# DPSP™ – DYNAMIC PL/SQL™ SERVER PAGES
# DPSP REGISTRY REFERENCE

**VERSION 2.0 BETA 1**



Document ID: 010-200002-003
Last Revision: February 28, 2002

**COPYRIGHT INFORMATION AND ACKNOWLEDGEMENTS**

# INTRODUCTION

## WHAT IS DPSP AND WHY WE DECIDED TO CREATE IT?

DPSP™, or Dynamic PSP™, is PL/SQL™ Server Pages interpreter and compiler designed for Oracle8*i*™ RDBMS and Oracle® Application Server (OAS)/Oracle9*i*™ Application Server (iAS), a simple yet very powerful server-side scripting solution for Oracle AS. It is installed into Oracle8*i* RDBMS as several PL/SQL packages and Java™ classes and is instantly available after that, provided that OAS/iAS OWA packages are already installed and target schema is published via iAS. DPSP units are created, edited and managed via web interface requiring no new software installation for developers beyond any HTML4.0/DHTML-compliant browser. Optional Java-based Unit Editor is also provided with enhanced user interface.

Although Oracle has a term 'PSP' with the same meaning of 'PL/SQL Server Pages', it differs seriously from what you will read about in this document. After a bit of playing with Oracle's PSP we decided that its way is too limited in functionality we needed, so we decided to create our own server-side extension to Oracle RDBMS/AS and devised it *Dynamic PSP* (DPSP) as this name best describes what we intend to achieve but sets us aside from Oracle's PSP approach. We believe Dynamic PSP is a viable alternative to Oracle's PSP and JSP as well, because it doesn't require experienced Oracle PL/SQL programmers to learn Java and doesn't limit them in the way they can get things done.

DPSP shares common syntax with Oracle's PSP, but extends it with some helpful features, like a number of predefined functions, dynamic execution (unlike Oracle's one-time compilation approach), unlimited number of parameters to any DPSP unit (in PSP, the number of parameters is fixed), and more, while preserving syntax and common functionality of PSP.

## DPSP REGISTRY MODULE

DPSP Registry module is a standalone component which implements persistent hierarchical data storage facility similar to Windows® Registry. DPSP Registry module is independent from the rest of Dynamic PSP modules, while DPSP Kernel and other modules are dependent on storage facilities provided by the DPSP Registry. The DPSP Registry creates and maintains a tree-like hierarchical structure and allows to insert, edit, rename or delete nodes (keys) and leaves (items) in the tree. DPSP Registry exposes PL/SQL API for manipulating the Registry tree and data. Each key is identified by a combination of numeric ID and character case-sensitive name. Data items are identified by the ID of their parent key and their own character case-sensitive name. Each key can have unlimited number of sub-keys and data items. Most of the API works with numeric IDs of keys, but special functions and procedures are provided for simple name-based access to Registry keys and data items. DPSP Registry comes with Java2-based GUI tool for Registry editing (called Regedit after the Windows utility for editing Windows Registry).

 

## DPSP REGISTRY API.

The DPSP Registry API is exposed as PL/SQL package named NN$REG. Below is description of all data types, global variables, procedures and functions exposed by the API.

## DPSP REGISTRY DATA TYPES.

NN$REG package defines the following data types:

```
NAME_ARRAY   is table of varchar2(64)    index by binary_integer;
```

This type is used to store a list of registry key or item names.

```
STRING_ARRAY is table of varchar2(32767) index by binary_integer;
```

This type is used to store character data lists.

```
NUMBER_ARRAY is table of number          index by binary_integer;
```

This type is used to store numeric data lists.

```
ITEM_RECORD  is record ( NAME varchar2(64), DATA varchar2(32767) );
```

This type is a fast cast of internal data item representation to character representation. Note that `DATA` component may contain truncated value if it is longed than 32Kb.

```
ITEM_ARRAY   is table of ITEM_RECORD     index by binary_integer;
```

This type is an array of cast item records.

## DPSP REGISTRY GLOBAL VARIABLES.

NN$REG package defines the following global variables:

```
sError  varchar2(32767);
```

This variable stores the textual description of the last error occurred while calling the API. It is updated whenever an error occurs while accessing the DPSP Registry and describes the cause of the problem.

## DPSP REGISTRY FUNCTIONS AND PROCEDURES.

The following functions and procedures are exposed by the NN$REG package (grouped by functionality):

### KEY MANIPULATION.

```
openKey (nKey in number, sName in varchar2) return number;
```

Obtains the key ID of the key that is sub-key of the key `nKey` and is identified by the name `sName`. Returns `NULL` if the key is not found. Root key is identified by `NULL` value.

```
makeKey (nKey in number, sName in varchar2) return number;
```

The same as the previous function, but the key is created if not found, otherwise existing key ID is returned.

 

```
dropKey (nKey in number);
```

Deletes the specified key with all its sub-keys and values.

```
openPath (nKey in number, sPath in varchar2) return number;
```

The same as `openKey()`, but the sub-key is specified by the path starting from `nKey`. If `nKey` is NULL then the path is absolute, i.e. starts from the root. Key names in the path are separated by slash ("/"). For example, the following path is considered valid: 'My_Component/MyKey/MySubkey'.

```
makePath (nKey in number, sPath in varchar2) return number;
```

The same as the previous function, but the path will be created if not exists.

```
getParentKey (nKey in number) return number;
```

Returns the ID of the parent key of the specified key.

```
getKeyName (nKey in number) return varchar2;
```

Return the name of the specified key.

```
setKeyName (nKey in number, sName in varchar2);
```

This function allows renaming the key as long as you know its ID.

```
getPathString (nKey in number, [cDelim in varchar2]) return varchar2;
```

Returns the absolute path of the specified key. Key names are separated by the delimiter character specified in `cDelim` which defaults to '/'. Root key is identified by NULL value.

```
getPathArray  (nKey in number) return NAME_ARRAY;
```

Returns the absolute path of the specified key as an array of names.

```
countLevel (nKey in number) return integer;
```

Returns the level of the specified key, i.e. its distance from the root. The level of the root key is 0.

```
countDepth (nKey in number) return integer;
```

Returns the depth of the specified key, i.e. the maximum distance from the key to its leaves.

```
countChildren (nKey in number) return integer;
```

Returns the number of children (sub-keys) of the specified key.

```
listChildren (nKey in number) return NUMBER_ARRAY;
```

Returns an array of children IDs.

 

```
listChildNames (nKey in number) return NAME_ARRAY;
```

Returns an array of child names.

```
dropChildren (nKey in number);
```

Deletes ALL sub-keys from the specified key. Key items are kept intact.

 

**ITEM MANIPULATION.**

```
countItems (nKey in number) return integer;
```

Returns the number of data items of the specified key.

```
listItems (nKey in number) return ITEM_ARRAY;
```

Returns an array of all the data items of the specified key.

```
listNames (nKey in number) return NAME_ARRAY;
```

Returns an array of names of all the items of the specified key.

```
copyItems (nKeyFrom in number, nKeyTo in number);
```

Copies all the items of one key to another key.

```
dropItems (nKey in number);
```

Deletes all the items from the specified key.

```
dropItem (nKey in number, sName in varchar2);
```

Deletes the items with the name "sName" from the specified key.

**DATA MANIPULATION.**

In the following functions, the data item is uniquely identified by its name and parent key: (nKey, sName). Most of functions also accept default value to be returned if the item in question does not exist. If the item is not found, and the default value is not specified, function returns `NULL`, otherwise it returns default value.

The following functions extract the item value and return it as one of the commonly used Oracle data types:

```
getClob    (nKey in number, sName in varchar2) return CLOB;
getString  (nKey in number, sName in varchar2) return varchar2;
getString  (nKey in number, sName in varchar2, sDef in varchar2) return varchar2;
getNumber  (nKey in number, sName in varchar2) return number;
getNumber  (nKey in number, sName in varchar2, nDef in number) return number;
getDate    (nKey in number, sName in varchar2) return date;
getDate    (nKey in number, sName in varchar2, dDef in date) return date;
getBoolean (nKey in number, sName in varchar2) return boolean;
getBoolean (nKey in number, sName in varchar2, bDef in boolean) return boolean;
getRaw     (nKey in number, sName in varchar2) return RAW;
getStringArray (nKey in number, sName in varchar2, [cSep]) return STRING_ARRAY;
getNumberArray (nKey in number, sName in varchar2, [cSep]) return NUMBER_ARRAY;
```

The last two functions allow obtaining data chunks as arrays. Data chunks are separated by separator character specified in `cSep`. Default separator is comma (",").

 

The following functions allow setting values of items. If the specified item does not exist, it will be created, otherwise its current content is replaced with new content.

```
putClob   (nKey in number, sName in varchar2, clobData in CLOB);
putString (nKey in number, sName in varchar2, sData in varchar2);
putNumber (nKey in number, sName in varchar2, nData in number);
putDate   (nKey in number, sName in varchar2, dData in date);
putBoolean(nKey in number, sName in varchar2, bData in boolean);
putRaw    (nKey in number, sName in varchar2, rData in RAW);
putStringArray(nKey in number, sName in varchar2, asData in STRING_ARRAY, [cSep] );
putNumberArray(nKey in number, sName in varchar2, anData in NUMBER_ARRAY, [cSep] );
```

The following functions are overloaded versions of data manipulation functions described above which use key names (paths) instead of IDs for item identification:

```
getClob    (sPath in varchar2, sName in varchar2) return CLOB;
getString  (sPath in varchar2, sName in varchar2) return varchar2;
getString  (sPath in varchar2, sName in varchar2, sDef in varchar2) return varchar2;
getNumber  (sPath in varchar2, sName in varchar2) return number;
getNumber  (sPath in varchar2, sName in varchar2, nDef in number) return number;
getDate    (sPath in varchar2, sName in varchar2) return date;
getDate    (sPath in varchar2, sName in varchar2, dDef in date) return date;
getBoolean (sPath in varchar2, sName in varchar2) return boolean;
getBoolean (sPath in varchar2, sName in varchar2, bDef in boolean) return boolean;
getRaw     (sPath in varchar2, sName in varchar2) return RAW;
getStringArray (sPath in varchar2, sName in varchar2, [cSep]) return STRING_ARRAY;
getNumberArray (sPath in varchar2, sName in varchar2, [cSep]) return NUMBER_ARRAY;

putClob    (sPath in varchar2, sName in varchar2, clobData in clob);
putString  (sPath in varchar2, sName in varchar2, sData in varchar2);
putNumber  (sPath in varchar2, sName in varchar2, nData in number);
putDate    (sPath in varchar2, sName in varchar2, dData in date);
putBoolean (sPath in varchar2, sName in varchar2, bData in boolean);
putBoolean (sPath in varchar2, sName in varchar2, rData in RAW);
putStringArray(sPath in varchar2, sName in varchar2, asData in STRING_ARRAY, [cSep] );
putNumberArray(sPath in varchar2, sName in varchar2, anData in NUMBER_ARRAY, [cSep] );
```

 

## REGEDIT TOOL.

**Regedit** is a GUI tool, which serves as visual browser/editor for DPSP Registry. **Regedit** is implemented as Java-archive and requires that your computer be Java2-enabled (that is, J2RE should be installed on target computer). You can run this tool in two ways:

1. As an application:

```
> java -jar regedit.jar
```

provided that CLASSPATH environment variable is set correctly (it should include Java2 runtime packages and swing packages).

2. As an applet in Web-browser. To enable **Regedit** as applet, copy `regedit.jar` and `Regedit.html` files to the root of document directory of your Web-server (in Apache Web-server it is usually "`.../Apache/htdocs`" directory) or any subdirectory of the root document directory as you see fit. Be sure to protect the files and require proper authentication before allowing access to them. Then you can invoke the tool from your browser by specifying an URL similar to the following:

```
http://your-server/[optional/subdirectory/]Regedit.html
```

Please note that when ran as applet, the **Regedit** tool can connect only to the server specified in URL. It will not allow you to connect to any other server even though they may have DPSP Registry installed.

**Figure 1. Regedit Tool running as applet.**