JOPA (Java-based Oracle PL/SQL Access)
Gateway Servlet
User's Guide and Reference

Version 1.6.1

Document ID: 001-205001-016
Last Revision: August 25, 2005

COPYRIGHT INFORMATION AND ACKNOWLEDGEMENTS

Table of Contents

INTRODUCTION

WHAT IS JOPA GATEWAY SERVLET AND WHY WE DECIDED TO CREATE IT?

Oracle® PL/SQL™ applications written for the web cannot interact with web servers directly. Special web server modules, commonly known as gateways, are used to execute the PL/SQL code processing web requests and to fetch and return the response. These gateways initially receive HTTP requests, preprocess them, determine which PL/SQL subprograms are called and which parameters are passed, connect to an Oracle server, execute requested subprograms, fetch results of their execution and return them back to the requesting client. Oracle Corp. supplies its own PL/SQL gateway module for its HTTP server (Oracle HTTP Server powered by Apache, OHS), called `mod_plsql`. This module is written to Apache module specifications in C and is loaded by the web server as shared library. Module routines are called whenever PL/SQL application is invoked and are responsible for connecting to the Oracle server, executing requested code, retrieving execution results and returning them to the client. During the evolution of this module, it received many attractive features PL/SQL web developers asked for, but it still lacks certain functionality. Most important missing feature is full control over the HTTP output, including HTTP headers, which prevents application developers from creating advanced applications adhering to the latest Internet standards. `mod_plsql` also does not provide support for WebDAV protocol in any form. `mod_plsql` is also known for its multiple security vulnerabilities discovered during the year of 2002, which also demonstrated how complex it is to maintain the module installations – Oracle had released multiple patches for multiple supported platforms and different OHS versions released on those platforms, and finding correct patches and applying them in correct order is not trivial. And last, but not the least, `mod_plsql` is written specifically for OHS and cannot be used with other web servers, whether or not they are based on Apache HTTPD code base.

The JOPA Gateway Servlet was designed to overcome all above limitations while maintaining compatibility with PL/SQL applications written to Oracle APIs and interfaces. Being written to the Java Servlet 2.0 specifications, the JOPA Gateway Servlet may be executed in any compatible environment, including Sun's reference Servlet Runner, Apache with JServ, Apache Tomcat, Oracle Containers for J2EE (OC4J, former Orion Server), Macromedia JRun, IBM WebSphere, etc. on all supported platforms (that is, virtually anywhere.) Being written in 100% Java, the JOPA Gateway Servlet is highly portable and comes in single WAR (Web Application) archive for all platforms and servers making updates and upgrades as easy as possible. The JOPA Gateway Servlet provides compatibility layer to keep your legacy PL/SQL web applications running without single change, while adding optimized layer for native Dynamic PSP™ execution and another layer for WebDAV (Web-based Distributed Authoring and Versioning) protocol support for Dynamic PSP developers. The servlet provides PL/SQL and Dynamic PSP developers with full control over HTTP output their applications generate, which allows them to overcome unwelcome `mod_plsql` HTTP output filtering and create applications fully conformant to the latest web standards whether they are created in traditional way (using `OWA` packages or Oracle PSP) or using Dynamic PSP. Dynamic PSP users will also enjoy enhanced performance of their applications due to optimized DPSP execution layer and advanced features, like progressive output and HTTP 1.1 partial requests support, implemented for native DPSP connections. In addition to WebDAV support for Dynamic PSP Repository access, the servlet also allows to export any directory in the server's file system through WebDAV. The servlet also provides web-based configuration interface allowing administrators to reconfigure the servlet online without the need for servlet container restart.

WHAT IS IN THIS DOCUMENT?

This document is your primary source of information about the JOPA Gateway Servlet. It provides information on servlet basics, installation, configuration and use. The document is divided into several chapters:

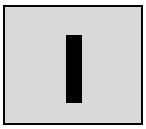
Chapter I. Installing and Configuring the JOPA Gateway Servlet.

This chapter describes the servlet deployment and initial configuration. Deployment to Apache JServ, Oracle OC4J (Oracle Containers for J2EE) and Apache Tomcat 5.x is described in detail. The chapter also provides the servlet configuration parameters reference.

Chapter II. Using the JOPA Gateway Servlet.

This chapter describes the servlet usage and dynamic configuration web interface. PL/SQL procedure call format through the servlet is described. The chapter also provides complete reference on the dynamic configuration web interface built into the servlet.

Other documents you need to familiarize yourself with are product README file, which includes important information about product installation and required Oracle software patches; and release notes accompanying each release. Release notes contain the change log for the product and should be used for quick reference on new features and important fixes for each release.



CHAPTER I. INSTALLING AND CONFIGURING THE JOPA GATEWAY SERVLET.

This chapter describes the JOPA Gateway Servlet installation and configuration. Servlet basics and principles are described in detail.

SYSTEM REQUIREMENTS.

The following are system and software requirements for JOPA Gateway Servlet:

- Java Run-time Environment (JRE) Version 1.4.0 or later (Version 1.4.2_06 or later recommended)
- Compatible servlets container (Servlet Specification 2.0 or later must be supported)
- Oracle8*i* Release 2 (8.1.6) or later database backend (Oracle8*i* Release 3 (8.1.7), Oracle9*i* Release 2 (9.2.0) or Oracle10g with the latest patch set recommended)
- 256MB RAM on application server host (512MB or more recommended)

The JOPA Gateway Servlet comes bundled with the Oracle Thin JDBC driver v9.2.0.7 and does not require Oracle client software installed on the application server host. If your servlets container loads/uses its own copy of Oracle Thin JDBC driver, you need to update the driver to the latest release (9.2.0.7 for Oracle9*i*DB at the time of this writing) to avoid problems with NLS/GSS we've seen with previous driver releases. Note that if the container loads its own copy of the JDBC driver on startup, driver bundled with the servlet will not be used.

JDOM Version 1.0 is also bundled with the servlet. Refer to http://www.jdom.org for details on JDOM and downloads.

The latest releases of the Oracle JDBC Driver are available for download at Oracle Technology Network web site (http://otn.oracle.com - the site requires free registration.)

Please note that aforementioned Oracle software components are bundled with the servlet for your convenience only and are subject to separate licensing. License for the JOPA Gateway Servlet does not automatically grant you any legal rights for Oracle JDBC Driver, you need to secure appropriate license from Oracle Corp. to legally use it.

DEPLOYING THE JOPA GATEWAY SERVLET.

You may deploy the servlet manually or automatically through the server management and deployment tools. To deploy the servlet manually, you need to copy the supplied JAR or WAR file (`jopa.jar` or `jopa.war`) to the application server and add it to the servlets container class path. Deploying servlets differs from container to container, so you need to consult your container's documentation for instructions on servlet deployment. This chapter will describe in detail how to deploy the servlet on Apache JServ and Oracle OC4J. Other containers may require different deployment procedures, though since the servlet is provided in standard WAR archive, there should be no problems deploying it to any J2EE 1.3 compatible container.

DEPLOYING TO APACHE JSERV.

Apache JServ is configured through several property files. The root property file is specified in `mod_jserv` module's main configuration file (usually `jserv.conf`) with `ApJServProperties` directive. Usually, this file is named `jserv.properties` and is located in `$APACHE_HOME/Jserv/conf/` directory. Several properties should be changed or added to this file in purpose for the JOPA Servlet to function properly:

`wrapper.bin` Set this to JRE 1.4 executable (for example, `c:\j2sdk1.4.2_06\bin\java.exe`). The JOPA Servlet **requires JRE 1.4.0** or later to function.

Note that this change may break the Oracle JSP engine – OJSP will be no longer able to compile the JSP pages because the new JRE has more recent class file format and the engine will use an older compiler. To fix this issue, you need to specify correct `tools.jar` from JDK 1.4 or later in zone properties file. Locate the line similar to the following:

```
wrapper.classpath=<ORACLE_HOME>\jdk\lib\tools.jar
```

where `<ORACLE_HOME>` is your Oracle HTTP Server home directory, and replace this path with the path to the `tools.jar` library from JDK 1.4 or later. For example,

```
wrapper.classpath=c:\j2sdk1.4.2_06\lib\tools.jar
```

Alternatively, you can add the following OJSP initialization argument (in zone properties file):

```
servlet.oracle.jsp.JspServlet.initArgs=javaccmd=<full-path-to-javac>
```

where `<full-path-to-javac>` is full path (including executable name) to the Java compiler 1.4 or later (for example, `c:\j2sdk1.4.2_06\bin\javac.exe`.)

`wrapper.classpath` Add new `wrapper.classpath` property entry and specify full path including file name to the `JOPA.war`

`zones` This property defines so-called servlet zones. There may be several zones defined, each with its own properties. Usually, one zone, called `root` is defined and is sufficient.

`<zone>.properties` Depending on zone names defined in `zones`, each defined zone should have its own properties file. These files are defined with `<zone>.properties` property, where `<zone>` is replaced with the zone name. For example, for the `root` zone, corresponding property will be named `root.properties`. This property should contain full path including file name to the zone properties file.

`wrapper.bin.parameters` Optionally specify additional parameters for the Java VM. We recommend at least adding the following parameters to be set:

`-server`	Start the JVM in server mode (better performance)
`-ea`	Enable assertions (some servlet code makes certain assertions and you would want to see them when they fail.)

`-Duser.country=US`	Start the JVM with US locale
`-Duser.language=en`	Start the JVM with English language – all error messages will be output in English rather than your current server language settings.

The zone properties file should also be edited and new servlet alias should be added to it. Assuming that you have single zone defined, `root`, and properties for this zone are stored in `zone.properties` file (usually located in `$APACHE_HOME/Jserv/servlets/` directory), you need to open this file and do the following modifications to it:

Add the following entries:

```
### JOPA configuration start
servlet.jopa.code=com.nnetworks.jopa.JOPA
servlet.jopa.initArgs=cfgfile=path/to/jopa.cfg
### JOPA configuration end
```

The above additions specify new alias for the JOPA Servlet, `jopa`, and specify which class is the main servlet class (`com.nnetworks.jopa.JOPA`). They also provide initialization arguments for the main servlet class. `cfgfile` is mandatory argument – it specifies the location of the JOPA Servlet configuration file, which contains other servlet parameters and DADs (Database Access Descriptors) definitions. The configuration file is an ASCII text file, which contains servlet configuration parameters. Its syntax is similar to the Windows .INI file syntax – the file is broken into sections; each section is delimited with section name in square brackets. Within each section there is one or more parameters related to that section. Each parameter is on separate line in `parameter_name=value` form. Parameter names are case-sensitive.

`[Config]` section defines global servlet configuration. Below is the list of all recognized servlet parameters, their meaning and allowed values:

`log_prefix`	Defines a sequence of characters with which all log entries will be prefixed. Default is blank (no prefix.)
`error_prefix`	Defines a sequence of characters with which all error log entries will be prefixed. Default is `**` (double asterisks.)
`logfile`	Log file name and location. If `loglevel` is higher than 0, the servlet will log its actions to the specified file. This parameter is optional and defaults to `<path/to/cfgfile>/jopa.log` (`<path/to/cfgfile>` is extracted from `cfgfile` parameter.) It may also be specified in servlet initialization property (`servlet.<servlet-alias>.initArgs`.)
`loglevel`	Logging level. Defines level of verbosity for servlet actions logging. Valid range of values is 0..5. 0 turns logging off, 5 is the most verbose level. Note that critical errors (unexpected exceptions, etc.) are still logged to the log file even if `loglevel` is set to 0. Such errors are prefixed with a sequence of characters defined by `error_prefix` parameter. This parameter is optional and defaults to 0 (no log.)
`debuglevel`	Console logging level. Defines level of verbosity for servlet actions log that goes to the system console. Valid range of values is 0..5. 0 turns logging off, 5 is the most verbose level. This parameter should be set to 0 in production environments and is only useful with text-mode servlet runners. This parameter is optional and defaults to 0 (no log.)

You now need to restart the Apache daemon, so that it will reload `mod_jserv` and the module will re-read its configuration and recognize new settings for JRE, class path and new servlet definition. Servlet deployment is now complete and the JOPA Servlet is accessible at `/<servlets location>/jopa/`, where `<servlets location>` is location configured in Apache to be handled by `mod_jserv`. However, you will also need to define some DADs before you will be able to use it. DAD definition format is discussed further in this chapter.

DEPLOYING TO OC4J (ORACLE CONTAINERS FOR J2EE) STANDALONE.

Unlike Apache JServ, OC4J Standalone follows the J2EE specifications and is configured through XML configuration files.

To deploy the JOPA Gateway Servlet on OC4J, you should first copy the `jopa.war` archive to the `{oc4jhome}/j2ee/home/applications` directory, modify `{oc4jhome}/j2ee/home/config/http-web-site.xml` file by adding new `<web-app>` entry to the `<web-site>` section:

```
<web-app application="default" name="jopa" root="/<application-root>" />
```

and restart the OC4J. `<application-root>` can be any prefix of your choice that will be used to access your PL/SQL applications. For example, if you set it to `/pls`, you will access your PL/SQL applications through the JOPA Servlet as `http://server/pls/<servlet_mapping>/<DAD>/<query>`. `<servlet_mapping>` is customizable, too (see below.)

When you restart OC4J, the `jopa.war` archive will be automatically expanded into the `{oc4jhome}/j2ee/home/applications/jopa/` directory. You now need to modify the `WEB-INF/web.xml` file in that directory to set correct value for the `cfgfile` servlet initialization parameter. Locate

```
<init-param>
 <param-name>cfgfile</param-name>
 <param-value>jopa.cfg</param-value>
</init-param>
```

tags in the `web.xml` file and alter the `<param-value>{path_to_config_file}</param-value>` tag to contain full path to the servlet configuration file (default is `jopa.cfg` with no path, which resolves to the directory, from which the JRE was started). You can store the configuration file anywhere on your file system. The default configuration file will be in `{oc4jhome}/j2ee/home/applications/jopa/` directory when OC4J auto-extracts the WAR archive, so you can set the `<param-value>` tag to `{oc4jhome}/j2ee/home/applications/jopa/jopa.cfg` to use the default configuration file.

The servlet is now ready – OC4J will automatically reload the application configuration and re-deploy the servlet when you save the `web.xml` file.

You can also alter `<servlet-mapping>` to use in gateway access URLs by modifying the `<url-pattern>` tag value under `<servlet-mapping>` tag in the `web.xml` file. Locate

```
<servlet-mapping>
  <servlet-name>jopa</servlet-name>
  <url-pattern>/jpls/*</url-pattern>
</servlet-mapping>
```

tags in the `web.xml` file and edit the contents of the `<url-pattern>` tag. After you save the file, OC4J will automatically re-deploy the servlet on next access and new mapping will be used. Default mapping is `/jpls/*`, so by default you will access your PL/SQL applications through an URL similar to this:

`http://server/<application-root>/jpls/<DAD>/…`

DEPLOYING TO APACHE TOMCAT 5.X.X

Apache Tomcat 5.x.x is the JServ offspring fully conformant to the J2EE 1.3 and later specifications. Deployment to the Tomcat server is very simple:

Copy the `jopa.war` file to the `{CATALINA_HOME}/webapps` directory or upload it through the Tomcat management interface. The Tomcat server will detect the new WAR and deploy it automatically. However, before the JOPA Servlet can be used, you need to edit the `WEB-INF/web.xml` file in the `{CATALINA_HOME}/webapps/jopa` directory, where the servlet will be automatically deployed, and set `cfgfile` initialization parameter to contain correct full path to the JOPA configuration file. Save the `web.xml` file and the Servlet will be fully functional.

DEPLOYING TO OTHER J2EE 1.3 AND LATER CONTAINERS.

Consult with your container manual for instructions on web applications deployment. You should use `jopa.war` WAR archive for deployment to J2EE 1.3 and later containers; `jopa.jar` archive is only intended to be used with Apache JServ. If the deployment tools supplied with the container allow modifying the servlet initialization parameters during deployment, make sure you set `cfgfile` parameter to point to correct configuration file location, otherwise you will need to set it manually in the `WEB-INF/web.xml` file (see the section on deployment to OC4J Standalone or the section on deployment to Apache Tomcat for details.)

CONFIGURING JOPA DATABASE ACCESS DESCRIPTORS (DADS).

Before your PL/SQL or Dynamic PSP web application can be accessed through the JOPA Gateway Servlet, the servlet Database Access Descriptor (DAD) should be configured so that the servlet knows how to connect to the database where the application resides and how to retrieve responses. All DADs are configured in single ASCII text file. Name and location of this file are specified in `cfgfile` servlet initialization argument. The configuration file layout is as follows:

```
[Default]
# global settings
parameter=value
parameter=value
…

[DAD1]
# DAD1-specific settings
parameter=value
parameter=value
…

[DAD2]
# DAD2-specific settings
…
```

Keyword in square brackets starts new configuration section. `[Default]` section defines parameters common to all DADs. It is also used to resolve parameters not set explicitly for particular DAD. `[Default]` is a special section in the configuration file. All other sections are DADs themselves. Word in the square brackets becomes the DAD name (case-insensitive) and all parameter-value pairs following it until the next section or the end of file become this DAD's parameters.

Unlike DAD names, parameter names are case-sensitive. Empty lines and unrecognized parameters in the DAD configuration file are ignored. At run time, if any particular parameter is unresolved in DAD configuration, value set in `[Default]` section is used, and servlet internal default is used if `[Default]` section does not set this parameter, too. Below is the list of all recognized parameters, their purpose and allowed values.

`username` Oracle user name to use for database connection or servlet administrator user name for servlet administration pages.

For UDAV DADs this is the username to access exported directory.

`password` Password to use for database connection or servlet administrator password for servlet administration pages.

For UDAV DADs this is the password to access exported directory.

Passwords can be entered in plain text, they are automatically encrypted when the servlet loads the configuration file. Passwords entered through the online administration interface are encrypted right away.

`host` Database host to connect to (DNS name or IP address).

`port` IP port the TNS listener is listening to (default value is 1521).

`sid` Oracle database SID.

`mode` PL/SQL call mode. Allowed values are:

`Native` JOPA uses native Dynamic PSP calls to process requests. If the application is pure Dynamic PSP, using native calls will significantly improve performance. Advanced servlet features, like progressive output support, are only available in native mode. However, execution of non-DPSP procedures is

not supported in this mode.

This is default value.

`OWA` JOPA uses OWA calls and emulates mod_plsql when processing requests. If your application is written to OWA or mixes legacy OWA code and Dynamic PSP, the DAD should be configured for this mode.

`WebDAV` JOPA accepts and processes WebDAV requests and exports Dynamic PSP Repository contents through WebDAV on this DAD. You cannot execute either legacy or DPSP applications through this DAD. Dynamic PSP with NTS is required on the backend.

`Admin` This DAD is used for online servlet configuration. No Oracle connection is established on this DAD, the servlet processes all requests internally. The Administration DAD is used to configure the servlet online, through the web browser, without the need to edit the servlet configuration file and restart the servlet container. `username` and `password` attributes for this DAD are used for Administration Interface access authorization.

Usually, there should be only one Admin DAD on the server.

`UDAV` JOPA accepts and processes WebDAV requests and exports local file system directory through WebDAV on this DAD. You cannot execute either legacy or DPSP applications through this DAD.

`charset` Force request and response character set. When set to an IANA character set name, this parameter overrides default request character set (which is usually determined from the locale of the system where the servlet is started because there are no provisions in the HTTP protocol for communicating the request character set from the client) and response character set (which may be set by your PL/SQL or Dynamic PSP program.)

`reset_mode` PL/SQL state reset mode. The JOPA Servlet maintains its own pool of Oracle connections and reuses connections from this pool. Consequently, it needs to reset PL/SQL state before reusing the connection for new request processing. Since Oracle8i 8.1.7.2, new lightweight mechanism for resetting PL/SQL package state was introduced. If the backend runs Oracle8i 8.1.7.2 or later, you can take advantage of this mechanism, which runs faster and preserves session cache, otherwise you should use traditional mechanism, which resets all PL/SQL session state and flushes cache.

Allowed values for this parameter are:

`full` Use standard PL/SQL state reset mechanism.

This is the default value.

`fast` Use lightweight state reset (only for 8.1.7.2 and later.) If the servlet will be unable to use this mechanism, it will automatically fall back to full state reset mechanism.

`array_type` Defines SQL table type that should be used for Flexible Parameter Passing. Note that this must be an SQL table type, not a PL/SQL index-by table type. Oracle Thin JDBC driver does not support PL/SQL index-by tables, hence this restriction. We recommend using `NN$VARCHAR_ARRAY`, which is default. However, if the schema that will be accessed through this DAD doesn't own this type, explicit `EXECUTE` privilege must be granted by the type owner and you should prefix the type name with type owner name (type synonyms are only supported in 10g and later.)

Default is `NN$VARCHAR_ARRAY`. The type definition is as follows:

```
CREATE OR REPLACE TYPE NN$VARCHAR_ARRAY AS TABLE OF VARCHAR2(32767);
```

Important note: for `OWA` DADs you still must declare an SQL table type similar to `NN$VARCHAR_ARRAY` and set `array_type` to the name of this SQL table type – because

Oracle Thin JDBC driver does not support PL/SQL index-by tables, the servlet cannot use them and needs an SQL table type to pass array arguments to your procedures.

`default_page` Relative URL for default (also known as welcome) page to be displayed when the DAD root is accessed (that is, a request is made to `http://server/servlets/jopa/DAD/`). The servlet will automatically redirect the requestor to the specified default page.

Default is blank (no default page.)

`document_path` The document access prefix. A string designating access to the binary documents either through the document download procedure or, if it is not defined, directly from the document table. Document path follows the DAD name in the request URL. Request like

`http://server/servlet/jopa/DAD/document_path/some.doc`

will cause the servlet to call the document download procedure defined in `document_proc`, or attempt to locate and fetch `some.doc` in the document table if the document procedure is not defined. The document procedure should have no non-default parameters and should be able to decode query string and return the requested document or return valid HTTP error code through `Status:` HTTP header, or raise an exception if this is not possible. If the document procedure is not defined for the DAD, the servlet will attempt to locate the requested file in the document table using document table's `NAME` column to resolve the file name.

Default is blank (no document access prefix.)

`document_table` Defines the name of the document table. Document table is special table to which all binary content is uploaded. This table should have at least the following columns:

```
NAME           VARCHAR2(256) NOT NULL,
MIME_TYPE      VARCHAR2(128),
DOC_SIZE       NUMBER(10),
DAD_CHARSET    VARCHAR2(128),
LAST_UPDATED   DATE,
CONTENT_TYPE   VARCHAR2(128) DEFAULT 'BLOB',
BLOB_CONTENT   BLOB
```

When a binary file is `POST`ed through a `multipart/form-data` HTML form, the servlet will decode the form data and insert the posted file into the document table. `NAME` column will receive the file name, `MIME_TYPE` will receive document's MIME type, `DOC_SIZE` will receive the file size, and `BLOB_CONTENT` will receive document's content. `LAST_UPDATED` timestamp column receives database current time when file is uploaded. This timestamp column may be used in the document download procedure in conjunction with HTTP `If-Modified-Since:` header to verify that the requested document was not changed since last request and the client may use its cached copy. If the document download procedure is not specified, the servlet does this verification automatically. After insert the form handler procedure receives the name of the file in corresponding form parameter, and may use this name to locate the file in the document table. The procedure may rename the file by updating the `NAME` column, as well as alter other properties of the document, like its MIME type.

Default document table, `NN$T_DOWNLOAD`, should be used for Dynamic PSP DADs.

Default is blank (no document table.)

`document_proc` Defines the name of the document download procedure. This procedure is automatically called when a request through document path is done. The procedure should have no non-default parameters and should be able to decode the HTTP query and return the requested document or return valid HTTP error code through `Status:` HTTP header, or raise an exception if this is not possible. For Dynamic PSP DADs, default download proce-

dure NN$PSP_RSP.DOWNLOAD should be used. If NTS is installed, NN$NTS_OWA_DOWNLOAD should be used if you want to be able to use NTS paths for addressing the documents.

Default is blank (no document download procedure.)

`flex` Number of parameters accepted by procedures following Flexible Parameter Passing convention. Valid values are 2 or 4. 4-parameter calls, where additional array size parameter specifying the array size for both parameter names and values arrays and special reserved parameter were passed, are deprecated and should not be used, but if your legacy code uses this old call format, you should set `flex` to 4.

Default is 2 (two arrays, for parameter names and values, respectively.)

`exclusion_list` For **OWA** DADs, this parameter contains the list of comma-separated regular expression patterns to be used when checking for stored procedures excluded from web access. The JOPA Servlet automatically denies access to certain packages and procedures that are not supposed to be accessible from the web and may leak sensitive information or allow an unauthenticated user to alter the database state in undesired ways. By default, the servlet excludes the following patterns:

```
SYS\..*,DBMS_.*,UTL_.*,OWA_UTIL\.SHOWSOURCE,OWA_UTIL\.CELLSPRINT
```

If you specify your own patterns in this parameter, they will be appended to the default list; you do not need to replicate the default list. You can also disable exclusions completely (including defaults) by specifying #NONE# as the value for this parameter (but you should only do this in protected development environments and never set this value on systems exposed to public access.)

`compression_enabled` HTTP 1.1 protocol allows clients to request that the response is compressed using GZIP or Deflate algorithm to save bandwidth. Usually, browsers capable of decompressing such content automatically request that the response is compressed. This parameter controls whether the servlet honors such requests and compresses output with requested method. Allowed values for this parameter are `0` (disabled) and `1` (enabled.)

Default is `0` (disabled.)

Note that once enabled, compression will be applied to **all** documents served through this DAD, even those that may already be compressed (for example, archives stored as BLOBs in the database.) The servlet skips compression only for media types starting with "`image/`" or containing "`compress`" in their media type string because such media is usually already compressed. If you have compressed files whose media type doesn't match one of this cases, it will be compressed again while sent, probably with no additional size reduction.

`upload_limit` Defines form size limit for HTTP POST method. Any form larger than this limit will be rejected. You can use this parameter to limit size of file uploads.

Default is blank (no limit.)

`progressive_output` Enables or disables progressive output support for **`Native`** mode DADs. When enabled, Dynamic PSP output will be sent to the client immediately as it is generated; otherwise it will be buffered in the database and then sent in whole. This feature may greatly reduce perceived page load times as the client will start receiving content almost immediately as opposed to standard mode of operation of the mod_plsql and JOPA Servlet, where all output is first accumulated in server-side buffer and only when the PL/SQL call completes the buffered output is fetched and streamed to the client. However, when enabled, this feature will add load to the database as it requires two simultaneous database connections per request – one for executing the request and another for fetching the output as it becomes available.

Possible values for this parameter:

- `0` progressive output disabled.
- `1` progressive output enabled.

Default is `0` (disabled.)

`localbase` For **UDAV** DADs this parameter should contain the full path to the directory that will be exported as root. You will then access this directory and its subdirectories via WebDAV.

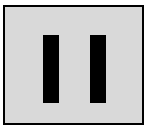
This parameter is mandatory for **UDAV** DADs.

`lock_table` For **UDAV** DADs this parameter should contain full path and name of the file to be used as lock table. Lock table contains information on locks put on resources accessed through WebDAV.

This parameter is mandatory for **UDAV** DADs.

`user_db` For **UDAV** DADs, this parameter may point to the XML-based user database that will be used for authentication in addition to the DAD username/password. This parameter allows more fine-grained control over access as each user may be assigned different access rights (read-only or read-write) to the underlying resources.

This parameter is optional.



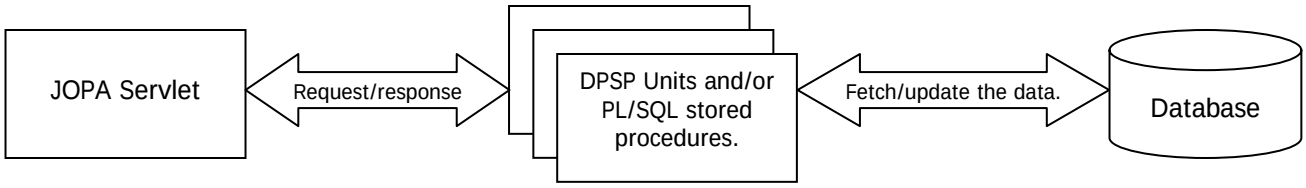
CHAPTER II. USING THE JOPA GATEWAY SERVLET.

This chapter describes how to use the JOPA Gateway Servlet to execute legacy PL/SQL applications written for OWA, Dynamic PSP applications, and administer the servlet through the built-in administration pages.

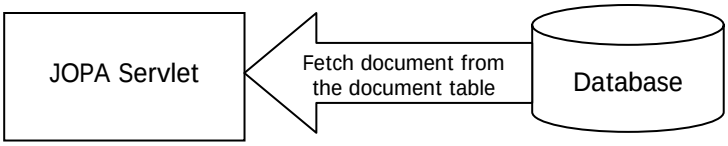
HOW THE SERVLET PROCESSES HTTP REQUESTS.

In terms of popular MVC (Model-View-Controller) pattern, the JOPA Gateway Servlet acts as the front controller: it initially receives all requests, pre-processes them, determines which model implementation unit or procedure should be called to process the request and dispatches the request to the corresponding model implementation unit or stored procedure. Called units then update the model and select next view to display. Called units are also responsible for rendering the next view. The Servlet then retrieves rendered view and presents it back to the client. The Servlet also supports requests for binary document downloads – the documents are stored in the document table and may be retrieved without any further processing. Files uploaded as part of an HTML form encoded using `multipart/form-data` encoding are automatically extracted from the form data and inserted into the document table specified in the DAD configuration. The request processing may be diagrammed as follows:

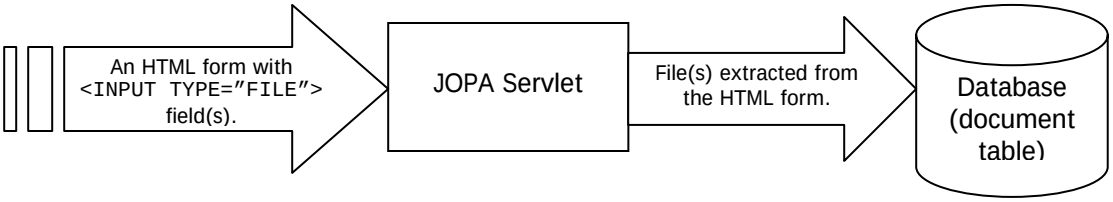
PL/SQL Procedure or DPSP Unit execution request



Document/file download request



File uploads via `multipart/form-data` encoded HTML forms



CALLING PL/SQL PROCEDURES THROUGH THE JOPA GATEWAY SERVLET.

PL/SQL calls through the JOPA Gateway Servlet are performed by accessing the servlet with specially formed URLs. The URL should be as follows (wrapped for reader's convenience, should actually be contiguous, parts in square brackets are optional, pipe (|) character separates alternate choices):

```
[<proto>://]server/<servlet-loc>/<jopa-alias>/DAD/[doc_access_path/[nts_path/]docname]|
                        [!][<schema>.][<package>.]]procedure[?parameter=value…]
```

in all access modes or

```
[<proto>://]server/<servlet-loc>/<jopa-alias>/DAD/[doc_access_path/[nts_path/]docname]|
                        <NTS-path-to-DPSP-unit>[?parameter=value…]
```

in Dynamic PSP Native access mode only.

URL parts are:

<proto>:// (optional)	The protocol used to access the server. Default is `http://`, can also be `https://` to specify secure access through SSL.
server	Server name or IP address.
<servlet-loc>	Servlets alias on HTTP server. A location, which is handled by the Java servlet container.
<jopa-alias>	JOPA Gateway Servlet alias registered with the Java servlet container.
DAD	Database Access Descriptor name.
! (optional)	Optional exclamation sign, designating that Flexible Parameter Passing convention is being used to pass parameters to the PL/SQL procedure. In OWA mode, the servlet automatically attempts to execute the procedure using flexible parameter passing if direct call fails due to missing or inappropriate arguments, but this adds unnecessary describe operation and second call. To avoid this overhead, it is recommended to always supply the exclamation sign when calling FPP procedures.
<schema>. (optional)	Optional Oracle schema (procedure owner) prefix for the PL/SQL procedure being called.
<package>. (optional)	Optional PL/SQL package name containing the called procedure.
procedure	Name of the PL/SQL procedure being called.
?parameter=value… (optional)	Optional parameters to be passed to the procedure.
doc_access_path/nts_path/docname (optional)	Optional document access path prefix and the document name. When present, indicates access to the binary document stored in the document table or accessible through the document download procedure. The `doc_access_path` should match the document access prefix configured for the DAD. Optional `nts_path` is the NTS path to the document from the NTS root.
<NTS-path-to-DPSP-unit>	NTS path to the Dynamic PSP unit to be executed (Dynamic PSP Native access method only.)

EXAMPLES OF JOPA URLS.

```
http://server/servlets/jopa/DAD/!package.procedure?arg1=val1&arg2=val2
```

Classic PL/SQL execution request. `PACKAGE.PROCEDURE` will be executed, and `arg1` and `arg2` will be passed to it using Flexible Parameter Passing convention.

```
http://server/servlets/jopa/DAD/docs/document.doc
```

Classic document access request. `docs` is the document access prefix configured for the `DAD`. The `document.doc` will be searched in the document table and returned.

```
http://server/servlets/jopa/DAD/nts/path/unit?arg1=val1&arg2=val2
```

Native Dynamic PSP request. `/nts/path` is the path in the NTS to the unit. The `DAD` should be configured as Native.

```
http://server/servlets/jopa/DAD/docs/nts/path/document.doc
```

Document access request using NTS. `docs` is the document access prefix configured for the `DAD`, and `/nts/path` is path in the NTS to the document..


```
http://server/servlets/jopa/DAD/!go?ln=/nts/path/unit&arg1=val1&arg2=val2
```

Classic Dynamic PSP request. `!go?ln=` is the standard Dynamic PSP unit invocation, `/nts/path` is the NTS path to the unit requested.

ADMINISTERING THE JOPA GATEWAY SERVLET THROUGH THE BUILT-IN INTERFACE.

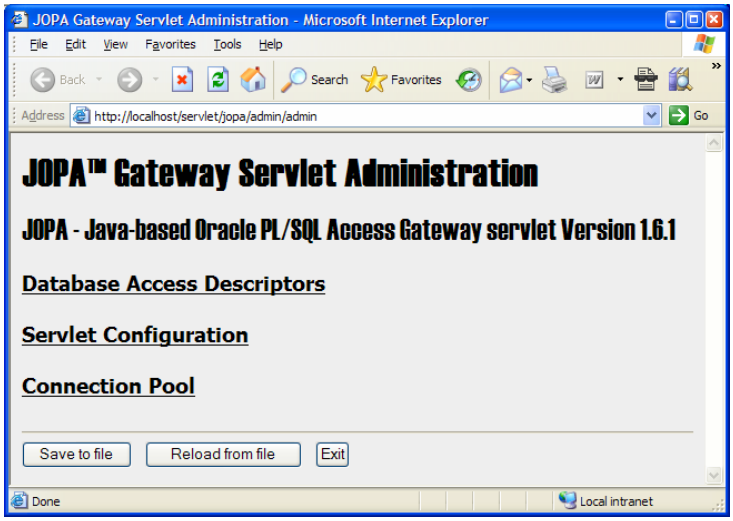
When initially configuring the servlet, you may create special Admin DAD (`mode=Admin`). This DAD then may be used to reconfigure the servlet dynamically through the web interface. Such DAD cannot be used for any other purpose, i.e. you cannot use it for connecting to your PL/SQL or Dynamic PSP applications. To access the built-in administration interface, point your browser to the URL like this:

`[http://]server/<servlets>/<jopa-alias>/AdminDAD/`

You will be prompted for username and password. Enter username and password you specified for the AdminDAD DAD (username and password are case-sensitive.) If you entered correct credentials, you will be presented with the servlet administration interface. Through this interface you can create, edit and delete DADs, monitor the connection pool activity and change servlet global configuration parameters. Changes will be in effect for all new requests immediately after you save them, without the need to restart the servlet container or the web server. Additionally, you can make all these changes permanent by flushing them to the configuration file.

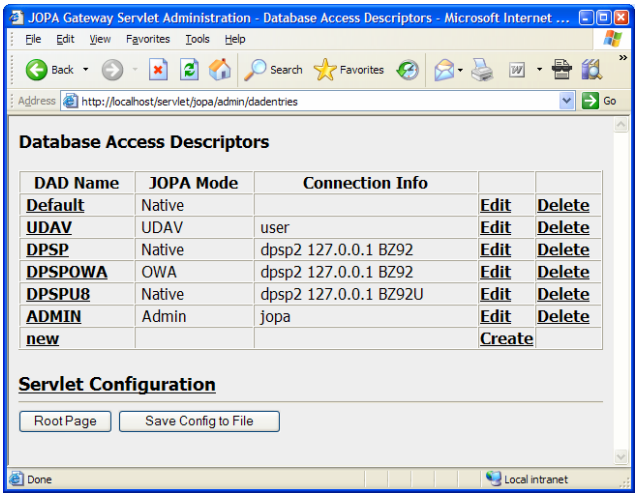
MAIN ADMINISTRATION INTERFACE PAGE.

This page provides access to three main administration areas: DAD Administration page, Servlet Configuration page and Connection Pool Configuration page:



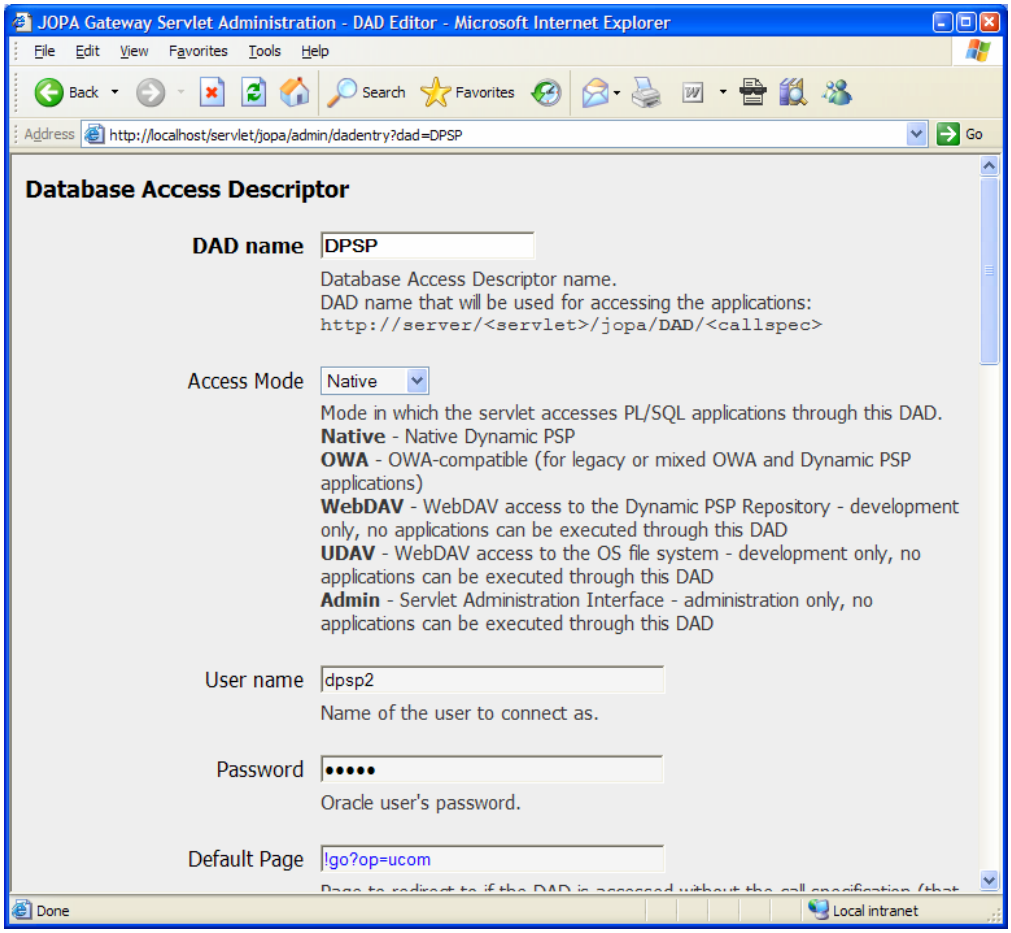
DAD ADMINISTRATION PAGE.

This page displays all DADs currently defined in the configuration file. You can also alter default configuration parameters for DADs, create new, edit and delete existing DADs here.



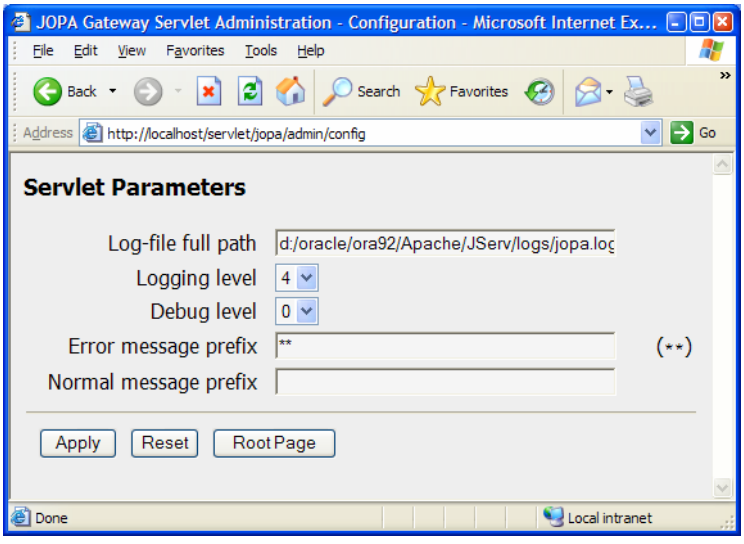
DAD EDITOR PAGE.

On this page you edit all DAD parameters. Each field is accompanied by a short description explaining its purpose and possible values. See Configuring JOPA Database Access Descriptors for detailed explanation of each DAD parameter.



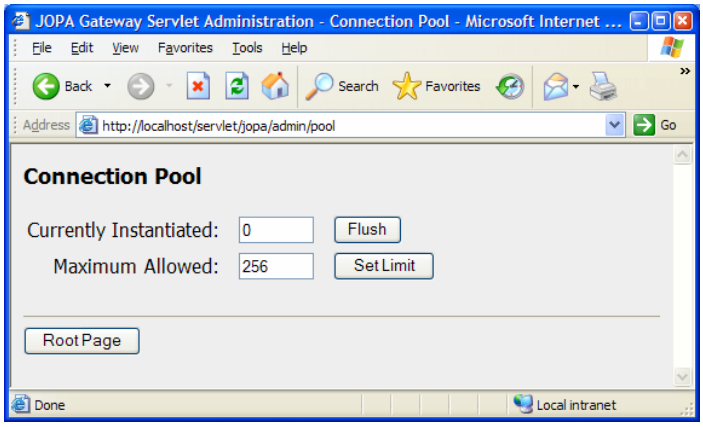
SERVLET PARAMETERS PAGE.

On this page you can edit and save the servlet initialization parameters (except for configuration file location, which is set in the J2EE servlet container configuration.)



CONNECTION POOL CONFIGURATION PAGE.

This page shows current connection pool utilization and allows flushing the pool (that is, closing and destroying all currently pooled connections) and changing maximum number of pooled connections the servlet may maintain at any given time.



UNDOING AND PERMANENTLY SAVING THE CHANGES TO THE CONFIGURATION.

All changes you make through the administration interface are active only during the servlet run time. To permanently save them in the configuration file, you should return to the main administration page and click **Save to File** button. This will make your changes persistent. Similarly, to undo the changes you made and did not yet save to the configuration file, click on **Reload from File** button on the main administration page – the configuration file will be re-read and all changes since last save will be discarded.

USING WEBDAV MODE.

WebDAV, or Distributed Authoring and Versioning through the Web, is the popular protocol supported by many web development tools, like Microsoft® FrontPage® and Macromedia® DreamWeaver™, which allows remote access and manipulation on the website content through the extensions to the HTTP protocol. The JOPA Gateway Servlet provides full support for WebDAV Level 1 and Level 2 operations, including authentication, node locking and unlocking, version control, etc. Support for WebDAV is implemented through special DADs configured to WebDAV mode. In this mode, the DAD cannot be used for normal operations, like executing PL/SQL procedures or DPSP units.

WebDAV mode requires DPSP NTS module to be installed and operational on the backend, because the site structure as seen through the WebDAV is maintained in the NTS tree.

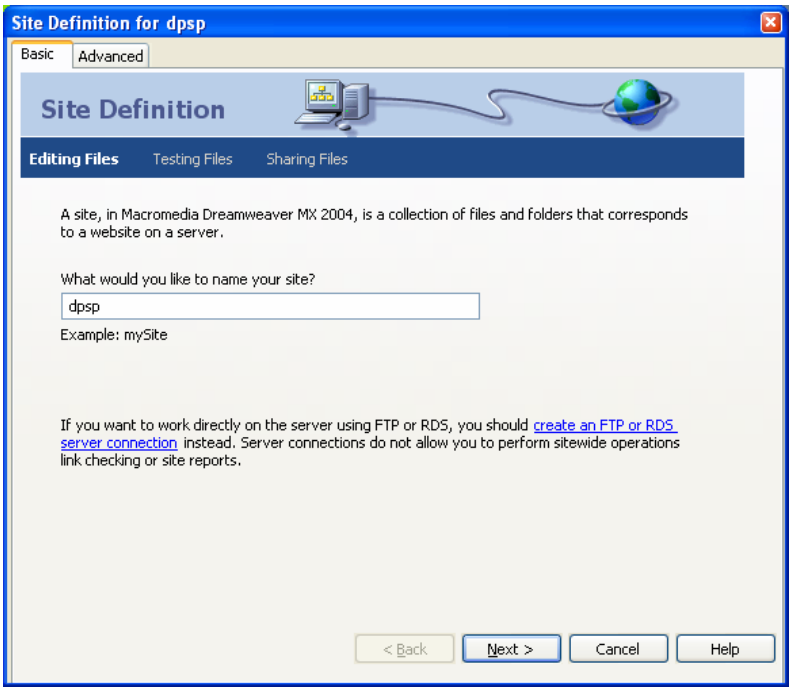
CONFIGURING THE DAD FOR WEBDAV.

You can create and configure the WebDAV DAD through the online administration interface or directly in the configuration file. The following parameters should be set: `mode=WebDAV`; `username`, `password`, `host`, `port` and `sid` should point to the correct Oracle instance and schema. Dynamic PSP with NTS should be installed on the backend Oracle database server and the schema this DAD will point to should be configured for Dynamic PSP and NTS.

CONFIGURING THE CLIENT SOFTWARE.

We will use Macromedia DreamWeaver MX 2004 as an example of the WebDAV client software, other software may have different dialogs and field names, but the overall setup will be the same.

First, create a new DreamWeaver site:

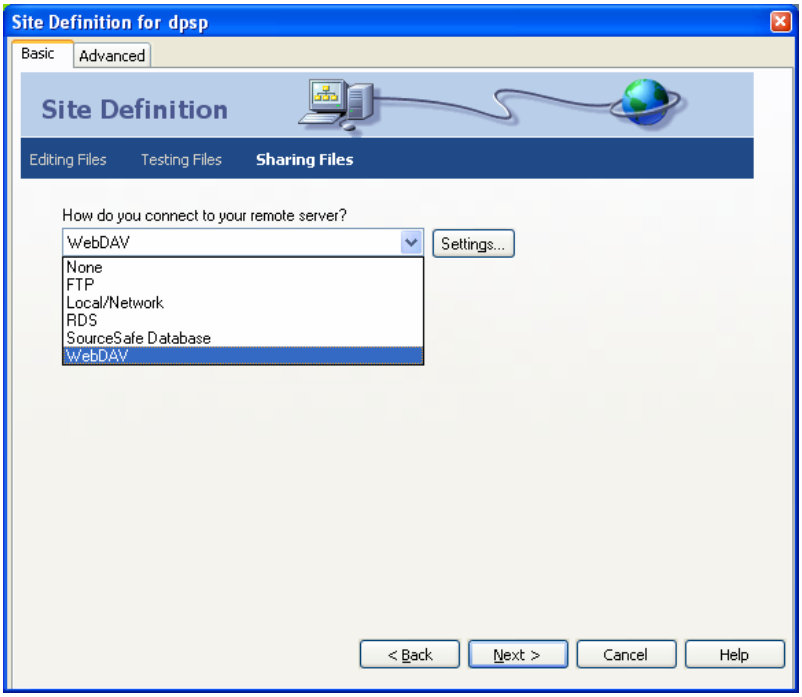


Enter any name for your new site (`dpsp` in this example.) Click the **Next** button to continue.

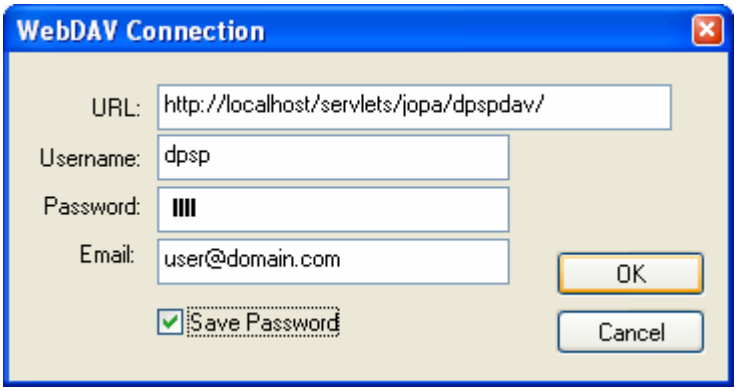
On the next screen, select your desired option for the server technology (you may download the DreamWeaver extension for Dynamic PSP Server Technology support from our web site at http://www.dynamicpsp.com.) If you did not install the Dynamic PSP Server Technology extension, leave this option as **None**.

On the next screen, select how you want the site to be edited – locally or remotely. If you will edit the site locally, DreamWeaver will automatically synchronize your local copy with remote copy when you establish connection with the remote site.

On the next screen, select WebDAV as the site connection:

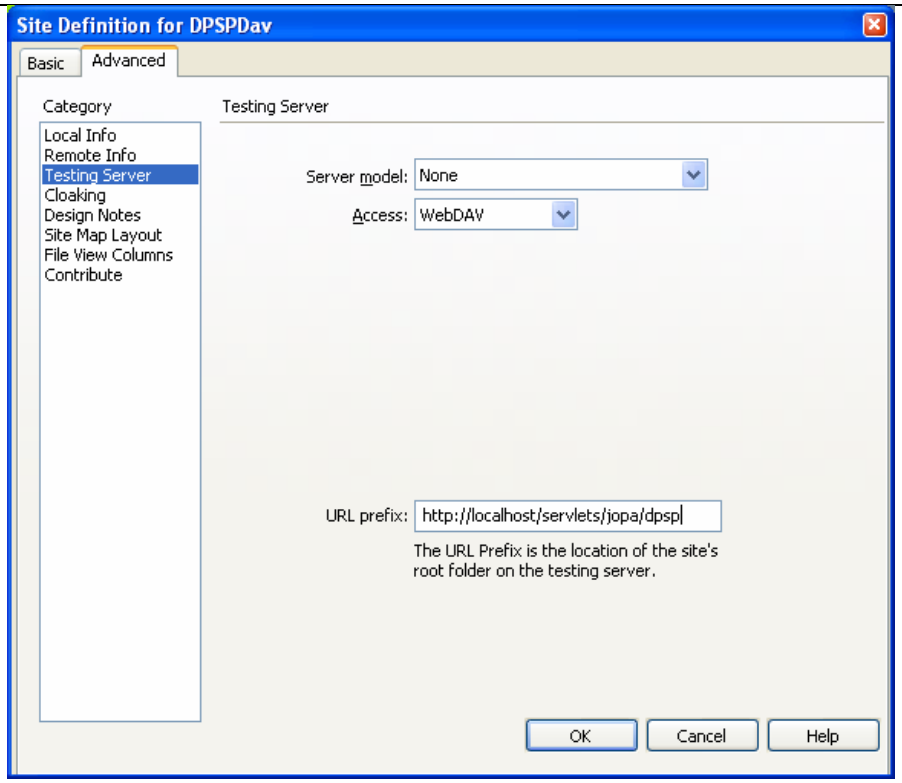


Configure the WebDAV connection by clicking the **Settings...** button and typing in necessary connection information:



Type in the URL to the WebDAV DAD you have created and authentication information that will be used to access the DAD (username and password should be registered in Dynamic PSP; these are **not** Oracle username/password.) Note that for proper locking functionality the Email field should not be left blank, because it is used for identifying who locks the content. Click **OK** to save the connection configuration.

Next, you may want to setup testing server or finish the configuration. To setup testing server, switch to the **Advanced** tab, select **Testing Server** category, select WebDAV as access method and type in the URL prefix that will map to a Native mode DAD, which will access the same schema as the WebDAV DAD:



You should not use the same URL for both WebDAV and Testing Server, because WebDAV DADs do not support normal requests – they return pages source code without interpreting them. If you do not plan to test your designs from within the DreamWeaver, you may skip testing server setup altogether.

You now have the DreamWeaver site configured for WebDAV access. To test the setup, connect to the remote site using the connect button in the **Files** panel – you should see the connection progress dialog and then the tree view of the remote site (matching the NTS tree for the target schema.)

NOTES