp.formTextareaOpen, htp.formTextareaOpen2, htp.formTextareaClose
              - generate <TEXTAREA> and </TEXTAREA>
htp.formReset    - generates <INPUT TYPE="RESET">
htp.formSubmit   - generates <INPUT TYPE="SUBMIT">
```

Table Tags

```
htp.tableOpen, htp.tableClose
              - generate <TABLE> and </TABLE>
htp.tableCaption - generates <CAPTION>
htp.tableRowOpen, htp.tableRowClose
              - generate <TR> and </TR>
htp.tableHeader  - generates <TH>
htp.tableData    - generates <TD>
htf.format_cell  - generates <TD>
```

IMG, HR, and A Tags

```
htp.line, htp.hr - generate <HR>
htp.img, htp.img2 - generate <IMG>
htp.anchor, htp.anchor2 - generates <A>
htp.mapOpen, htp.mapClose - generate <MAP> and </MAP>
```

Paragraph Formatting Tags

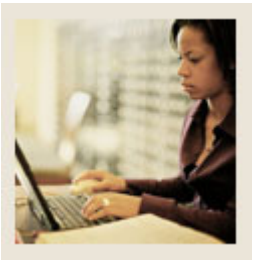
```
htp.header       - generates heading tags (<H1> to <H6>)
htp.para, htp.paragraph - generate <P>
htp.print, htp.prn - generate any text that is passed in
htp.prints, htp.ps - generate any text that is passed in;
                 special characters in HTML are escaped
htp.preOpen, htp.preClose – generate <PRE> and </PRE>
htp.blockquoteOpen, htp.blockquoteClose
              - generate <BLOCKQUOTE> and </BLOCKQUOTE>
htp.div          - generates <DIV>
htp.nl, htp.br   - generate <BR>
htp.nobr         - generates <NOBR>
htp.wbr          - generates <WBR>
htp.plaintext    - generates <PLAINTEXT>
htp.address      - generates <ADDRESS>
htp.mailto       - generates <A> with the MAILTO attribute
htp.area         - generates <AREA>
htp.bgsound      - generates <BGSOUND>
```

Character Formatting Tags

```
htp.basefont   - generates <BASEFONT>
htp.big        - generates <BIG>
htp.bold       - generates <B>
htp.center     - generates <CENTER> and </CENTER>
htp.centerOpen, htp.centerClose
            - generate <CENTER> and </CENTER>
htp.cite       - generates <CITE>
htp.code       - generates <CODE>
htp.dfn        - generates <DFN>
htp.get_download_files_list - generate <EM>
htp.fontOpen, htp.fontClose
            - generate <FONT> and </FONT>
htp.italic     - generates <I>
htp.keyboard, htp.kbd – generate <KBD> and </KBD>
htp.s          - generates <S>
htp.sample     - generates <SAMP>
htp.small      - generates <SMALL>
htp.strike     - generates <STRIKE>
htp.strong     - generates <STRONG>
htp.sub        - generates <SUB>
htp.sup        - generates <SUP>
htp.teletype   - generates <TT>
htp.underline  - generates <U>
htp.variable   - generates <VAR>
```

Frame Tags

```
htp.frame
           - generates <FRAME>
htp.framesetOpen, htp.framesetClose
           - generate <FRAMESET> and </FRAMESET>
htp.noframesOpen, htp.noframesClose
           - generate <NOFRAMES> and </NOFRAMES>
```



Section H: PL/SQL Web Toolkit Reference / Sample Packages

Lesson: Hello_World Package Specification

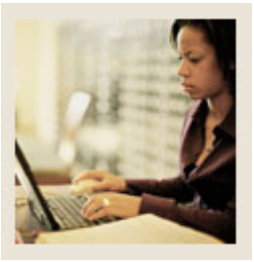
Jump to TOC

Code

```
CREATE OR REPLACE PACKAGE Hello_World_Chris
IS
Procedure P_DisplayHello;
Procedure P_DisplayVacation;
Procedure P_ProcessVacation(vacation_v in varchar2);
END Hello_World_Chris;
/
```

Code

```
CREATE OR REPLACE PACKAGE BODY Hello_World_Chris
 IS
 Procedure P_DisplayHello
    IS
  cursor C1 is select * from SWBADDR;
     BEGIN
      HTP.P('Hello World');
      htp.para;
      htp.comment('below is exercise 422');
      HTP.ulistOpen;
      HTP.listHeader('Street Line');
       FOR my_rec IN C1 LOOP
           EXIT WHEN C1%NOTFOUND;
           HTP.listItem(my_rec.street_Line1);
         END LOOP;
      HTP.ulistClose;

   END P_DisplayHello;
```



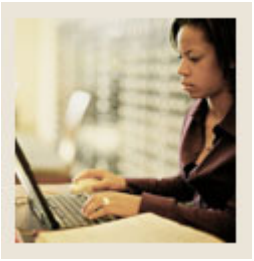

```
---------------------------------------------- Vacation Form
        Procedure P_DisplayVacation
          IS
          BEGIN
Htp.FormOpen('/pls/C5X/hello_world_chris.P_ProcessVacation','POST');
              http.tableOpen;
               http.tableRowOpen;
                  http.tableData(htf.bold('Where do you like to ski?'));
                  -- put your drop select box here
                  http.p('<TD>');

   http.formSelectOpen('vacation_v','Select your      vacation');
                  http.formSelectOption('Alps');
                  http.formSelectOption('Canadian Banff National Park');
                  http.formSelectOption('Colorado Rockies');
                  http.formSelectClose;
                  http.p('</TD>');
               http.tableRowClose;
               http.tableClose;
              http.formSubmit(null,'Win a free trip on SCT');
              http.formClose;
       End P_DisplayVacation;

      Procedure P_ProcessVacation(vacation_v in varchar2)
        IS

        BEGIN

          http.para;
          http.header(1,'Your Favorite Skiing Destination');
          http.hr;
          http.para;
          http.p(vacation_V);
      END;
    END Hello_World_Chris;
    /
```

Code

```
create or replace package ssb_chris
IS

procedure p_displaycourse;
procedure p_promptterm;
procedure p_displaygrades(term varchar2);

End ssb_chris;
```

Code

```
CREATE OR REPLACE PACKAGE BODY ssb_chris
IS
procedure p_displaycourse
  IS
    cursor c2 is select * from swvcrse;
    pidm number;
  BEGIN
      -- top of document + information text

if not twbkwbis.F_ValidUser(pidm) then  return;
     end if;
   twbkwbis.P_OpenDoc('ssb_chris.p_displaycourse');
        twbkwbis.P_DispInfo
('ssb_chris.p_displaycourse','DEFAULT');

        -- display table caption and headers
       twbkfrmt.P_TableOpen('DATADISPLAY',ccaption => 'Course
Information');
      twbkfrmt.P_TableRowOpen;
          twbkfrmt.P_TableHeader('Course Number');
          twbkfrmt.P_TableHeader('Description');
          twbkfrmt.P_TableHeader('Credit Hours');
          twbkfrmt.P_TableHeader('Date');
      twbkfrmt.P_TableRowClose;
        FOR course_rec IN C2  LOOP
         -- display Table Data
            twbkfrmt.P_TableRowOpen;
                twbkfrmt.P_TableData(course_rec.crn);
     twbkfrmt.P_TableData(course_rec.description);
```



Section H: PL/SQL Web Toolkit Reference / Sample Packages

Lesson: SSB_Yourname Package Specification (Continued)

Jump to TOC

```
      twbkfrmt.P_TableData(course_rec.credit_hours);
                 twbkfrmt.P_TableData(course_rec.activity_date);
            twbkfrmt.P_TableRowClose;
          END LOOP;
       -- close table
        twbkfrmt.P_TableClose;
     -- close document
      twbkwbis.P_CloseDoc('5.1');
      END p_displaycourse;
    procedure p_promptterm     IS
         cursor c2 is select distinct term_code from trainXX.swrregs;
         pidm number;
     BEGIN

    if not twbkwbis.F_ValidUser(pidm) then  return;
         end if;
      twbkwbis.P_OpenDoc('ssb_chris.p_promptterm');
           twbkwbis.P_DispInfo ('ssb_chris.p_promptterm','DEFAULT');
     htp.formOpen('/pls/scunc/ssb_chris.p_displaygrades', 'post');
     twbkfrmt.P_TableOpen('DATAENTRY');
          twbkfrmt.P_TableRowOpen;
          twbkfrmt.P_TableDataLabel('Choose a Term', 'left');
             twbkfrmt.P_TableDataOpen;
           htp.formSelectOpen('term', 'Choose a Term');
                  FOR term_v IN C2  LOOP

                IF term_v.term_code = '199601' THEN
                 twbkwbis.P_FormSelectOption(term_v.term_code, null,
           'SELECTED');
                ELSE
                  twbkwbis.P_FormSelectOption(term_v.term_code,
term_v.term_code);
                END IF;
            END LOOP;
          htp.formSelectClose;
                twbkfrmt.P_TableDataClose;
          twbkfrmt.P_TableRowClose;
          twbkfrmt.P_TableClose;
       htp.formsubmit;
       htp.FormClose;
      twbkwbis.P_CloseDoc('5.1');
     END p_promptterm;
```

Section H: PL/SQL Web Toolkit Reference / Sample Packages

Lesson: SSB_Yourname Package Specification (Continued)

Jump to TOC

Code (cont.)

```
---------------------------------------- P_DISPLAYGRADES ----------------------------------
    procedure p_displaygrades (term varchar2) IS
     cursor c2(pidm_v number, term_v varchar2)  is
          select * from swrregs
           where pidm_v = pidm
             and  term_v = term_code ;
    pidm number;
      BEGIN
    if not twbkwbis.F_ValidUser(pidm) then  return;
         end if;
       twbkwbis.P_OpenDoc('ssb_chris.p_displaygrades');
            twbkwbis.P_DispInfo
('ssb_chris.p_displaygrades','DEFAULT');

           -- display table caption and headers
         twbkfrmt.P_TableOpen('DATADISPLAY',ccaption => 'Your Grades');
        twbkfrmt.P_TableRowOpen;
            twbkfrmt.P_TableHeader('Course Number');
            twbkfrmt.P_TableHeader('GPA');
        twbkfrmt.P_TableRowClose;
          FOR gpa_rec IN C2(pidm, term)  LOOP
           -- display Table Data
              twbkfrmt.P_TableRowOpen;
                  twbkfrmt.P_TableData(gpa_rec.CRN);
                  twbkfrmt.P_TableData(gpa_rec.GPA);
                  twbkfrmt.P_TableRowClose;
          END LOOP;
       -- close table
        twbkfrmt.P_TableClose;
     -- close document
      twbkwbis.P_CloseDoc('5.1');
    END p_displaygrades;
    END ssb_chris;
```

Release Date

Jump to TOC

This workbook was last updated on 10/172005.