Dynamic PSP™ Version 2 – The Most Advanced Web-based Tool to Develop Web Applications in Oracle® PL/SQL™.

Introducing Dynamic PSP.

Dynamic PSP is the server-side scripting solution for Oracle8*i*™ and Oracle9*i*™. DPSP2 makes true remote rapid application development a reality with its web-based development interface, modular design and advanced features not available in any other server-side scripting solutions for Oracle. It is installed into the database and provides a web-based development framework for building even the most sophisticated web applications right from your favorite web browser. DPSP2 uses PL/SQL as scripting language leveraging existing Oracle programmers' skills and bringing them to web development. Being itself implemented in PL/SQL, DPSP2 runs everywhere Oracle does. DPSP2 uses established web applications framework provided standard with Oracle8*i* and later, and extends it with 100% HTML-based interface, advanced features for code debugging, profiling, sharing and reuse, strict access control and more.

This white paper describes internal architecture of Dynamic PSP Version 2 and highlights some of the exciting features found in this latest incarnation of the toolkit.

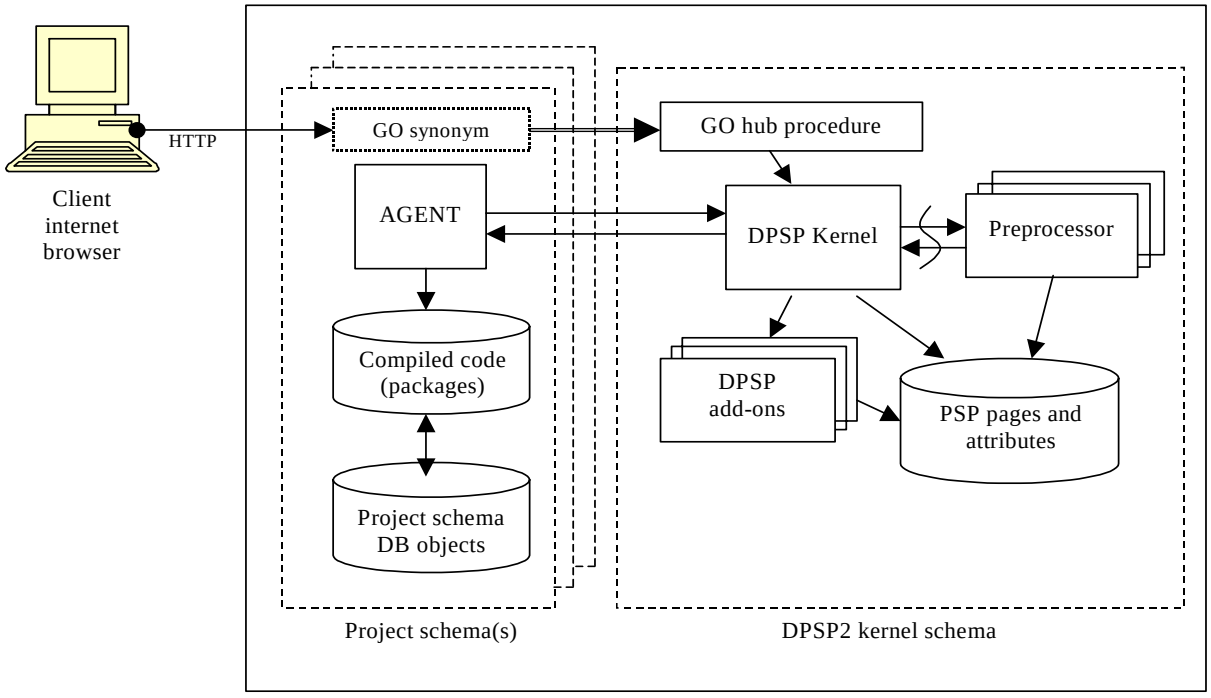
Internal Architecture

DPSP2 core is broken into modules, all of which provide different functionality and depend on DPSP Kernel. DPSP2 introduces internal persistent hierarchical data storage, similar to Windows Registry, which is called DPSP Registry and is independent from the rest of the DPSP2 core. All DPSP2 modules store their properties and other data in the Registry. DPSP Registry also exposes an API that allows developers to store and manipulate their own data using this hierarchical storage. DPSP2 is built as a set of modules interacting with each other at runtime to achieve needed functionality. For example, when a unit is executed, the call hub procedure calls the Kernel, which in turn calls the global Preprocessor module. Preprocessor module generates executable code from the page source depending on the unit type, and passes it to Kernel for execution. The Kernel sets up the page environment, including parameters passed to the page, and calls project schema-resident Agent module, which executes the page code in project schema context. Kernel then retrieves the result of code execution and passes it over to the HTTP gateway, which returns the page to the requesting client. This modular approach allows extending DPSP2 functionality by adding new modules to it (preprocessors for different languages, postprocessors for additional output tweaking/processing, etc).

DPSP2 pages are stored as blocks of PSP code or plain HTML/CSS/XML/etc. and are called units. This is because each unit may represent a complete page, or a reusable part of page. Each DPSP2 unit has a profile associated with it. Profile contains unit attributes, which define the way the unit should be processed, which security settings to apply to it at compilation and run time, etc.

DYNAMIC PSP VERSION 2 CALL DIAGRAM

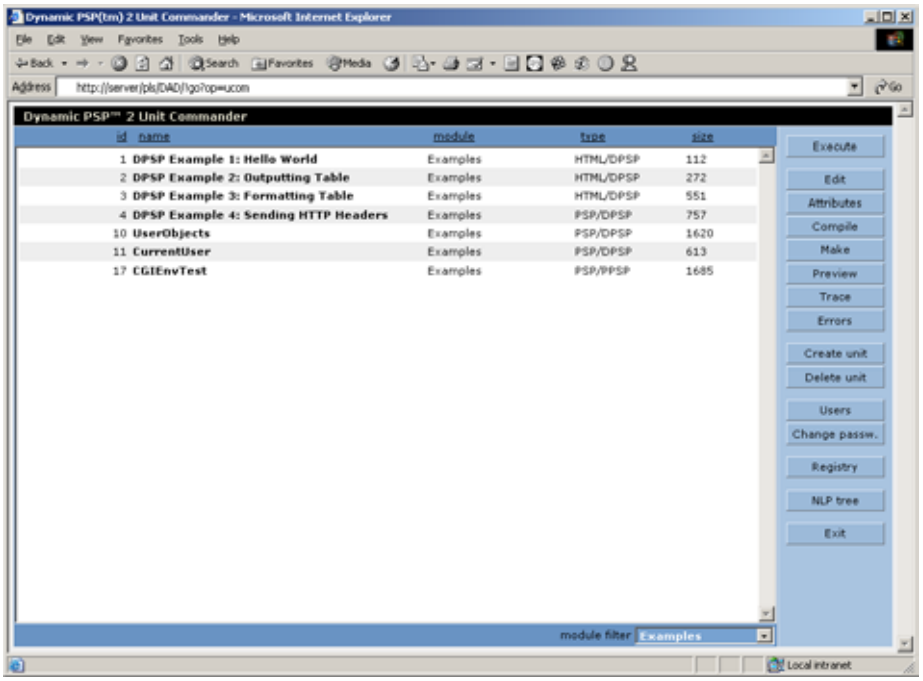
The call flow is presented on the following diagram:



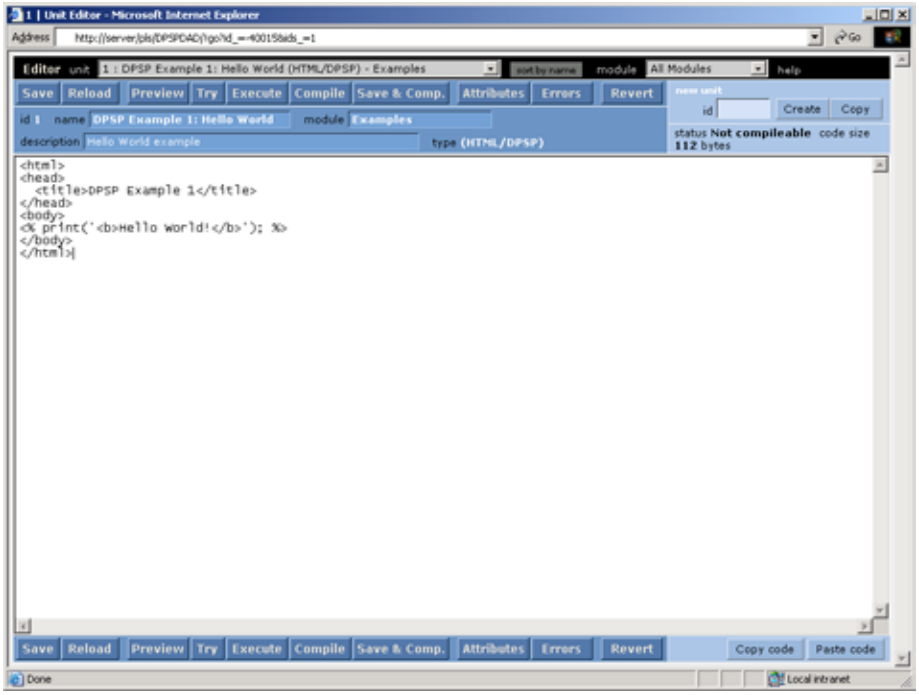
When a user makes a request, the GO procedure is invoked through project-resident synonym. The GO hub procedure processes the call parameters and passes them to the Kernel. Kernel retrieves the unit code and passes it to the PSP Preprocessor assigned for the called unit. PSP Preprocessor determines if the executable code should be regenerated or there is a compiled version of the unit available and generates the executable code for the unit if needed. Kernel then calls project schema Agent and requests it to execute the unit in context of the project schema. Dynamic or compiled unit code is executed against the project schema (accessing all schema objects as needed). The result of execution is stored in internal Kernel result buffer and is returned to the requesting user via HTTP gateway (it may be post-processed by a postprocessor module before it is returned if needed – for example, if the result document is XML and an XSLT template should be applied to it). If the unit requests other units to be executed, these calls are carried out by the Kernel and results of calls are either embedded into result buffer or passed to the calling unit for further processing.

DEVELOPING WITH DYNAMIC PSP.

To develop with Dynamic PSP, the only tool you need is a HTML 4.01 browser with JavaScript enabled (Microsoft Internet Explorer 5.5 or later, or Netscape 6.2 or later is recommended.) As soon as you install and configure the Dynamic PSP Kernel on target system, you can use the browser to access the Development Interface:

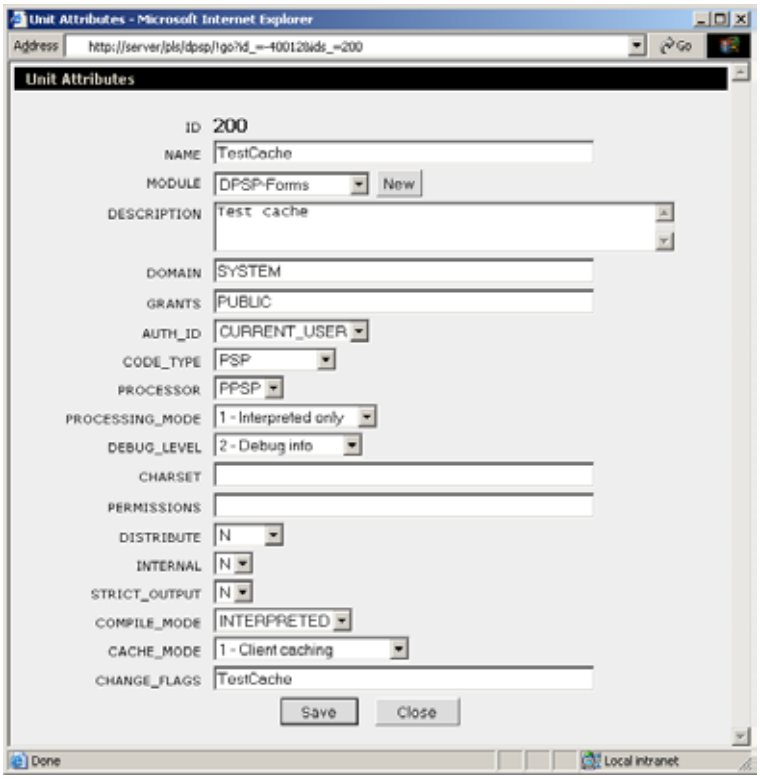


From the Unit Commander, you can access Unit Editor, Unit Attributes editor, create, drop or compile units, manage DPSP2 Kernel users and Registry, and perform various other tasks.

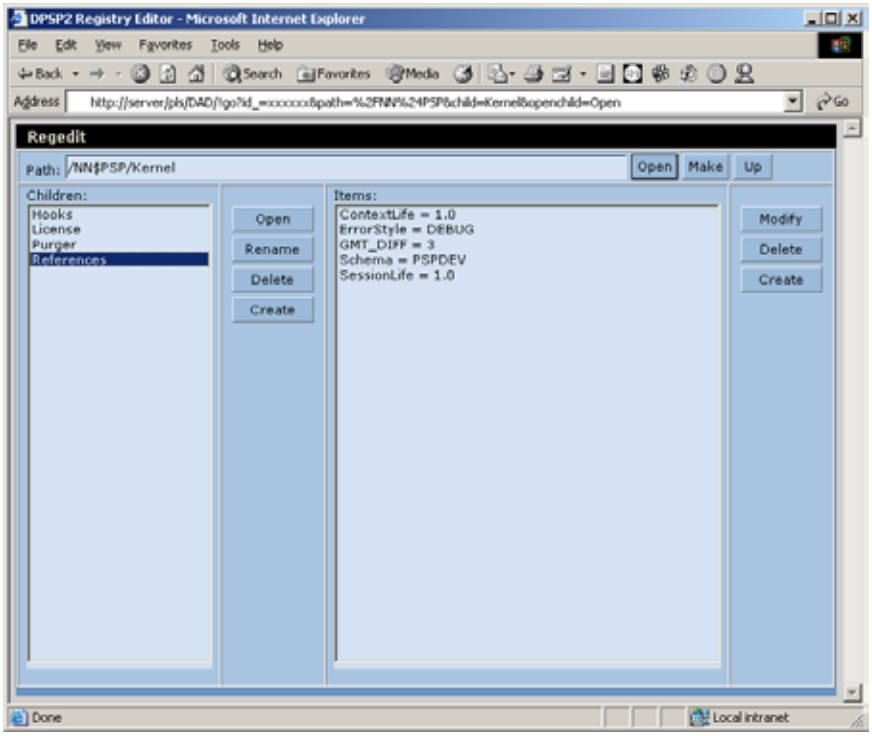


In the Unit Editor, you can copy and paste your code to and from a WYSIWYG tool like Macromedia Dreamweaver, or use the provided text area to manually edit the PSP code. Unit Editor allows you to save, compile and execute your code, edit

unit attributes, see unit error log and restore last backup of the unit code if you do not like the result of your modifications.



Each unit has number of attributes; all of them can be easily altered through the Unit Attributes editor.



DPSP2 Registry Editor allows you to browse and alter information stored in the DPSP2 Registry.

FEATURE HIGHLIGHTS

- **Extensibility**: Dynamic PSP v2 is designed with extensibility in mind. DPSP2 can be easily extended with new features without the need to change the internal architecture of the kernel thanks to modular architecture.
- **Modularity**: DPSP2 consists of several components (modules) and each module is installed separately. Some modules are required and some are optional. New modules can always be added to the existing set, seamlessly extending the capabilities of the system.
- Comprehensive **Built-in APIs** allow to control HTTP output and server security, invoke units from other units, profile and debug the code, easily manage server-side persistent session contexts, and much more.
- **DPSP Registry** subsystem provides hierarchical persistent data storage facility similar to Windows Registry. This subsystem is extensively used by DPSP2 modules to store and retrieve their configuration information, but can be used standalone by any PL/SQL applications.
- DPSP2 has strictly defined **kernel** separated from DPSP2 projects. The kernel should exist as single instance in the database, and can be used to support unlimited number of independent DPSP2 projects in different schemas. All DPSP2 projects use shared kernel services, but operate in their own database schemas.
- **Independent Preprocessor modules** perform DPSP2 unit code processing. Each DPSP2 unit has attributes defining which preprocessor to use to generate the executable page code. DPSP2 system can have unlimited number of specialized preprocessors. FLAT, DPSP and PPSP preprocessors come standard with DPSP2. FLAT preprocessor does not do any processing and is used for static HTML/XML/JavaScript units, DPSP preprocessor implements DPSP v1 syntax and features and PPSP preprocessor implements Procedural PSP extensions to DPSP. Optional OPSP (Objective PSP) preprocessor is also available.
- DPSP2 units may be **interpreted or compiled**. Interpreted mode allows seeing the effect of a smallest change in the code immediately, while compilation of final code allows obfuscating its source using WRAP utility and speeds up its execution (orders of magnitude if native PL/SQL compilation feature of Oracle9*i*™ is used.)
- DPSP2 modules interact with each other using **Dynamic SQL**. Thus changing, upgrading or removing one module does not cause automatic cascaded invalidation of dependent modules.
- DPSP2 widely uses **synonyms**, which minimizes the risk of naming conflicts and allows effective code sharing. Synonyms are also used to seamlessly plug in DPSP modules and upgrade Kernel functionality.
- DPSP2 implements new system for **unit attributes** which allows assigning any unit any number of attributes without the need to change internal tables layout. This new system allows assigning new attributes to existing units to implement new functionality in the future.
- Centralized **exception handling** which provides a number of options for handling errors in DPSP units, including printout of the source code with error source line highlighted.
- Strict internal **security**. DPSP2 implements strict access control for DPSP units. All operations with DPSP units except execution (creating, editing, etc.) require explicit user registration in DPSP kernel and logging into the system. Units are generally executable by anonymous users unless execution of the unit through the web call is expressly prohibited (for example, if unit is reusable) or the need for authentication is explicitly set for the unit.
- **Automatic versioning and archiving** of DPSP2 units. Any unit can be reverted to its previous saved state if needed.
- **Automatic execution tracing** for DPSP2 units. Trace data can be used to gather unit execution statistics (for example, to detect unused units).

- **Session Context Subsystem** built into DPSP2 kernel allows storing session context from call to call eliminating the need to store this data on client as cookies or pass it as parameters from call to call. Each context gets its own unique identifier tied to session identifier (sessid). Session contexts can inherit some data from other contexts.
- **Extensive Internationalization Support** is provided in form of additional module (NLP, National Language Processing). This module allows creating international applications and translating user interfaces by supporting an extensible dictionary of translations. Units should not output any text to be translated directly, but rather through the translation engine, which will lookup the dictionary and output correct translated text if it can locate it in the dictionary. This module thus acts as a string resource manager.
- **Logical Names subsystem** implemented as additional module (Resolver). When installed, this module allows to assign each unit a logical name and to access DPSP2 units by their logical names rather than by their IDs. Logical names are unique and may be reassigned to different units allowing transparent unit exchange.
- **Virtual File System** (Name-Tree Service, or NTS) allows creating and maintaining file system-like hierarchies to identify and access files, DPSP units or other types of data stored in the database. NTS provides 100% web-based interface for manipulating the hierarchies and their nodes, and an API for extending basic NTS capabilities and using its features in any PL/SQL applications.
- **Dynamic Database Browser** is 100% web-based tool for browsing and manipulating (creating, dropping, altering) database objects like tables, views, indexes and stored procedures, and run ad-hoc queries against the database through user-friendly HTML-based GUI.