LEVERAGING ORACLE PORTAL AS AN ENTERPRISE IDENTITY MANAGEMENT REPOSITORY

Gregory Pike, PIOCON Technologies, Inc.

OVERVIEW

Today's technology landscape provides a myriad of solutions for enabling enterprise-wide, access right provisioning. Provisioning strategies range from simple LDAP repositories to comprehensive Identity Management packages or complex SOA/BPEL implementations with granular business processes. However, these robust provisioning solutions often require highly-specialized skill sets and can be costly to purchase, deploy and maintain. However, an alternative method for managing user identities is available to organizations already invested in Oracle Portal technology.

Although Oracle Portal primarily provides its proprietary Group objects to simplify the management of user privileges *internally* (i.e. page group access), these constructs can additionally be leveraged to facilitate centralized, granular access to data and functionality throughout the organization. Everything from an end-user's security profile to graphical experience in *external* applications can be provisioned directly from Oracle Portal.

USER PROVISIONING AND ORACLE PORTAL

The case for centralized identity management is compelling, especially considering the proliferation of disparate repositories for user metadata in today's workplace. Employing scattered islands of user information is inefficient, difficult to control and may ultimately detract from the initial user experience while these separate profiles are created. Technologies like Oracle's Identity Management ("OIM") product fill this void by "allow[ing] enterprises to manage [the] end-to-end lifecycle of user identities across all enterprise resources both within and beyond the firewall."[1] Implemented properly, OIM and similar solutions enable the enterprise to regain control of the virtual user, ensure metadata integrity, and tighten security protocols.

As an alternative to OIM, the Single-Sign-On ("SSO") component of Oracle Application Server can play a similar role in user provisioning by centralizing and automating the authentication process for external accounts and applications. The benefit to the user is immediate; only a single user name and password is required for enterprise-wise access. In some cases, SSO actually eliminates the need to create multiple accounts for new users by synchronizing with external repositories such as Microsoft Active Directory™.

Combined with OID, the infrastructure designed to secure content in Oracle Portal can additionally fill Identity Management roles by creatively implementing and exposing Group and Item constructs. The hierarchical nature of Groups combined with Group membership can effectively control access to externally-developed menus or database records. Custom Portal Items may be designed to contain a variety of user metadata and this information may be pushed to non-Portal applications using APIs or web services. The concepts presented here are not designed to replace robust identity management solutions, but rather explore the benefits of leveraging Portal technologies to enable provisioning concepts.

LAYING THE FOUNDATION

UNDERSTANDING PORTAL GROUPS AND ITEMS

Groups primarily empower administrators to craft comprehensive but efficient security models to govern user access to Portal objects such as Page Groups, Portlets, Providers and Items. Groups are also hierarchical in nature; Groups can themselves be members of other Groups. Similarly, Portal Items also include a hierarchical structure (top-down only); Sub-Items may be subordinated to parent Items much like a menu or an organizational chart. Portal Items are traditionally used to expose (and secure) content including HTML links to pages or documents, but powerful custom Items may also be defined.

By default, Groups are not customizable and include only three attributes supplied during creation (group name, display name and description). Groups are created from the Administer Tab of the Portal Builder Page in Portal.

In contrast, Portal Items are architected to represent any type of content found on a page, including images, text boxes and other constructs. Portal Items inherit their properties from either default or custom types upon instantiation. Custom Items may be configured to include the necessary attributes to emulate real-world business concepts. Creation of Items is a two step process that first requires the creation of custom attributes and later gathers these attributes into a new Item Type.

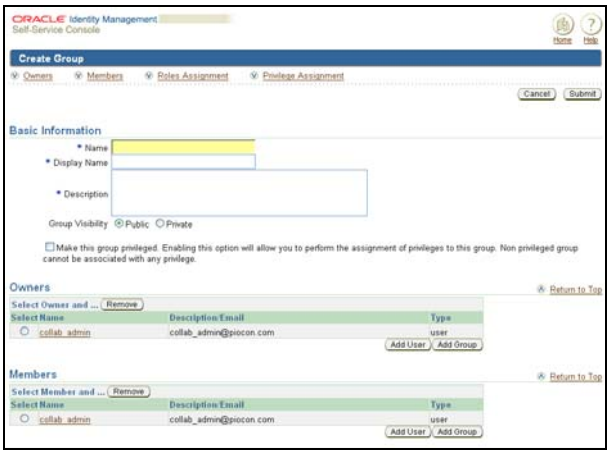

Figure 1: Create a Group in Oracle Portal

Attribute navigation: Navigator→Page Group→Shared Objects→Attributes→Create New Attribute

Item Type navigation: Navigator→Page Group→Shared Objects→Item Types→Create New Item Type

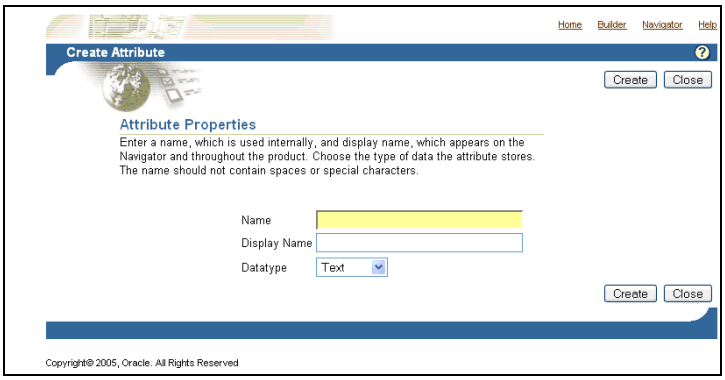

Figure 2: Create custom attributes in Oracle Portal

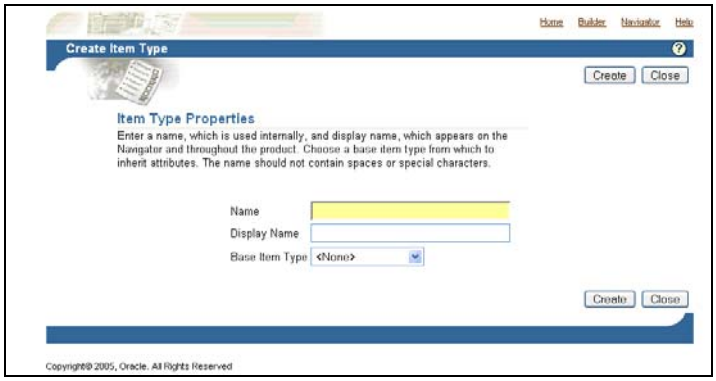

Figure 3: Create custom Item Types in Oracle Portal.

CASE STUDIES

XYZ Manufacturing employs a wide variety of technologies throughout its organization and employees require access to varied reports and applications all with independent security and access privileges. Portal Groups and Items centralize the maintenance of user access in the following examples.

CASE STUDY 1: LIMITING DATA ACCESS IN ORACLE REPORTS

XYZ Manufacturing provides a standard set of Oracle Reports to its field managers, but restricts the viewable information for each facility location. Without user provisioning, this requirement might be achieved by maintaining a User/Privilege schema and including these supporting tables directly into the Oracle Report queries. Of course, this solution also requires development of an administration application to maintain the users and their respective locations. Oracle's Virtual Private Database (VPD) used in conjunction with provisioning through Portal provides an alternate solution that dynamically adds additional clauses to new or existing report queries. With VPD, a function is created to enable row level security and Portal Groups provide the mechanism to feed these functions. The process requires modeling XYZ's location hierarchy as Groups and creating database users in the external system combined with identical Portal users. Finally, a VPD security function and simple Portal API is created to check user privileges at report runtime.

CREATING THE SUPPORTING PORTAL GROUPS

The first step in developing supporting structures for the security function involves modeling XYZ Manufacturing's facility locations as a hierarchical Portal Group structure. By modeling the locations as a set of Groups, a user may be added to a single Group and automatically gain access to all of its children. In the following example, XYZ Manufacturing's four locations (the lowest level of the tree) are modeled into two regions (US and Europe) with an ultimate parent for the entire organization. In Portal, seven Groups are constructed and *parents are added as members to the immediate child Groups*.

In figure 4, User 1 is an explicit member of the Chicago Group and does not gain access to any other Group in the tree. User 2 is explicitly granted access to the Europe Group and additionally gains access to the London and Berlin Groups via hierarchical inheritance. To facilitate the creation of a VPD function in the remote database, Group names incorporate a phrase to identify this as a special security construct (i.e. "LOC") as well as the unique identifier from the external table where the locations are stored (i.e. 1,2…7). In this case, the Group "LOC-4" represents the Chicago location and the Group description field is used to add user-friendly metadata (i.e. Chicago Location Group). An optional approach for adding metadata to Groups is found in the following Case Study.

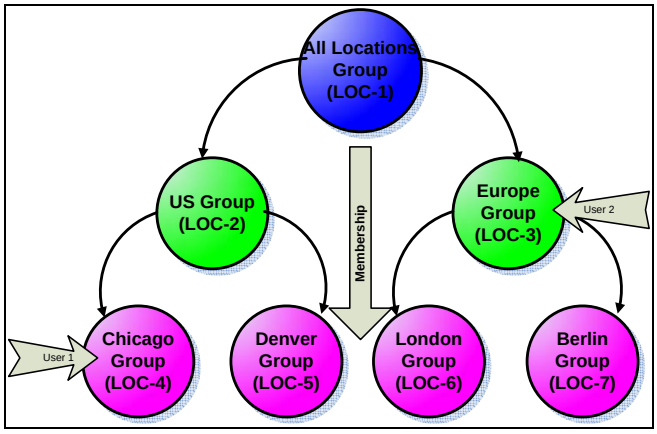

Figure 4: To simulate a location hierarchy, Group membership moves DOWN the tree.

ENABLING ACCESS TO THE LOCATIONS

To access Group membership information in Portal, the remote system must provide metadata identifying the user. Although several methods may be employed to pass user credentials, a transparent but secure option involves creating Portal users with the same name as the remote database user. If the remote schema owner is identified as GPIKE, a portal user named GPIKE is also created. Since the Portal user is created specifically to enable location hierarchy access verification and these users will not access the Portal directly, the password is maintained by the Administrator and may be reused.

From the Portal Administer Tab of the Builder page, new users are added as members to the appropriate location Group.

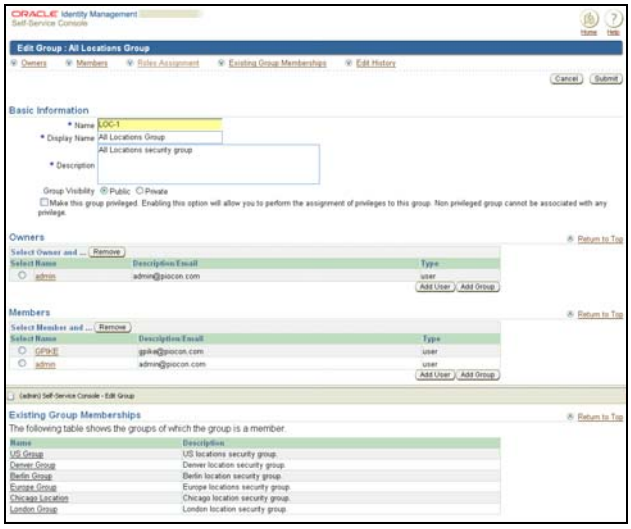

Figure 5: The Edit Group View shows LOC-1 is a member of all child location groups and user GPIKE is a member of LOC-1.

STEPS TO CREATE THE PORTAL API AND VPD FUNCTION

1. The PORTAL.WWV_USER_GROUPS is a simple view that contains a single row for each Portal User/Group combination regardless of whether the user is was granted direct membership to the Group or if membership was inherited. Stored in an ASDB schema with SELECT access to PORTAL-owned views and made available via database link to the external system, the `get_user_locations()` function encapsulates the logic necessary to return a list of allowed location IDs for the supplied user.

```
FUNCTION get_user_locations(f_in_username IN VARCHAR2) RETURN VARCHAR2 IS

    csv_locations VARCHAR2(32767):='';
    CURSOR LOC_cur (c_in_username IN VARCHAR2) IS

        SELECT to_number(regexp_replace(wug.group_name,'^LOC-','')) location_id
        FROM   wwv_user_groups
        WHERE  user_name = c_in_username
        AND    group_name like 'LOC-%';

BEGIN

    FOR loc_rec IN loc_cur(f_in_usernane) LOOP

        IF length(csv_locations) > 1 THEN -- append a comma to the list

            csv_locations := csv_locations ||',';

        END IF;
        csv_locations := csv_locations ||loc_rec.location_id;

    END LOOP;
    RETURN(csv_locations);

END; -- function get_user_locations
```

2. The following Virtual Private Database enabling function is compiled in the schema or database where the reports are executed. The function takes no parameters and simply passes the invoker's schema owner name (using the USER keyword to the ASDB get_user_locations() function (above). The return value from the function becomes a dynamic VPD clause automatically added to queries executed by the reports.

```
FUNCTION request_user_locations RETURN VARCHAR2 IS

    allowed_location_ids VARCHAR2(32767):='';

BEGIN

    allowed_location_ids := get_user_locations@ASDB(USER);  -- Requires synonym
    RETURN 'location_id IN ('||allowed_location_ids||')';   --  and a DB link!

END;  -- function request_user_locations
```

3. To associate the request_user_locations() policy function with a hypothetical LOCATIONS_D table, first create a database link from the external database to ASDB and a synonym to the get_user_locations() function. Finally use the DBMS_RLS package to enable row-level security:

```
DBMS_RLS.add_policy (object_schema    => 'XYZ_DW',
                     object_name      => 'LOCATIONS_D',
                     policy_name      => 'LOC_SECURITY',
                     policy_function  => 'REQUEST_USER_LOCATIONS);
```

System Architecture

Figure 6 depicts the complete solution for securing an external reporting system using Oracle Portal users, hierarchical Groups and VPD. In (1), the user executes a report as GPIKE in the external reporting system database and the VPN policy (2) executes a call to the portal (3) to validate the allowed locations. The portal user (4) is a member of one or more groups (5) and this information automatically becomes an additional WHERE clauses in the originating query.

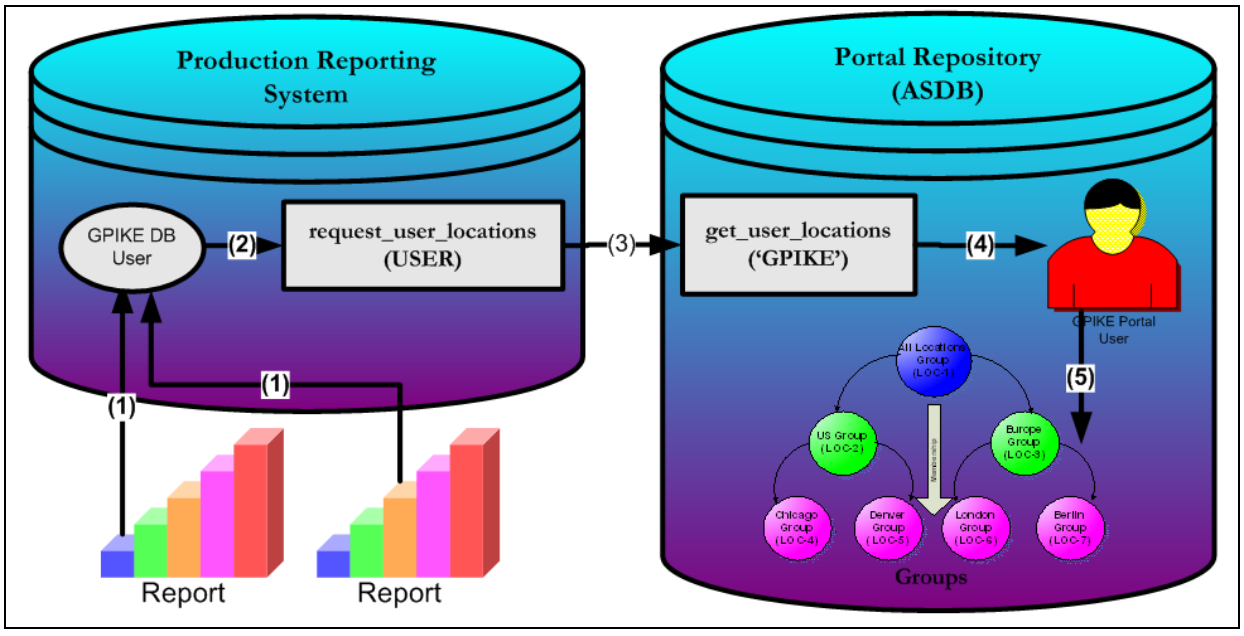

Figure 6: Report security process model.

CASE STUDY 2: CREATING DYNAMIC WEB MENUS

XYZ Manufacturing decides that future web development will leverage the Oracle Portal framework; however, any legacy applications will continue to operate using their existing environments. For administrators, this scenario presents challenges for synchronizing security among the various legacy systems and complexity since multiple administration systems are likely to exist. In the following scenario, XYZ creates a new, web-based reporting interface with hierarchically-structured menu-items designed to access specific reports. The dynamic menus are constructed based upon privilege settings from the user's Portal group membership.

CREATING "BOTTOM-UP" GROUP HIERARCHIES

Unlike the top-down location hierarchy created in Case Study 1, a web menu hierarchy requires bottom-up membership. With a web menu, users gain access to the individual lowest-level menu items as well as all direct parents to preserve the visual structure of the menu. For this example, the XYZ Manufacturing's web-enabled reporting system includes the following menu structure:

Level 1: Menu Node	Level 2: Menu Nodes	Level 3: Reports
Reports	Sales	Sales Forecast
		Prior Year Sales
	Orders	Historical Orders
		Order Detail

Table 1: XYZ Manufacturing's reporting web menu structure

A user with access to only the *Sales Forecast* report must also gain access to the *Sales* node at Level 2 in the menu and the *Reports* node at the top of the menu. This is accomplished by creating a series of Portal Groups *with children added as members to the direct parents.*

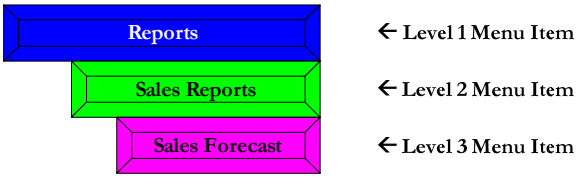

Figure 7: Available menu items for a user privileged to view only one report

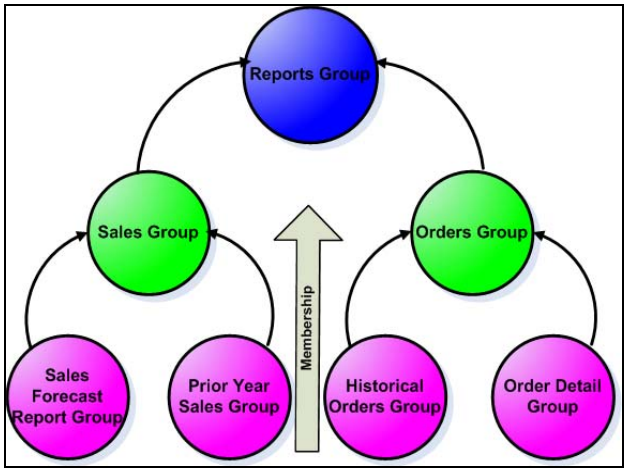

Figure 8: To simulate a menu, Group membership moves UP the tree.

Combining Portal Groups and Items

Portal Groups provide a powerful framework for developing hierarchical access to content but they lack the breadth of additional metadata that allow them to serve as a repository for dynamic web menu items. However, when Portal Groups and Items are joined into a single construct, the combined object provides a powerful tool for describing and applying a highly-customizable user security model to menus. Since Portal Items can include any number of custom attributes, a Menu Item Type is designed to store metadata such as the displayed text, the menu item type (node or report), the order the item appears in a menu and an optional URL for the lowest level items to invoke a report when clicked.

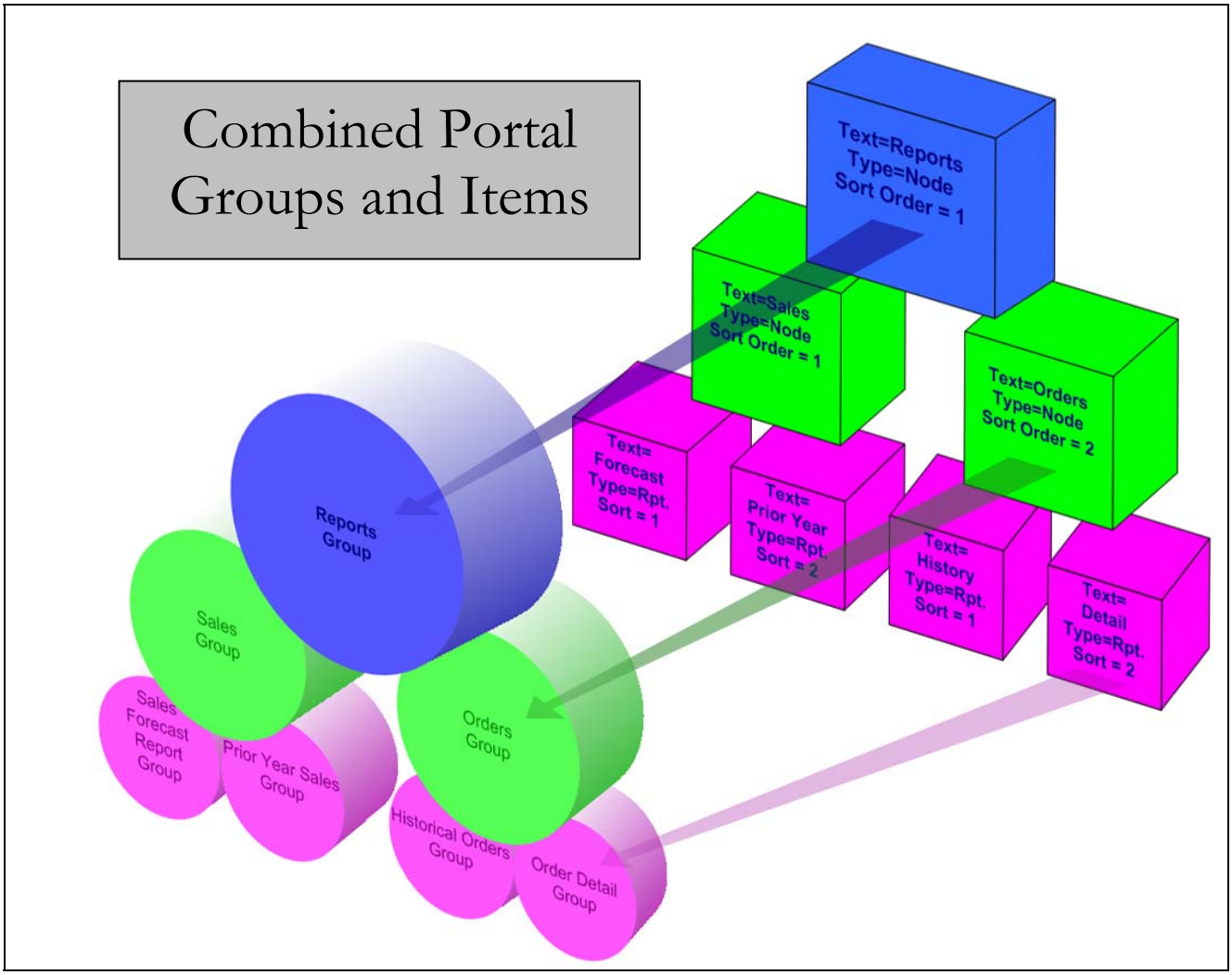

Figure 9: Portal Groups and Associated Items are combined to create a powerful provisioning tool.

CREATING CUSTOM ITEM TYPES AND THE HIERARCHICAL MENU

1. Create attributes from Navigator → Page Groups → Shared Objects → Attributes → Create New Attribute screen:

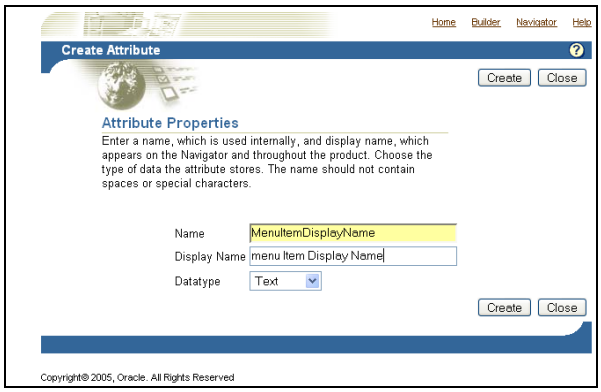

Figure 10: Create a custom attribute for Menu Item types.

2. Create a custom item type from Navigator → Page Groups → Shared Objects → Item Types →Create New Item Type screen Click to edit the item properties. The custom attributes are added under the Edit Item Type → Attributes screen:

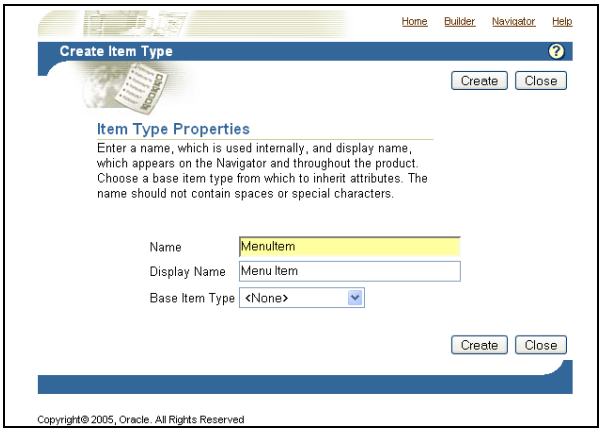

Figure 11: Create the custom Menu Item Type

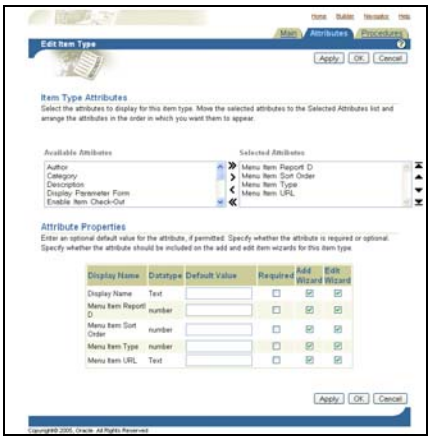

Figure 12: Add the custom Attributes to the Menu Item type.

3. Create a page to contain the hierarchical menu as a set of Items and Sub-Items. The page itself is not accessed or viewable by users and does not require formatting, however it serves as a helpful visual aide for administrators to review the complete menu structure. In order to successfully traverse the menu with queries, subordinate items are created as *Sub-Items*; only menu items with no parents are created as *Items*.

Figure 13: Create the menu structure with by adding 7 Menu Items. Remember to preserve the hierarch by using sub-items.

4. Enable Item Level Security for the entire page as well as for each individual Item that is added to the page. Granting access to the associated Group in the menu hierarchy links the two constructs (i.e. Sales Item ← → Sales Group) and allows the Item to serve as the repository for the menu item metadata.

Figure 14: Enable Item Level Security

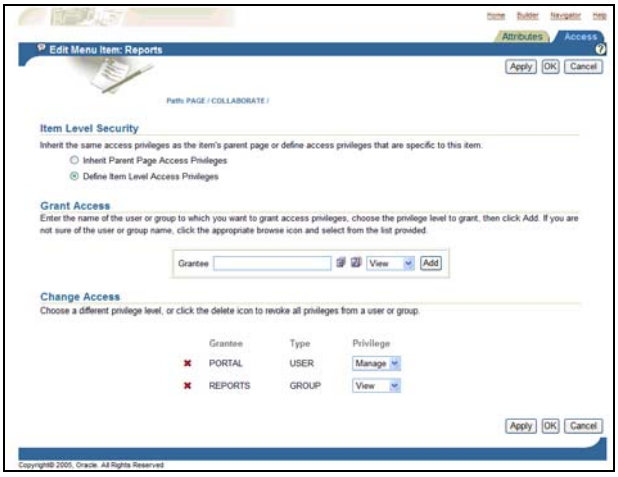

Figure 15: Link Groups to Items with Item Level Security

CREATING CONSOLIDATION GROUPS TO EASE ADMINISTRATION

Because the Group hierarchical structure for a menu is built in a bottom-up fashion, administration can be tedious without the help of a Consolidation Group. In the absence of a Consolidation Group, granting access to all reports in the menu requires the administrator to add the user to each individual lowest level Group. Although this example depicts only four report options, a real-word scenario might have dozens or even hundreds of reports.

In Figure 15, the menu structure is inverted from Figure 8 to depict the lowest levels (individual reports) at the top and additionally includes a Consolidation Group (the All Menu Items Group). A Consolidation Group can include any logical subset of the reports that might be granted to a large number of users. It is conceivable that a Sales or Orders Consolidation Group (no to be confused with the Sales and Orders Groups that govern the menu items only) might be added to simplify the administration of different user roles.

In this example, User 1 sees the Order Detail Group, Order Group (the immediate "child" in this inverted view) and the Reports Group (ultimate "child" in this inverted view) while User 2 inherits access to all Menu Items from the Consolidation Group.

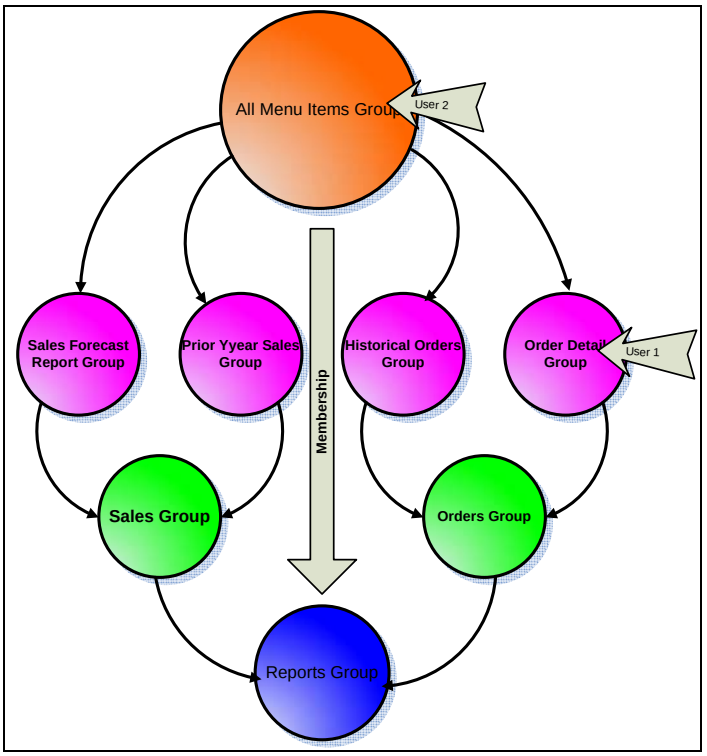

Figure 16: Granting access to an "All Menu Items Group" eases administration.

Putting it All Together – Dynamically Determining Group Access (Menu Items)

As with the locations hierarchy, the final step for using Portal menu security is to expose the list of allowable Menu Items for a particular user. Since information regarding the structure and order of the menu as well as the URLs for the reports is also included via the attached Items, it is preferable to create a view into this information and return the data in the order the menu might be displayed. The view returns the following rows for a user with access to the entire menu (member of the All Menu Items Group) and the data is used to dynamically render a menu using JSPs or PL/SQL Server Pages and JavaScript.

Item ID	Title	Parent ID	Item Type	URL	Sort Order (rel)
1	Reports	<NULL>	Node	<NULL>	10
2	Sales	1	Node	<NULL>	10
4	Sales Forecast	2	Report	/reports/sales.html	10
5	Prior Year Sales	2	Report	/reports/prior.html	20
3	Orders	1	Node	<NULL>	20
6	Historical Orders	3	Report	/reports/hist.html	10
7	Order Detail	3	Report	/reports/detail.html	20

Table 2: Rows returned from the dynamic menu view for a user with full access

The query on the following page was constructed to return a SIBLINGS-sorted list of available menu items for the supplied user. Optionally, the query could include a passed Portal Page name to enable implementation of multiple menus. The query was designed with Subquery Factoring (better known as the WITH clause) to maximize efficiency and could include any number of custom attributes to enable the external application to properly render and control a web menu.

Table or View Name	Description
wwv_things	Master list of all Portal objects.
wwpob_page$	Allows the query to be limited to a single menu on a single page using the parent_name column.
wwsec_group$	Master list of Portal groups.
wwsec_sys_priv$	Ties items to groups with Item Level Security.
wwv_user_groups	Direct and inherited Group membership by user.
wwsbr_item_types	Specify custom menu Item Type.
wwsbr_item_type_attributes	Attributes for the custom menu Item Type.
wwsbr_attributes	Attribute names for the custom menu Item Type
wwv_thingattributes	Values contained in attributes for each instantiation of a custom menu Item.

Table 3: Table descriptions for a dynamic menu item query

Dynamic Menu Query

This query returns an ordered list of allowable menu items provided *portal.wwctx_api.set_context* is first executed for the specified user to ensure that tables are synchronized with Oracle Internet Directory (OID). Please note the **bolded** clauses which are provided at run time but hard-coded here for discussion purposes.

```
WITH temp AS (      -- This query gets 1 row for each item/attribute combination
    SELECT t.masterthingid id,
           t.title,
           t.parentid,
           wa.name item_name,
           DECODE(ta.valuetype,'text',ta.value,ta.numbervalue) item_value,
           pg1.name parent_page
    FROM   portal.wwv_things t,
           portal.wwpob_page$ pg1,
           portal.wwsec_group$ gp,
           portal.wwsec_sys_priv$ p,
           portal.wwv_user_groups wg,
           portal.wwsbr_item_types wit,
           portal.wwsbr_attributes wa,
           portal.wwsbr_item_type_attributes wita,
           portal.wwv_thingattributes ta
    WHERE  t.siteid           = SUBSTR(p.name,1,INSTR(p.name,'/')-1)
    and    t.masterthingid    = substr(p.name,instr(p.name,'/')+1,8)
    and    p.object_type_name = 'ITEM' -- ITEM / GROUP
    and    p.grantee_group_id = gp.id
    and    p.grantee_type     = 'GROUP'
    and    t.siteid           = pg1.siteid
    and    pg1.id             = 1
    and    wg.group_name      = gp.name
    and    wg.user_name       = UPPER('GPIKE')  -- Provided at run time.
    and    wit.name           = 'MenuItem'    -- The name of the custom item type.
    and    wit.id             = t.subtype
    and    ta.masterthingid   = t.id
    and    wita.item_type_id  = wit.id
    and    wita.attribute_id  = ta.attributeid
    and    wa.id              = wita.attribute_id
    )
SELECT id,title, parentid, menu_item_type, report_id, url, sort_order -- Order the result
FROM  (    -- A classic pivoting mechanism to place all attributes in a single row
       SELECT temp1.id,                        -- Needed for the CONNECT BY
              temp1.parentid,                  -- Needed for the CONNECT BY
              temp1.title,                     -- The menu item text
              temp1.item_value menu_item_type, -- Is this a node or a leaf
              temp2.item_value report_id,      -- What report to run
              temp3.item_value url,            -- What portal page to show
              temp4.item_value sort_order      -- Order of the item in the menu
       FROM   temp temp1, temp temp2, temp temp3, temp temp4
       WHERE  temp1.id        = temp2.id
       AND    temp2.id        = temp3.id
       AND    temp3.id        = temp4.id
       AND    temp1.item_name = 'MenuItemType'      -- Custom item type attribute.
       AND    temp2.item_name = 'MenuItemReportID'  -- Custom item type attribute.
       AND    temp3.item_name = 'MenuItemURL'       -- Custom item type attribute.
       AND    temp4.item_name = 'MenuItemSortOrder' -- Custom item type attribute.
      )
CONNECT BY parentid = PRIOR id
START WITH parentid = 0
ORDER SIBLINGS BY sort_order;  --Orders the menu items in the order displayed
```

System Architecture

A dynamic web menu is realized using two options with both ultimately leveraging a combined Group-Item construct in Portal. In Option 1, the requesting web application (from JSPs or PL/SQL Server Pages) connects directly to a local database where the menu item query is exposed as a database view over a DB link. In this situation, the query is modified to include USER and perhaps Portal Page to enable appropriate filtering. In Option 2, the web application requests the menu items from a web service and passes the username and the desired menu. The web service employs any one of many options for retrieving the menu items including execution of a stored database function that inquires on the view.

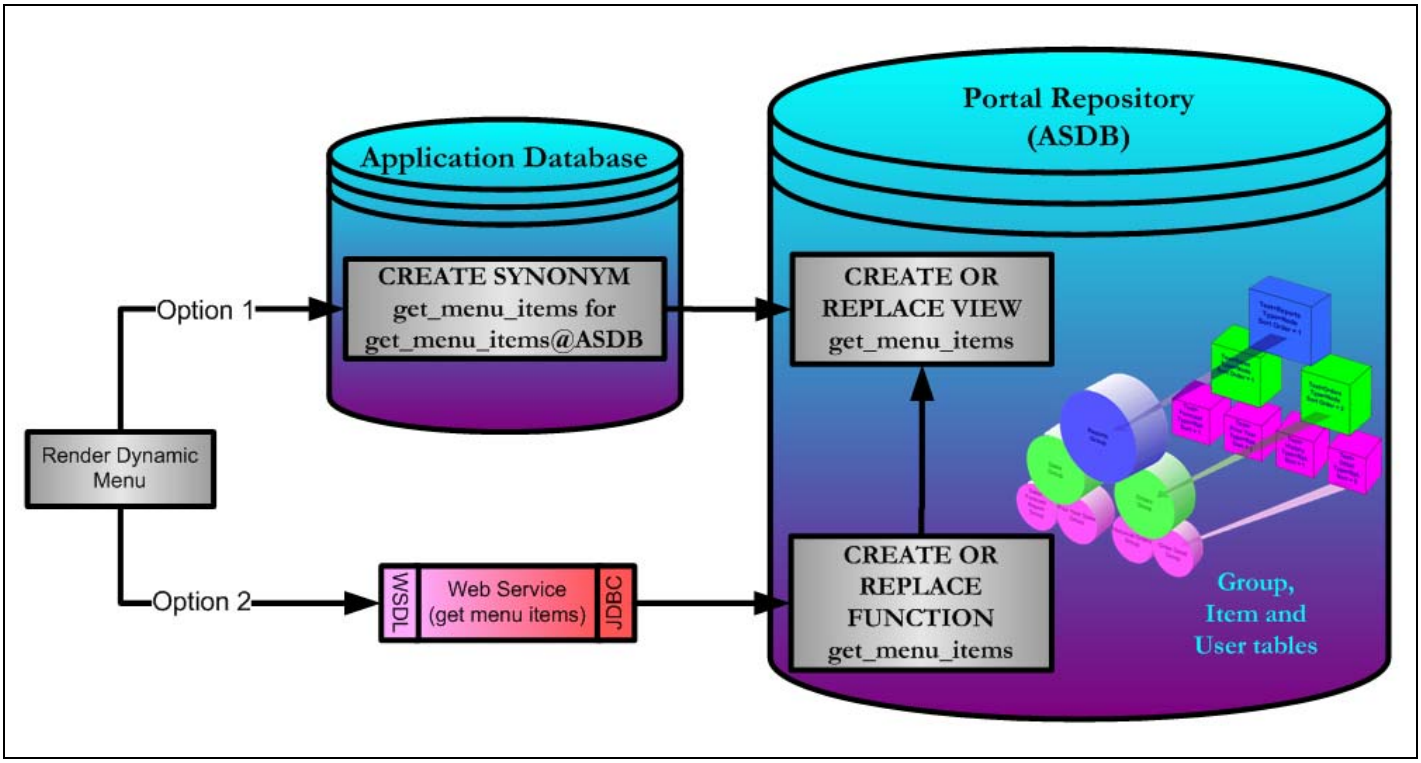

Figure 17: Report security process model.

CASE STUDY 3: DEVELOP AN ENTERPRISE PROVISIONING API OR WEB SERVICE

For the organization that selects Oracle Portal as a user provisioning solution, using the default Portal Administration screens may hinder full integration with external systems. Often, legacy applications already include administration systems and while Portal may be used as the central data repository, the external systems are still leveraged for user creation and maintenance. Fortunately, all of the functionality described here is accessible via Portal APIs. However, exposing these APIs directly to external systems is unwise from a security standpoint and requires developers to understand the sometimes complex relationships between Portal Users, Groups and Items. As an alternative, the Portal APIs may be repackaged as an Enterprise Provisioning API (EP-API) that provides the mechanisms to create users and manage preferences. This repackaged API is exposed either directly through database links for Oracle-based external applications or using a web service.

CREATING AN ENTERPRISE PROVISIONING API

Because the included APIs from Oracle are often complex and require a detailed understanding of Portal, the EP-API is designed to simplify integration for the external developer. The robust EP-API should include the ability to perform the day-to-day user management functions available in Portal's administration screens. The following list describes the base functionality provided by a robust EP-API:

- Create and delete/inactivate Portal users.
- Grant or revoke user access to Portal Groups.
- Inquire on the allowed Groups for a user.
- Create new Groups and modify their hierarchical structures.
- Create new Items and associate these to Groups.
- Modify the custom attributes for Portal Items.
- Inquire on existing Portal hierarchies.
- Inquire on enterprise access privileges to facilitate reporting and audit.

ORACLE PORTAL APIS

A complete understanding, or even an overview, of the supplied Oracle Portal API's is beyond the scope of this document, however the majority of the functionality described here is accessed using the WWSEC_API and WWSBR_API packages. See http://www.oracle.com/technology/products/ias/portal/html/plsqldoc/pldoc1012/index.html for a complete description of all Portal APIs. The following list describes the candidate components be exposed through an EP-API.

API Package	Function or Procedure Name	Description
WWSEC_API	add_group_to_list	Adds a group as a member to another group.
	add_portal_user	Adds a Portal profile entry to the OracleAS Portal 10G repository.
	create_list	Creates a group in OID and returns the profile ID for the portal.
	delete_group_from_list	Deletes a group from the list of members of another group.
	delete_list	Deletes a group from OID and associated references to the group.
	delete_portal_user	Deletes an OracleAS Portal 10G user profile entry.
	delete_user_from_list	Deletes a user from the membership list of a group.
	grantee_list	Returns a list of grantees for a named object/specified owner.
	is_user_in_group	Checks whether a user belongs to a specific group.
	person_info	Returns user information, given a person ID.
WWSBR_API	add_item	Creates a new item on a specified page in a page group.
	add_item_ils_privileges	Adds item level security access privileges for one or more users or groups to a single specified item.
	enable_ils_for_item	Enables item level security for a specified item.
	modify_item	Modifies an existing item.

Table 4: Selected Oracle Portal API descriptions

The key to providing a usable EP-API is obfuscating the complexities of Portal's APIs and combining multiple API calls into a single function or procedure. For example, a "Create User" EP-API function should only include those elements of a user's record that are applicable to the external system and might additionally include the ability to add a new user to hierarchical groups during the creation process.

Exposing the Enterprise Provisioning API as a Web Service

Although the complexities of exposing PL/SQL packages as Web Services is beyond the scope of this document, Oracle's own Portal Center (http://portalcenter.oracle.com) provides valuable resources. Of particular note is Jason Price's article found at http://www.oracle.com/technology/pub/articles/price_10gws.html that describes the process of exposing PL/SQL procedures as web services.

Portal Provisioning Advantages and Disadvantages

Oracle Portal is not a true provisioning tool and there are several considerations to evaluate prior to leveraging Group and Item constructs to centrally manage user security and access privileges.

Advantages

- Portal's out of the box functionality includes a complete suite of support structures and management screens to start provisioning immediately.
- Centralizing user metadata helps enable a help desk model for application support.
- For current owners of Oracle Portal, user provisioning leverages current technologies and in-house skills.
- Compatible with BPEL when appropriate APIs are constructed.

Disadvantages

- The effort of creating duplicate users in Oracle Portal may be greater than the perceived time savings. However, an EP-API can be employed to create users programmatically from the external systems.
- Default portal administration pages are not intuitive; however, a custom page or an EP-API can be substituted.
- Exposing Portal privileges to external applications requires careful security scrutiny.
- Centralizing user privileges upgrades the Portal to a mission-critical system.

Summary

Although there are clear benefits derived from enterprise user provisioning, barriers to entry may cause some organizations to resist the changes. Oracle Portal enables the first steps by limiting investment in new technologies and leveraging existing skill sets to deliver a robust model for managing the user experience and life-cycle.

About the Author

Greg Pike is a PIOCON Technologies Managing Principal with over 15 years experience in the delivery of Oracle-based custom applications and solutions. Mr. Pike currently focuses on Business Intelligence web applications and he is the founder and moderator of www.singlequery.com (Oracle technology blog).

Acknowledgements

- Chip Dawes, D&D Technologies – PL/SQL security functions.
- Jeremy Simmons, PIOCON Technologies, Inc. – Graphic arts and architectural review.
- John Weicher, PIOCON Technologies, Inc. – Graphic arts and architectural review and editing.

References

(1) Oracle identity and Access Management Suite – Key Features and Benefits, Oracle Corporation, 2006.