Developer Tutorial

Release 5.0.2

www.*TURBO*-Enterprise.com

TURBO Enterprise Release 5.0.2, Developer Tutorial

Table of Contents

About This Guide

The TURBO Enterprise Developer Tutorial provides you with information and step-by-step instructions to create a complete Rich Web Application using the TURBO Enterprise software. Using the tutorial, we will build a basic Sales application called *Pipeline* with a simple form input page for collecting customer information. We will use the Application Wizard to create the application called *Pipeline*.

Audience
This guide is intended for PL/SQL developers who want to create Rich Web Applications for the enterprise.

Downloads
You can download a trial version of TURBO Enterprise 5.0.2 at www.TURBO-enterprise.com, by selecting the 'Product' tab, and clicking on 'Download'. To access the link directly from this document, click here: Downloads.

Document Conventions
A greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path.
A trademark symbol (®, TM, etc.) denotes a company trademark. An asterisk (*) denotes a third-party trademark.
An underlined word or phrase denotes an internal hyperlink or cross reference.
The `Courier New` font is used to denote variables, parameters, file and script names, and commands.

What's New

TURBO Enterprise Release 5.0.2 offers several new features and key enhancements to various components. These enhancements improve management, and enhance security:

- Oracle Forms Conversion utility: a powerful tool to convert your unsupported client/server Oracle Forms to a supported, Web-based PL/SQL environment.
- The File System: a complete file management and version control repository, with drag and drop features, 'on-demand' loading capabilities and the option to save default navigation tree settings.
- The Calendar: an integrated, customizable, event tracking and scheduling tool.
- Enterprise Social Networking Tools: tools that enable secure corporate communication, and streamline team collaboration and productivity. Example: the Wall, Buddy List, Instant Messaging.
- Application Builder: an easy way to start your new application directly from the user interface with just one click. You also have the option of adding a File System and Calendar to your application when you create it.

To learn more about these features, refer to the list of additional documents provided in the section Additional Resources.

Product Overview

Rich Internet Applications (RIAs)—familiar to us in the latest web interfaces such as Google and Yahoo—offer superior user interfaces for applications. They are more intuitive to use and more user-friendly, typically pulling data onto a single page and requiring fewer if any “click and wait” situations where data is submitted and processed.
The Enterprise has been slow to adopt the latest web technologies because of security and architecture concerns. Access control is difficult; different development skills are commonly required to create these applications; and common development tools such as AJAX, used indiscriminately, leave back doors open to the back-end system. AJAX (asynchronous JavaScript and XML) is a group of interrelated web development techniques used for creating interactive web applications—RIAs. The benefit of AJAX is that by using it web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page—in other words, providing dynamic page content without a page refresh. Data is retrieved using the `XMLHttpRequest` object or through the use of Remote Scripting in browsers that do not support it. Figure 1 shows some examples of Rich Web Applications.

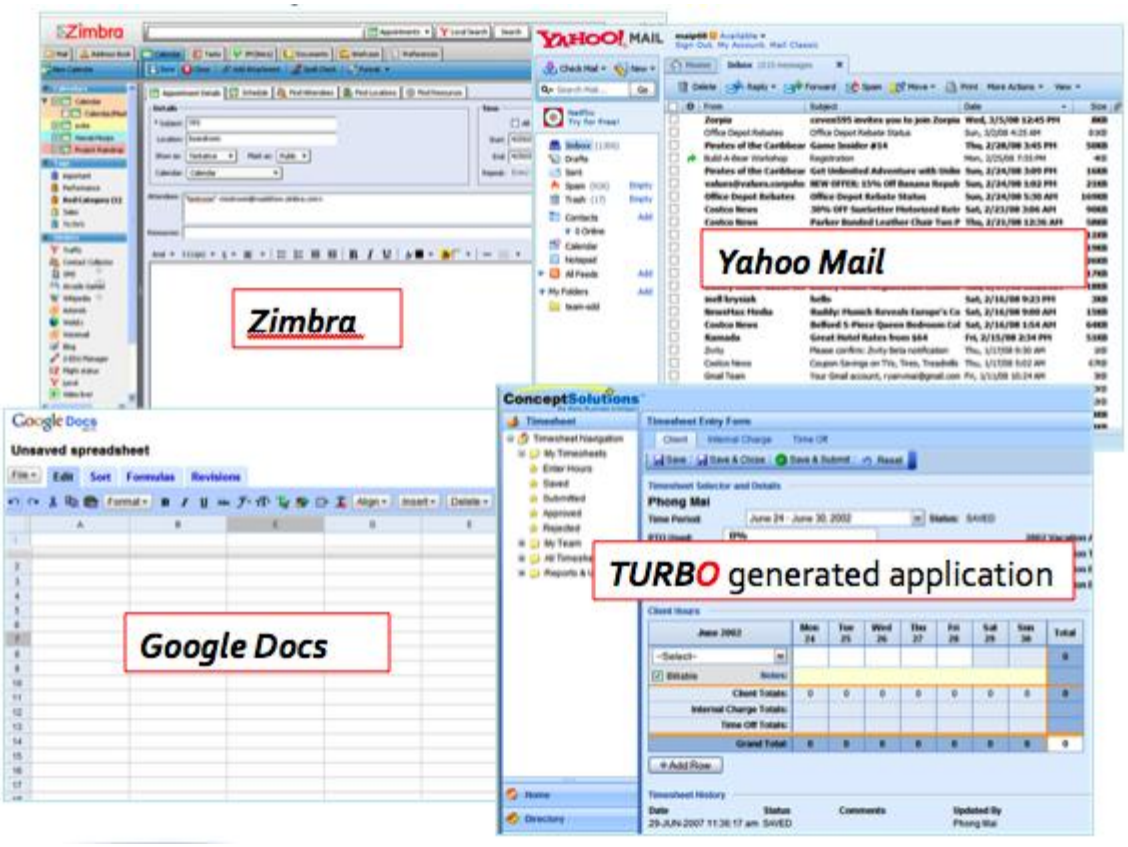


Figure 1 "Rich Web Applications"

AJAX, then, is the best solution for delivering RIA because of its open standards and its ability to be integrated into the Oracle database. Using it allows users to spend less time waiting for

data to be transmitted and less time completing tasks. AJAX loads only what a particular page needs, and presents users with a familiar interface, reducing the time needed to learn new applications. Applications and users both respond better.

Thus, developing an RIA gives users the interactivity they expect from desktop applications and increases Enterprise productivity. There are reasons, however, that RIAs are not often used for Enterprise applications. First, different sets of development knowledge are required—JavaScript developers rather than Oracle developers are the primary creators of RIAs—meaning that when AJAX applications are built they often don't take full advantage of database features. Second, AJAX has specific performance and security issues that require a high level of development expertise: browser page freeze is a common problem, and AJAX can create security holes, making it difficult for the Enterprise architecture to adhere to specific business use rules. In addition, many AJAX offerings are just "bags of code," requiring specialized knowledge to put them together into a functioning application—and this complexity often requires developers to relearn development and application coding and techniques and does not prove efficient or cost-effective. Virtually every Enterprise application is connected to a database, and Oracle is present in virtually every Enterprise. As the top-performing database, Oracle's powerful features are typically overlooked, however, and it is often used as a simple data repository because of the difficulty of building a user-friendly application to take full advantage of its power. Database processing, though, can add significant performance to Enterprise applications that must conform to specific business rules and that require large amounts of data processing. Database processing improves security, simplifies deployment, simplifies maintenance and backups, and improves application performance. Bizwhazee created TURBO—an RIA development platform for Oracle—to connect the benefits of database processing with the benefits of AJAX and RIAs.

TURBO Enterprise is an application development framework and platform for building Rich Web Applications (RWAs) using Oracle's Procedural Language/Structured Query Language (PL/SQL).
New technologies are often viewed as 'disruptive' because they introduce unproven technology, change how things are done, and require new types of technical specialists. TURBO Enterprise turns 'disruptive technology' into a 'disruptive approach using stable technology'. Rather than introduce new unfamiliar technology, TURBO Enterprise makes greater use of the technology you already know: the Oracle database. This enables you to make better use of time and money by leveraging your current staff skills, hardware infrastructure, and the existing investment in your Oracle database. Using open standards and methodologies, TURBO Enterprise uses Oracle as a 'thick database'. The entire application, including business logic, data processing and User Interface rendering, is served from within the database. This 'technology efficiency' eliminates middle-tier processing, reducing overhead and bandwidth requirements, and increasing application efficiency. Enterprises can now reap all the benefits of the latest Web technologies to produce RWAs, while adhering to the security and performance needs of an enterprise environment. The RWAs are easy to deploy and provide a richer user experience.

Architecture

TURBO resides within the Oracle database and has a compiled, virtual, three-tier system architecture to contain the components needed to build and run Enterprise solutions: security, user interface rendering, stored business logic, and optimized performance. The system architecture is database-centric, and relies on the power and performance of PL/SQL. Data is separated from the application by setting up specific layers in which each resides and operates. For detailed information on TURBO Enterprise architecture, refer to the *TURBO Enterprise Resource Guide* at www.turbo-enterprise.com.

TURBO Enterprise follows Model-View-Control (MVC) design principles. MVC ensures that data and application code are separated. For example, data from a web form is never passed directly to the DATA layer packages and the 'Execute' privilege is never granted from the DATA layer packages directly to the APP_USER account. Because TURBO's architecture is focused on security, we do not advocate making any of the DML packages accessible directly from the web. Separation prevents intentional or accidental data manipulation within your database.

Developing in TURBO Enterprise

TURBO Enterprise allows you to build a complete portal of web applications. Developing is easy: a complete library of Application Programming Interfaces (APIs) has been built into TURBO. All you need to do is call these APIs using PL/SQL. This tutorial will show you how to use the TURBO Application Wizard to build a TURBO web page and will then walk you through building a basic Form for data entry, loading data into a Form, saving data from a Form, and loading data to the table. You will also learn how to build a grid to display your data in tabular form, add filters, and generate charts and graphs to display your data.

The four main parts to develop in TURBO are:

- ✓ Render
- ✓ Retrieve
- ✓ Receive
- ✓ DML

These key parts (which we will refer to as 'RRRD') will help you understand the basics of TURBO's architecture, data security, and coding efficiency. The following sections show you how you can use the RRRD concept to render new fields in the form, retrieve data and display it in your form, receive and pass the data from these fields to the Data Manipulation Layer (DML), and save data to the database. Figure 2 "Developing in TURBO: RRRD" depicts TURBO's four main parts in a workflow.

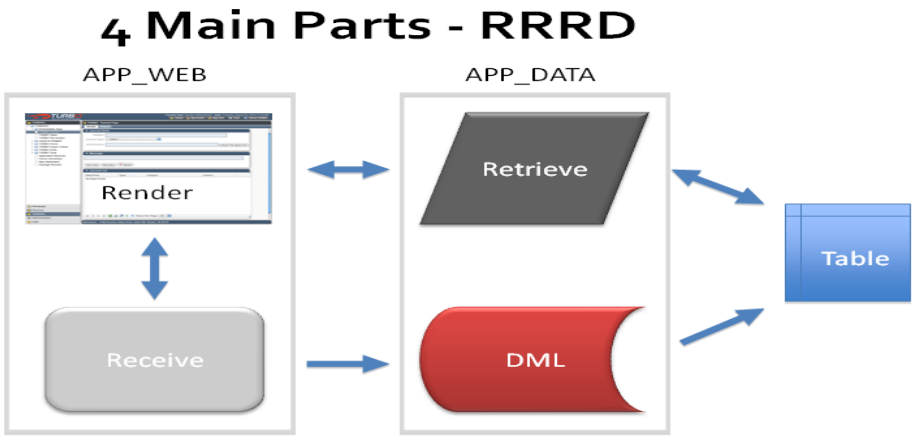


Figure 2 "Developing in TURBO: RRRD"

This tutorial will show you how to use:

- The Toolbar – a special menu of page-specific actions (e.g., Save page, Save & Close, Print, etc.)
- Tabs – an efficient way to organize your information by tabbed pages so that end-users can access all information relevant to the module.
- Fieldgroups – a way to organize related information within a single page.
- TURBO field elements – an extensive list of APIs that you can see, interact with, and use as source code. Examples: text input, AJAX auto-complete, date, phone.

Pre-requisites: ensure you have installed TURBO Enterprise 5.0.2 and your desired SQL editing software (For example, SQLNavigator, SQL Developer, or TOAD) installed.

Application and Package Setup

1. After installing TURBO Enterprise, log in to your new website.

2. Navigate to the **New Application** page in the built-in TURBOKit application.

3. Enter a name for your application in the **Application Name** field. (In this case, enter *Pipeline*).

4. Select from the list of available items for features you can add to your application. For this tutorial select Tabs, Toolbar, and Form.

5. Click **Submit** to generate your new application.

   The Application Wizard will create all the necessary packages in the GUI and DATA/DML layers, along with the appropriate database grants and synonyms, and an application-specific and form-specific access control list with default profiles and privileges that you can edit. You will see a notification page indicating all the steps completed by the Application Wizard.

6. Refresh your page to see your new application, *Pipeline*.

Application and Page Security

TURBO Enterprise handles application security in two ways. The application layer is separated from the login layer (where there is access to the Internet) and from the data layer. Within the application, security is handled by an Access Control List (ACL)-based security model that controls user access to various parts of the application. Every object, node, folder, or sub-folder you create within TURBO as part of your application has an ACL associated with it. An ACL is a list of permissions attached to an object. An ACL specifies which users or system processes are granted access to objects, as well as what operations are allowed on specified objects. Each entry in a typical TURBO ACL specifies a profile, a privilege, and a member. The profile with a

list of associated members is the 'Subject', and the privilege is an 'operation' or an activity that is performed. After you create your application, TURBO creates an ACL for any new object that you add (Example: new web page).

Build Your Page

Follow these steps to build a basic page. The steps to build a page also show you the basic overall procedure for developing with TURBO Enterprise:

1. Log in to the `APP_WEB` schema through your SQL editor, and navigate to your application's default package, created by the Application Wizard. Example: for the *Pipeline* application, the package will be named `L_PIPELINE`.

2. Open the package body (`L_PIPELINE_BODY`) and select the procedure named `p_XXX_form`, where `xxx` is the name of the application you specified in the Application Wizard. You will see that some default code already exists in the procedure.

Note: procedure names are prefixed with '`p_`', function names are prefixed with '`f_`'.

```
PROCEDURE p_pipeline_form
/*****************************************************************************
-- Name:                p_pipeline_form
-- Purpose:             procedure to generate the Form page
-- Developer:           System Administrator
-- Create Date:         09-AUG-10
-- Modification History:
--
--  Date        Developer Name      Description
-- 09-AUG-10 System Administrator Version 1.0
```

3. Modify the starter set of parameter fields provided. You can use this set of fields to dynamically interact with your form. You can also add more parameters as you need them.

The page initialization procedure (`te_gui_utils.p_page_init`), which is a part of your form's procedure, is the template that generates the header, application buttons, navigation tree, and footer sections. The Application Wizard has already filled in some of the parameters you'll need to get started:

```
te_gui_utils.p_page_init(
    i_page_title    => 'Pipeline Form',
    i_rec_id        => '',
    i_inactive_flg  => '',
    i_save_close    => '',
    i_version_id    => '',
    i_tab           => '1',
    i_redirect      => '',
    i_form_proc     => 'p_pipeline_proc',
```

```
i_toolbar => TRUE);
```

Table 1 "Page Initialization Parameters" lists the parameters in the procedure `te_gui_utils.p_page_init`, along with their descriptions.

Table 1 "Page Initialization Parameters"

Input Parameter	Description
`i_page_title`	Sets the page title directly above the form
`i_rec_id`	The ID or record value of the entity that will be displayed in this form
`i_inactive_flg`	Allows you to specify if the form is active or inactive
`i_save_close`	Allows you to save and close the form
`i_version_id`	A generic field used to pass in a version ID
`i_tab`	A generic field that allows you to specify which tab to set focus on when displaying the page.
`i_redirect`	
`i_form_proc`	Indicates where the form will send the data when the end-user clicks the Save or Save and Close button in the toolbar.
`i_toolbar`	Indicates if a toolbar will be displayed on the page

4. Modify the default toolbar buttons, according to your application's needs. The Application Wizard provides some basic toolbar buttons to perform the 'Save', 'Save and Close', and 'Reset' functions.

```
te_navigation.p_toolbar_add('Save', 'save.gif', te_inputs.f_save);
te_navigation.p_toolbar_add('Save & Close', 'save.gif',
                            te_inputs.f_save(i_save_close => TRUE));
te_navigation.p_toolbar_add('Reset', 'undo.gif', te_inputs.f_reset);
```

5. Add tabs according to your application's needs. The Application Wizard provided one tab, named 'Tab 1,' for this tutorial page. You can edit the name to something more meaningful to your application, such as 'Customer'. Keep tab names short if you plan on dividing your page into multiple tabs.

```
te_navigation.p_tab_add('Tab 1');
```

6. Add input fields. The application wizard provided the following Page Start and Page End procedures:

```
te_gui_utils.p_page_start;
te_gui_utils.p_page_end;
```

After starting the page with `te_gui_utils.p_page_start` you can fill the page with form elements or input fields, using the following block of code:

```
te_navigation.p_tab_div_open(1);

te_inputs.p_fieldgroup_open(800, 'Input Fields',100);

-- Add your input fields

te_inputs.p_fieldgroup_close;
```

Note: Repeat this block of code to add multiple tabs on your page. Be sure to include a call to the procedure `te_navigation.p_tab_add()` before `p_page_start()` for each tab you want to display. For reusable tabs (Example: to reuse an Employee Contact tab in the Directory application, in a Human Resources application that you may want to build, and on the My Profile page), we encourage using procedures for each tab so that you can call these procedures.

Customize Your Page

Follow these steps to build and customize your page:

1. In the procedure `p_pipeline_form`, change the name of the field group to something more meaningful. Example: 'Customer Information'.

```
te_inputs.p_fieldgroup_open(
      i_width       => 800,
      i_label       => 'Customer Information',
      i_label_width => 100);
```

All form elements in this group will inherit a label width of 100. However, a new label width can be set in each form element as well.

2. Select **TURBOKit > TURBO Forms**.

3. Select **Input Fields > Text Input**.

   This is a simple Text input field we will use.

4. Click the 'Source' tab to view and copy the source code used to generate this field.

```
te_inputs.p_text(
       i_name => 'i_first_name',
       i_value      => 'John',
       i_label      => 'Name:',
       i_lab_hint   => 'User Name entry fields',
       i_size => 10,
```

```
        i_maxlength   => 15,
        i_newline     => FALSE,
        i_required    => TRUE,
        i_readonly    => FALSE,
        i_desc => 'First',
        i_desc_hint   => 'User''s first name');
```

Note: Copy the code beginning with the package and procedure name, with all the parameter fields, down to the closing parenthesis and semi-colon.

5. Paste the code in your procedure, between the `Fieldgroup Open`, and `Fieldgroup Close` statements, and save the package.

6. Click the *Pipeline* link from your browser to reload the page and view your new field element.

Note: There are many parameters available for the `p_text` form field, as there are for all of the TURBOKit Form Field APIs. These parameters allow for very simple control of information to be displayed around the form field. See the 'Form Fields' section of this Tutorial for more information about how to use the power of these parameters to control how your information is displayed.

Add Fields to Your Page

Follow these steps to add more fields to the same form:

1. Insert the `te_inputs.p_text` source code again.

2. Make changes to the values being passed to the input parameters, as follows:

```
te_inputs.p_text(
        i_name => 'i_last_name',
        i_value       => 'Smith',
        i_size => 10,
        i_maxlength   => 15,
        i_newline     => TRUE,
        i_required    => TRUE,
        i_readonly    => FALSE,
        i_desc => 'Last',
        i_desc_hint   => 'User''s last name');
```

Note: for simplicity, we will use hard-coded values here. The next section will show you how to populate these fields with live data.

3. From the TURBOKit Forms/Input Fields folder, copy the source code for Phone Input.

4. Paste the code into your procedure and modify as follows:

```
te_inputs.p_phone(
i_name      => 'i_phone',
i_lab_width => NULL,
i_value     => '1800555121254321',
i_newline   => TRUE,
i_required  => TRUE,
i_label     => 'Phone:');
```

5. From the TURBOKit Forms/Input Fields folder, copy the source code for Email Input.
6. Paste this code into your procedure and modify as follows:

```
te_inputs.p_email(
       i_name      => 'i_email',
       i_lab_width => NULL,
       i_value     => 'john.smith@test.com',
       i_size      => 45,
       i_newline   => TRUE,
       i_required  => TRUE,
       i_label     => 'Email Address:');
```

7. From the TURBOKit Forms/Input Fields folder, copy the source code for Date Input.
8. Paste this code into your procedure and modify as follows:

```
te_inputs.p_date(
       i_name      => 'i_date',
       i_lab_width => NULL,
       i_value     => SYSDATE,
       i_date_mask => 'FMMonth DD, YYYY',
       i_newline   => TRUE,
       i_label     => 'Birthday:');
```

9. From the TURBOKit Forms/Text Area folder, copy the source code for Text Area.
10. Paste this code into your procedure and modify as follows:

```
te_inputs.p_textarea(
       i_name    => 'i_notes',
       i_value   => 'Enter customer notes',
       i_newline => TRUE,
       i_rows    => 7,
       i_cols    => 50,
       i_label   => 'Customer Notes:');
```

11. Now close out the Customer Information fieldgroup with the following line:

```
te_inputs.p_fieldgroup_close;
```

Your form should now generate a new page with all the fields you added.

You've created your first TURBO Enterprise web form. We'll tighten up the layout in a later step.

Feel free to play around with the parameter values for the Fieldgroup, as well as the individual form fields. For example, in `p_fieldgroup_open`, change the value for `i_label_width` to 200 and see what happens to your form. Also, try adding these parameters to any of the form fields:

```
i_disabled  => TRUE,
i_readonly  => TRUE,
i_required  => TRUE,
i_desc      => 'Place text above field',
i_desc_hint => 'Pop up text when mouse-over field',
```

Create Table and Data

TURBO Enterprise follows the MVC design principle to keep the application and UI code separate from the data. Therefore, we will create the 'Customer' table, as well as the functions to retrieve the data, in the `APP_DATA` schema. This data will then be passed to the `APP_WEB` schema for presentation in the browser. Follow these steps to create the 'Customer' table and load data:

1. Run the following script:

```
CREATE TABLE customer
    (customer_id                NUMBER NOT NULL,
     first_nm                   VARCHAR2(200),
     last_nm                    VARCHAR2(200),
     birthday_dt                DATE,
     phone                      VARCHAR2(50),
     email                      VARCHAR2(200),
     aim_name                   VARCHAR2(16),
     addr_1                     VARCHAR2(200),
     addr_2                     VARCHAR2(200),
     city                       VARCHAR2(200),
     state                      VARCHAR2(2),
     zip                        VARCHAR2(10),
     company                    VARCHAR2(50),
     web_page                   VARCHAR2(250),
     notes                      CLOB ,
     version_id                 NUMBER,
     load_dt                    DATE,
     load_user_id               NUMBER,
     update_dt                  DATE,
     update_user_id             NUMBER,
     hide_cont                  CHAR(1) DEFAULT 'N',
     inactive_flg               CHAR(1) DEFAULT 'N',
     inactive_dt                DATE)
   TABLESPACE  app_data
   LOB ("NOTES") STORE AS cust_notes
   ( TABLESPACE  app_clob );

 GRANT SELECT ON customer TO app_web;

 ALTER TABLE customer
 ADD CONSTRAINT pk_customer PRIMARY KEY (customer_id)
 USING INDEX TABLESPACE app_indx;

INSERT INTO customer
 VALUES (1,'','',TO_DATE('03-SEP-1968','DD-MON-
YYYY'),'7038898444','John.Doe@test.com','JohnDoe123','11600 Sunrise Valley Dr','Suite
500','Reston','VA','20192','CNN','www.cnn.com','Number one customer',
1,sysdate,NULL,NULL,NULL,'N','N',NULL);

commit;
```

This block of code creates a table called 'Customer' in the APP_DATA schema, grants 'Select' privileges on this table to the APP_WEB schema, adds a primary key and indexing to the table, inserts values into the fields in your new page, and saves the new table in the database.

2. Write a function to retrieve data from your new table: add the following code to your `L_PIPELINE_DML` package in the `APP_DATA` schema.

```
FUNCTION f_customer_data
/*************************************************************
 -- Name:         f_customer_data
 -- Purpose:      Loads rowtype with customer record value
 -- Developer:    Your name
 -- Create Date: yyyy-mm-dd
 -- Modification History:
 --    Date          Developer Name             Description
 --
 *************************************************************/
 (i_id IN customer.customer_id%TYPE)
RETURN customer%ROWTYPE
IS
  customer_data   customer%ROWTYPE;
BEGIN
  SELECT * INTO customer_data FROM customer WHERE customer_id = i_id;

  RETURN customer_data;
EXCEPTION WHEN OTHERS THEN
  RETURN customer_data;
END f_customer_data;
```

3. Create the function specification in your `L_PIPELINE_DML` package specification.

This is a simple example of retrieving data from the database. Oracle can handle much more complex queries such as multi-table joins, views or materialized views. Wherever you have your data, connect it here and it will be easily accessible to your web form in the next step.

Load Data to the Page

Follow these steps to load customer data into your form:

1. Insert the following lines into the Declaration section of your code (the area between 'IS' and 'BEGIN') in `pipeline.p_pipeline_form`.

```
…
IS
customer_data app_data.customer%ROWTYPE := app_data.pipeline_dml.f_customer_data(1);
BEGIN
…
```

You may have noticed that we set a value of 1 in the call to `pipeline_dml.f_customer_data` for this demonstration. In a later section, we'll show you how to pass in a value to the form page for `customer_id` from a list of customer records that we'll show in a dynamic data grid on the same page. When called, this number will pull the data from the DATA layer (`APP_DATA`) and return it to the GUI/APP layer (`APP_WEB`) for loading into the customer form.

2. In the customer form procedure, replace all the hard coded values for the `i_value` parameter in each of the TURBOKit API calls with live data from `customer_data.{fieldname}`. In the web form field for First Name, it should now look as follows:

```
te_inputs.p_text(
i_name      => 'i_first_name',
i_lab_width => NULL,
      -- displays live customer data
i_value     => customer_data.first_nm,
      i_label     => 'First Name:',
i_size      => 10,
i_maxlength => 15,
i_newline   => TRUE,
i_required  => TRUE,
i_readonly  => FALSE,
i_desc      => 'First',
i_desc_hint => 'Customer first name');
```

3. Save both your packages: `pipeline_dml` and `pipeline`.

Save Data to the Database

Now that we've created a form and displayed real data, let's create a procedure to save the form data, whether it's a new record or an update to an existing record.

In the very first section, we looked at the `p_page_init` procedure added to `p_pipeline_form` by the App Wizard:

```
te_gui_utils.p_page_init(
   i_page_title   => 'Pipeline Form',
   i_rec_id       => i_id,
   i_inactive_flg => NULL,
   i_save_close   => NULL,
   i_version_id   => NULL,
   i_tab          => i_tab,
   i_redirect     => NULL,
   -- procedure for saving form data
   i_form_proc    => 'pipeline.p_pipeline_proc',
   i_toolbar      => TRUE);
```

The `i_form_proc` parameter lets the application know where to send the data from the form page. In this case, when the **Save** button is clicked on the web page, the application will send the form data to `pipeline.p_pipeline_proc`.

`pipeline.p_pipeline_proc` is a procedure in the APP/UI layer (`APP_WEB`) where we can perform some business rules before sending the data to the DATA layer (`APP_DATA`) for loading into the database table(s). Performing business rules can include cleansing the data and calling other procedures and/or functions. In some cases, we may not need to perform any business rule processing, and can simply relay the data to the DATA layer (`APP_DATA`) to load.

For this example, we will simply relay the data to the `pipeline_dml` package for loading.

1. Pass the saved form data from the GUI/APP layer to the DATA layer. Modify the template procedure that the Application wizard created, to receive all the fields from our Pipeline web form tutorial.

```
PROCEDURE p_pipeline_proc
/*********************************************************************
-- Name:         p_pipeline_proc
-- Purpose:      Receives form data from p_pipeline_form
-- Developer:
-- Create Date:
-- Modification History:
--   Date        Developer Name      Description

*********************************************************************/
(i_id                    IN NUMBER   DEFAULT NULL,
 i_tab                   IN NUMBER   DEFAULT 1,
 i_save_close            IN VARCHAR2 DEFAULT NULL,
 i_redirect              IN VARCHAR2 DEFAULT NULL,
 i_version_id            IN VARCHAR2 DEFAULT NULL,
 i_inactive_flg          IN VARCHAR2 DEFAULT NULL,
 /* new fields */
 i_first_nm              IN VARCHAR2,
 i_last_nm              IN VARCHAR2,
 i_phone                 IN VARCHAR2,
 i_email                 IN VARCHAR2,
 i_birthday              IN VARCHAR2 DEFAULT NULL,
 i_notes                 IN VARCHAR2 DEFAULT NULL)
```

2. Make the required changes in the package specification.

Add Business Rules

You can add business rules in the GUI or application layer, if you wish. The following block of code shows a sample business rule to be performed when an error is encountered.

```
IS
BEGIN
--add business rules
 pipeline_dml.p_pipeline_load(
  i_id       => i_id,
  i_first_nm => i_first_nm,
  i_last_nm  => i_last_nm,
  i_phone    => i_phone,
  i_email    => i_email,
  i_birthday => i_birthday,
  i_notes    => i_notes);

 EXCEPTION
WHEN OTHERS THEN
te_errors.p_register(SQLCODE, SQLERRM,
                     te_errors.f_get_procedure, NULL);
END p_pipeline_proc;
```

Receive and Process Data

You can use all your Oracle expertise to load data. Follow these steps to receive and process data:

1. Select `p_pipeline_dml.p_pipeline_load`.

```
PROCEDURE p_pipeline_load
 /*********************************************************************
 -- Name:        p_pipeline_load
 -- Purpose:     Loads the data into the customer table
 -- Developer:
 -- Create Date:
 -- Modification History:
 --  Date        Developer Name      Description
 --
 *********************************************************************/
 (i_id             IN NUMBER DEFAULT NULL,
  i_first_nm       IN VARCHAR2,
  i_last_nm        IN VARCHAR2,
  i_phone          IN VARCHAR2,
  i_email          IN VARCHAR2,
  i_birthday       IN VARCHAR2 DEFAULT NULL,
  i_notes          IN VARCHAR2 DEFAULT NULL)
IS
  l_load_dt                          DATE := SYSDATE;
  l_load_user_id NUMBER:= te_security.g_security_rec.user_id;
  l_id           NUMBER := i_id;
BEGIN
  IF l_id IS NULL THEN
    -- Normally we'd use a Sequence here…but this is just a demo
    SELECT MAX(customer_id) + 1
      INTO l_id
      FROM customer;
  END IF;

  MERGE INTO customer a
  USING (
        SELECT l_id customer_id
        FROM dual) t
     ON (a.customer_id = t.customer_id)
   WHEN MATCHED THEN
         UPDATE
           SET first_nm       = i_first_nm,
               last_nm        = i_last_nm,
               phone          = i_phone,
               email          = i_email,
               birthday_dt    = i_birthday,
               notes          = i_notes,
               update_dt      = l_load_dt,
               update_user_id = l_load_user_id
          WHERE customer_id    = l_id
   WHEN NOT MATCHED THEN
         INSERT
               (customer_id, first_nm, last_nm, phone, email,
              birthday_dt, notes, load_dt, load_user_id)
         VALUES
               (l_id, i_first_nm, i_last_nm, i_phone, i_email,
              i_birthday, i_notes, l_load_dt, l_load_user_id);

END p_pipeline_load;
```

You've now successfully created a web form, loaded data into the web form, and saved data from the web form.

Build a Grid

TURBO Enterprise interactive grids have powerful capabilities and are easy to use. Once you supply the data, end users will be able to customize the display so that it automatically saves their preferences—without you having to code this functionality. Follow these steps to build a grid:

1. Create a new `p_customer_grid` procedure in `pipeline_dml`

```
PROCEDURE p_customer_grid
/*****************************************************************
-- Name:                  p_customer_grid
-- Purpose:               Demo Customer Grid
-- Developer:
-- Create Date:
-- Modification History:
--  Date        Developer Name      Description
-- ****************************************************************/ (i_xml  IN
VARCHAR2,
 i_type IN VARCHAR2,
 i_auto IN NUMBER DEFAULT 0)
IS
  -- We'll put a data cursor here
BEGIN
  -- We will define grid parameters and columns here, then
  -- loop through the data cursor and create the grid.
END p_customer_grid;
```

2. Create a procedure specification in your `pipeline_dml` package specification.

3. Cut and paste the following cursor into the declaration section of `p_customer_grid`:

```
CURSOR grid
 IS
 SELECT * FROM (
       SELECT first_nm, last_nm, email, phone, birthday_dt,
              to_char(load_dt,'MM-DD-YYYY HH24:MI:SS') load_dt,
       COUNT(*) OVER () total_rows,
       ROW_NUMBER() OVER (ORDER BY
              DECODE(te_grid.g_order_stmt, '2a', first_nm) ASC,
              DECODE(te_grid.g_order_stmt, '3a', last_nm) ASC,
              DECODE(te_grid.g_order_stmt, '4a', email) ASC,
              DECODE(te_grid.g_order_stmt, '5a', phone) ASC,
              DECODE(te_grid.g_order_stmt, '6a', birthday_dt) ASC,
              DECODE(te_grid.g_order_stmt, '7a', load_dt) ASC,
              DECODE(te_grid.g_order_stmt, '2d', first_nm) DESC,
              DECODE(te_grid.g_order_stmt, '3d', last_nm) DESC,
              DECODE(te_grid.g_order_stmt, '4d', email) DESC,
              DECODE(te_grid.g_order_stmt, '5d', phone) DESC,
              DECODE(te_grid.g_order_stmt, '6d', birthday_dt) DESC,
              DECODE(te_grid.g_order_stmt, '7d', load_dt) DESC
       ) row_num
       from customer
 )
 WHERE row_num BETWEEN te_grid.g_start_row AND te_grid.g_last_row;
```

4. Define the grid columns in the body of `p_customer_grid` (between the `BEGIN` and `END` statements):

```
-- Define the grid columns
 te_grid.p_add_column(i_name =>'Row',i_width => 40, i_show =>TRUE, i_locked => FALSE,
i_sortable =>FALSE,i_printable => TRUE, i_style => null);
 te_grid.p_add_column('First Name', 150, FALSE, FALSE, TRUE,TRUE,NULL);
 te_grid.p_add_column('Last Name', 150, TRUE, FALSE, TRUE,TRUE,NULL);
 te_grid.p_add_column('Email', 150, TRUE, FALSE, TRUE,TRUE,'color:red;');
 te_grid.p_add_column('Phone Number', 150, TRUE, FALSE, TRUE,TRUE,NULL);
 te_grid.p_add_column('Birthday', 150, TRUE, FALSE, TRUE,TRUE,NULL);
 te_grid.p_add_column('Load Date', 150, TRUE, FALSE, FALSE,TRUE,NULL);

 -- Add grid parameters
 te_grid.p_add_parm('i_type', i_type);
 te_grid.p_add_parm('i_auto', i_auto);

 -- Configure the grid size, number of rows, etc.
 te_grid.p_config_grid(
       i_xml           => i_xml,
       i_callback      => 'pipeline.p_customer_grid',
       i_rows_per_page => 75,
       i_rpp_max       => 100,
       i_rpp_increment => 25,
       i_sort_col      => 3,
       i_sort_dir      => 'a',
       i_width         => 800,
       i_height        => 300,
       i_auto_refresh  => i_auto
 );

 -- Create the grid
```

```
 IF te_grid.f_run = FALSE THEN
   RETURN;
 END IF;

 -- Fill the grid with data from the cursor
 FOR x IN grid LOOP
   te_grid.p_row_add(
     i_total => x.total_rows,
     i_style => NULL);
   te_grid.p_cell_add(x.row_num);
   te_grid.p_cell_add(x.first_nm);
   te_grid.p_cell_add(x.last_nm);
   te_grid.p_cell_add(x.email);
   te_grid.p_cell_add(
     te_utils.f_format_phone(replace(x.phone,'.','')));
   te_grid.p_cell_add(x.birthday_dt);
   te_grid.p_cell_add(x.load_dt);
 END LOOP;

 -- Display the grid!
 te_grid.p_xml;
```

Grid Code

This section provides a detailed breakdown of the code that goes into a grid.

Grid Data Cursor

The Grid Data Cursor is a select statement that contains the columns `total_rows` and `total_num`:

```
COUNT(*) OVER () total_rows,
ROW_NUMBER() OVER (ORDER BY
              DECODE(te_grid.g_order_stmt, '2a', first_nm) ASC,
              DECODE(te_grid.g_order_stmt, '3a', last_nm) ASC,
              DECODE(te_grid.g_order_stmt, '4a', email) ASC,
              DECODE(te_grid.g_order_stmt, '5a', phone) ASC,
              DECODE(te_grid.g_order_stmt, '6a', birthday_dt) ASC,
              DECODE(te_grid.g_order_stmt, '7a', load_dt) ASC,
              DECODE(te_grid.g_order_stmt, '2d', first_nm) DESC,
              DECODE(te_grid.g_order_stmt, '3d', last_nm) DESC,
              DECODE(te_grid.g_order_stmt, '4d', email) DESC,
              DECODE(te_grid.g_order_stmt, '5d', phone) DESC,
              DECODE(te_grid.g_order_stmt, '6d', birthday_dt) DESC,
              DECODE(te_grid.g_order_stmt, '7d', load_dt) DESC
) row_num
```

Table 2 lists the variables and descriptions.

Table 2 "Grid Data Cursor Variables"

Input Parameter	Description
`total_rows`	An Oracle analytic statement to count the rows returned by the query
`row_num`	A variable used to order the grid rows based on the current sort column and direction (ASCENDING or DESCENDING) .

Clickable Column Sorting

The TURBO Enterprise grid allows you to click on the column headers to sort the data in ascending or descending order. You can use DECODE statements to support the clickable column sorting feature in your grids. For each column in the query, add two DECODE statements, similar to the following:

```
DECODE(te_grid.g_order_stmt, '2a', first_nm) ASC,
DECODE(te_grid.g_order_stmt, '2d', first_nm) DESC,
```

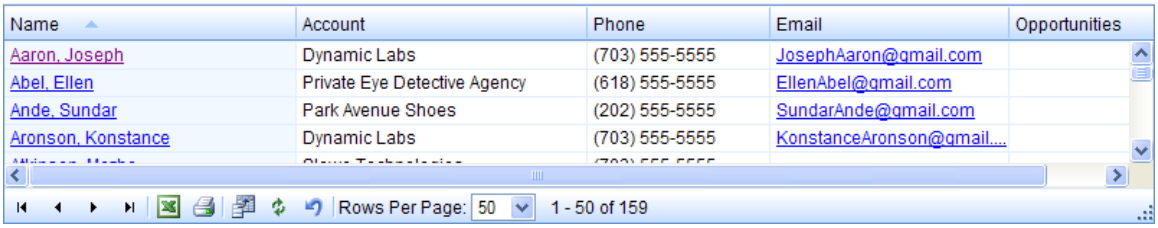
There are three parts to each line. Each line starts with "`DECODE(te_grid.g_order_stmt,`" This starts the `DECODE` statement and tells it to evaluate the result of the click action which is indicated in `te_grid.g_order_stmt`. The next part, '`2a`' refers to the column position and the sort direction. In this example, '`2a`' indicates the second column ascending. Next is the column name. In this case, we are indicating that column 2 corresponds to the field `first_nm`. Finally, the last part, `ASC` and `DESC`, is the actual command that indicates the sort direction.

Grid Paging

You can control grid paging by using the `WHERE` clause of the grid cursor to determine which rows will be currently displayed.

```
WHERE row_num BETWEEN te_grid.g_start_row AND te_grid.g_last_row;
```

In the GRID toolbar, there are several buttons dedicated to navigating through the data set in the GRID. The values of `g_start_row` and `g_end_row` will change as a user navigates through your grid. Figure 3 shows a sample grid.



First Page – Moves to the first page of data.

Next Page – Moves forward one page.

Previous Page – Moves back one page.

Last Page – Moves to the last page.

Figure 3"Sample Grid"

Adding Columns to your Grid

`te_grid.p_add_column` allows you to define columns for your grid.

```
te_grid.p_add_column(i_name,i_width,i_show,
                     i_locked,i_sortable,i_style);
```

Table 3 lists the variables in this procedure and provides descriptions.

Table 3 "Add Column Variables"

Variable	Description
i_name	Name that will be displayed on the column header
i_width	Default width for the column
i_show	Indicates if the column should be displayed or hidden (TRUE/FALSE)
i_locked	Indicates is the column can be resized (TRUE/FALSE)
i_sortable	Indicates is the column can sorted (TRUE/FALSE)
i_style	allows for special HTML formatting (e.g. ‘color:red;’)

Grid Parameters

For most all Grids, you’ll need at least two parameters. TURBO uses the `i_type` parameter to identify the type of grid and the `i_auto` parameter to keep track of the grid auto-refresh rate.

```
te_grid.p_add_parm('i_type', i_type);
 te_grid.p_add_parm('i_auto', i_auto);
```

Grid Configuration

`te_grid.p_config_grid` is used to configure your grid:

```
te_grid.p_config_grid(
       i_xml                => i_xml,
       i_callback           => 'pipeline.p_customer_grid',
       i_rows_per_page      => 75,
       i_rpp_max            => 100,
       i_rpp_increment      => 25,
       i_sort_col           => 3,
       i_sort_dir           => 'a',
       i_width              => 800,
       i_height             => 300,
       i_auto_refresh       => i_auto
 );
```

Table 4 lists the variables in this procedure and provides descriptions.

Table 4 "Grid Configuration Variables"

Variable	Description
i_xml	TURBO-generated parameter
i_callback	Procedure in the `APP_WEB` schema that browser can use to call your grid `p_customer_grid` procedure whenever a user changes sort order, resizes the grid, etc. See the *Grid Callback* section for more details.
i_rows_per_page	Number of rows to display per page
i_rpp_max	The maximum number of rows that the user can choose per page
i_rpp_increment	The increment between row number options. Example: If `i_rpp_max` is 100, and `i_rpp_increment` is 25, the displayed options will be 25, 50, 75, and 100.
i_sort_col	Default sort column
i_sort_dir	Default sort order (‘a’ for ASCENDING, ‘d’ for DESCENDING)
i_width	Default grid width (user can resize grid)
i_height	Default grid height (user can resize grid)
i_auto_refresh	Numeric value indicating number of seconds between data refreshes. 0 indicates that the grid will not auto-refresh.

Create the Grid

`te_grid.f_run` generates the div to hold the grid.

```
IF te_grid.f_run = FALSE THEN
  RETURN;
END IF;
```

Load Cursor Data into the Grid

To add data to the grid, add code to loop through the cursor. You can place the data as it is into the corresponding column, or you can add additional business rules. Example: highlighting any birthdays that occur today. Follow these steps to load cursor data into a grid:

1. Inside the cursor, call `te_grid.p_row_add` to start a new row.

2. Add data from the cursor to each column of the grid using `te_grid.p_cell_add`.

```
FOR x IN grid LOOP
       te_grid.p_row_add(
              i_total       => x.total_rows,
              i_style       => NULL);
       te_grid.p_cell_add(x.row_num);
       te_grid.p_cell_add(x.first_nm);
       te_grid.p_cell_add(x.last_nm);
       te_grid.p_cell_add(x.email);
       te_grid.p_cell_add(
        te_utils.f_format_phone(replace(x.phone,'.',')));
       te_grid.p_cell_add(x.birthday_dt);
       te_grid.p_cell_add(x.load_dt);
 END LOOP;
```

Display the Grid

All the information assembled for the grid is sent to the browser to be displayed. The following command displays the grid:

```
te_grid.p_xml;
```

Grid Callback

After creating `p_customer_grid` in the `pipeline_dml` package, we need to define a grid callback. This callback will be used by the web server to refresh the grid whenever a user changes sort order or pages through the data in the grid. Follow these steps to define a grid callback.

1. Create a procedure called `p_customer_grid` in the pipeline package in the `APP_WEB` schema.

```
PROCEDURE p_customer_grid
/*****************************************************************
-- Name:                    p_customer_grid
-- Purpose:                 Demo Customer Grid
```

```
-- Developer:
-- Create Date:
-- Modification History:
--  Date        Developer Name      Description
-- ********************************************************************/ (i_xml IN
VARCHAR2 DEFAULT NULL,
 i_type IN VARCHAR2,
 i_auto IN VARCHAR2 )


IS
BEGIN
  -- Call grid procedure
  pipeline_dml.p_customer_grid(i_xml,i_type,i_auto);
END p_-customer_grid;
```

2. Create a procedure specification in your pipeline package specification.

3. Add a call to the new `pipeline.p_customer_grid` procedure in your homepage, `pipeline.p_pipeline`.

4. Pass in a value of `NULL` for the `i_xml` parameter.

Security: Grid Callback Page Registration

To ensure security, every page built in a TURBO application will have to be registered. To register `pipeline.p_customer_grid`:

1. Choose the Pipeline application and privilege for the grid. This permits users to access your grid.

2. Navigate to your application homepage and view your grid. Figure 4 shows what your grid will look like.

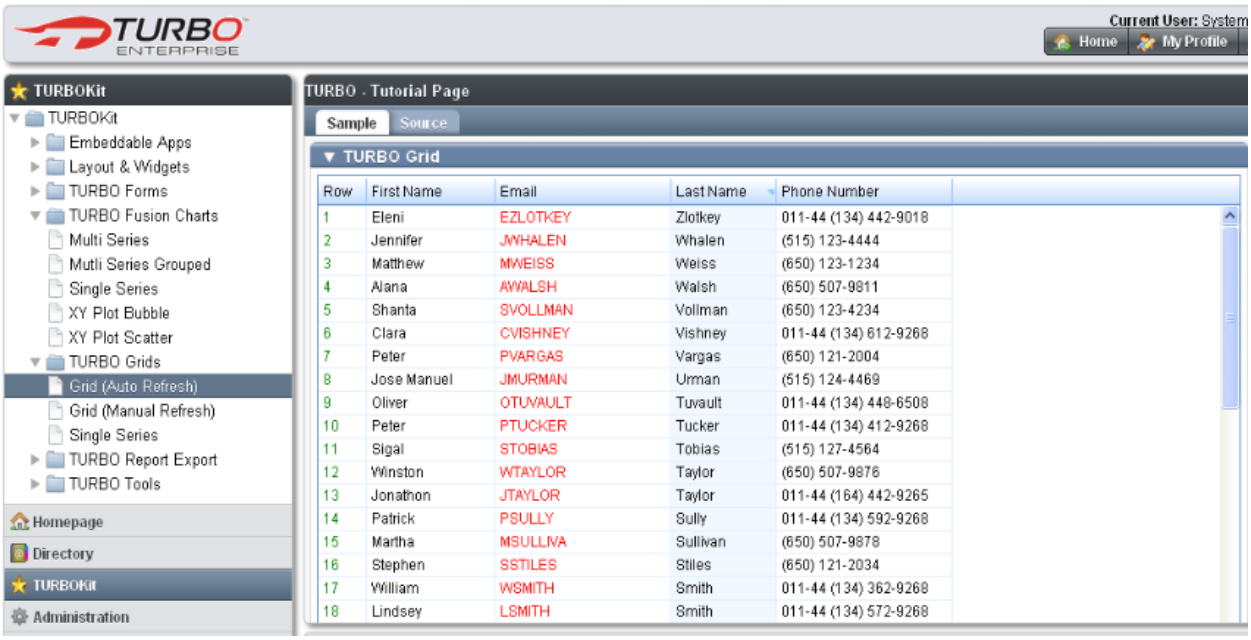


Figure 4 "Sample Grid"

Add Charts and Graphs

TURBO Charts and Graphs are a powerful way to add visual impact to your data. Table 5 lists the different classes of charts you can create with TURBO Enterprise:

Table 5"Charts and Graphs"

Chart Class	Description	Available Chart Styles
Single Series	Single set of X-axis labels and Y-axis values, such as number of employees (Y) per office (X).	Column3D, Pie3D, Area2D, Columns2D, Pie2D, Line, Bar2D, Doughnut2D, Doughnut3D
Multi Series	Each X-axis label may have more than one Y-axis value or "series". For example, base salary (Y1) and commission (Y2) for each employee (X). Y1, Y2, … should be of the same type and order of magnitude for a meaningful display. If they are not, use a Dual Y-Axes chart.	MSArea, MSColumn2D, MSColumn3D, MSLine, MSBar2D, MSBar3D, StackedColumn2D, StackedColumn3D, StackedArea2D, StackedBar2D, StackedBar3D, MSCombi2D, MSColumnLine3D, ScrollColumn2D, ScrollLine2D, ScrollArea2D, ScrollStackedColumn2D, ScrollCombi2D
Multi Series Grouped	Similar to Multi Series, but allows sets of Y-axis series to be grouped and stacked. For example, a chart of sales (Y1), marketing (Y2), development (Y3), and quality assurance (Y4) departments by office (X) could group (Y1 with Y2) and (Y3 with Y4).	MSStackedColumn2D
XY Plot Scatter	Useful for plotting data concentrations. For example, average manager (Y1) and employee (Y2) salaries by year (X). This chart may have multiple Y-axis series. The x-axis values are numeric.	Scatter
XY Plot Bubble	Similar to XY Plot Scatter but with an additional size value assigned to data points. Adding to the previous example, each average salary could also specify the number of employees in the average (Z), which would determine the size of the "bubble".	Bubble
Dual Y-Axes	Similar to Multi Series charts, but with one Y data series assigned to a second Y-axis. This is useful when a data series is not of the same type of magnitude of other Y series. For example, average salary (Y1) by department	MSColumn3DLineDY, MSCombiDY2D, StackedColumn3DLineDY, ScrollCombiDY2D

Chart Class	Description	Available Chart Styles
	(X), with number of employees in the department (Y2) on a second Y-axis.	
Multi Series Grouped Dual Y-Axes	A Multi Series Grouped chart with the addition of second Y-axis for one data series.	MSStackedColumn2DLineDY

Take a look at the TURBOKit to see samples of the different chart classes. The Fusion Charts folder demos each chart class and provides source code. The Interactive Charts folder shows the same charts but also provides some interactive color and font controls to demonstrate some of the available chart options. On the toolbar at the bottom of each chart you can dynamically select the chart style. This will give you a feel for the different styles available. For more information on charts and graphs, refer to the *TURBO Enterprise Resource Guide* available at www.turbo-enterprise.com.

Follow these steps to create a simple single-series chart for your application:

1. Create a new procedure in the `pipeline_dml` package.

```
PROCEDURE p_sample_chart
 /*************************************************************************
 -- Name:           p_sample_chart
 -- Purpose:       Sample Single Series Chart
 -- Developer:
 -- Create Date:
 -- Modification History:
 --  Date        Developer Name      Description
 --
 *************************************************************************/
(i_xml IN VARCHAR2 DEFAULT NULL)
IS
  l_sql VARCHAR2(32767);
BEGIN

END p_sample_chart;
```

2. Create a procedure specification in the package specification.

3. Select TURBO Fusion Charts > Single Series from the TURBOKit.

4. Copy the source code from the Source tab on the Single Series page into `p_sample_chart()`.

```
   l_sql := 'select * from ' ||
            '(select city as label, ' ||
            '(select count(*) from demo_employees de INNER JOIN demo_departments dd ON
de.department_id = dd.department_id ' ||
            '  where dd.location_id = dl.location_id) as employee_cnt ' ||
            'from demo_locations dl) s ' ||
            'where s.employee_cnt > 0 ' ||
```

```
            'order by s.label ';

  te_fusion_charts.p_chart_init(
    i_callbackXML  => i_xml,
    i_callback     => 'te_fusion_chart_tutorial.p_demo_ss_fusion_chart',
    i_datasetSQL   => l_sql,
    i_chartStyle   => 'Column3D',
    i_width        => 800,
    i_height       => 400,
    i_caption      => 'Worldwide Employees',
    i_subcaption   => NULL,
    i_xAxisName    => 'Offices',
    i_yAxisName    => 'Number of Employees by Office Location',
    i_numberPrefix => NULL);

  te_fusion_charts.p_run(i_xml);
```

Note: If your select statement references tables outside of the `TURBO_DATA` schema, you must prefix the table name with the schema name. Example: `APP_DATA.customer`. For this tutorial, we are using the SQL statement that selects data from two demo tables. You can either use the same SQL statement or modify it to access other table(s) of your choosing.

5. Create a new procedure in the pipeline package.

```
PROCEDURE p_sample_chart
/*************************************************************************
 -- Name:           p_sample_chart
 -- Purpose:       Sample Single Series Chart
 -- Developer:
 -- Create Date:
 -- Modification History:
 --  Date        Developer Name      Description
 --
 *************************************************************************/
(i_xml IN VARCHAR2 DEFAULT NULL)
IS
BEGIN
  l_pipeline_dml.p_sample_chart(i_xml);
END p_sample_chart;
```

6. Create a procedure specification in the package specification.

7. Add the following procedure call to `pipeline.p_pipeline`

   ```
   pipeline.p_sample_chart();
   ```

8. Navigate back to the *Pipeline* homepage to see your chart. Figure 5 shows a sample single-series chart.

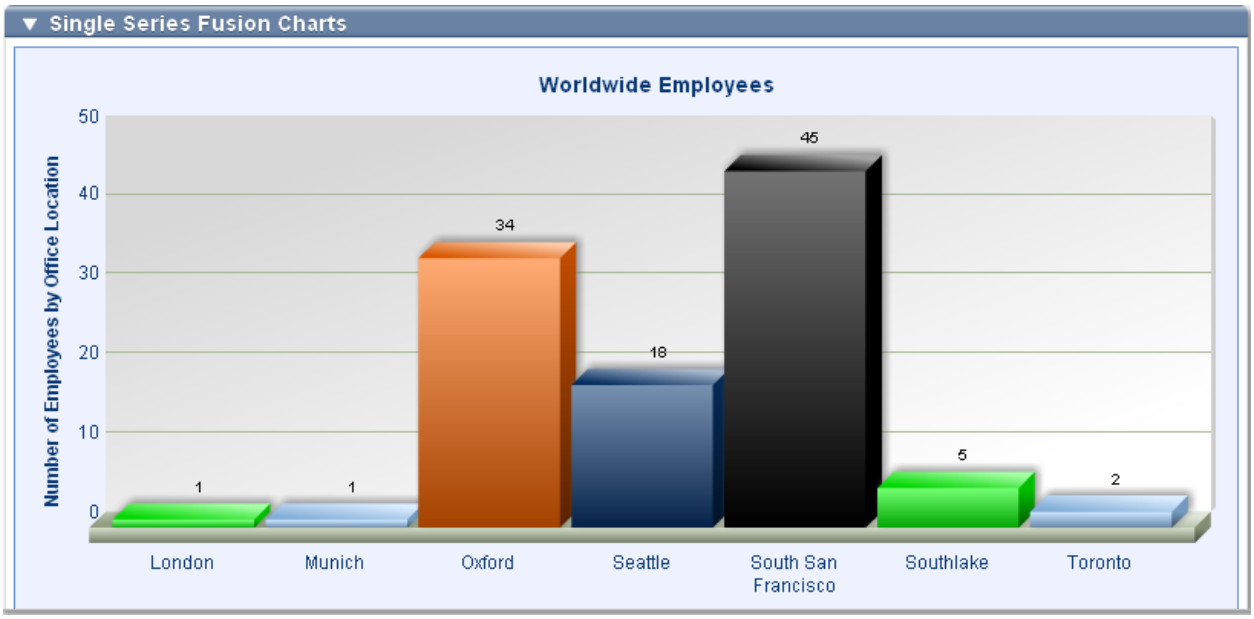


Figure 5 "Sample Single-Series Chart"

9. Register the `pipeline.p_sample_chart()` procedure.

You will notice that if you select a different chart style from the toolbar drop-down list at the lower left of the screen, nothing will happen. This is due to a built-in security measure that requires all calls to the database to be registered. The `pipeline.p_sample_chart()` procedure must be registered because of a JavaScript call which reloads the data when a selection is made.

Register Database Calls

Follow these steps to register your page:

1. Navigate to the Administration application and click on Page Security.
2. Find the `pipeline.p_sample_chart()` and associate it with your application.
3. Select a privilege.
4. Return to the TURBO homepage and confirm that you can now select different chart styles from the toolbar.

Additional Resources

Documentation

You can access the latest online documentation for this product at www.turbo-enterprise.com, by selecting the 'Resource Lounge' tab and clicking on 'Documentation'. To access the documentation link directly from this document, click here: Documentation. Other documentation in this series includes:

TURBO Enterprise Resource Guide: provides detailed information on TURBO Enterprise architecture, security, and application development features.

TURBOEnterprise Installation and Configuration Guide: provides step-by-step procedures to install and configure TURBO Enterprise.

TURBOEnterprise Administration Guide: describes administration tasks and provides step-by-step procedures to perform these tasks.

TURBOEnterprise Oracle Forms Conversion Guide: provides step-by-step procedures to convert unsupported client/server Oracle Forms to a supported Web-based PL/SQL environment.

Videos

To learn more about a specific topic, download watch the TURBO Enterprise demo videos available at www.turbo-enterprise.com. The following videos are currently available:

- TURBO Overview
- Oracle Forms Conversion
- File Management System
- Enterprise Social Networking