# DPSP™ – Dynamic PL/SQL™ Server Pages

# Name-Tree Service (NTS) Module

# UI User's Guide

**Version 2.3.0**



Document ID: 008-200002-051
Last Revision: January 10, 2003

**COPYRIGHT INFORMATION AND ACKNOWLEDGEMENTS**

# Table of Contents



# INTRODUCTION

## WHAT IS DPSP NAME-TREE SERVICE?

Dynamic PSP Name-Tree Service (NTS) is a Dynamic PSP add-on module, which provides developers and content managers with familiar and comfortable file system-like resource naming subsystem for managing Dynamic PSP units and static content, like images and documents, over the web. When installed on Dynamic PSP system, NTS provides name resolution layer for DPSP Kernel and adds web interfaces for managing the tree structure and resources of the site. NTS is transparent to the end users and developers accessing the DPSP site. In addition, NTS provides a framework API that allows developers to extend basic capabilities of the system by adding new resource types and mechanisms for accessing and processing them.

NTS is also a vital component for JOPA Gateway servlet's WebDAV support. JOPA servlet relies on NTS for building the virtual file system and managing resources (Dynamic PSP units and files) of a DPSP-powered site.

## WHAT IS IN THIS DOCUMENT?

This document is your primary source of information about NTS User Interface (UI). It describes NTS architecture and principles, documents NTS User Interface and how to use it for managing your Dynamic PSP sites static content. The document is comprised of several chapters:

### Chapter I. DPSP Name-Tree Service Architecture.

This chapter describes how NTS works.

### Chapter II. Managing DPSP Site Units and Resources with NTS.

This chapter describes NTS module installation, configuration, NTS web interface and how it may be used to manage DPSP site units and resources.

# I CHAPTER I. DPSP NAME-TREE SERVICE ARCHITECTURE.

This chapter describes DPSP Name-Tree Service architecture and principles put behind it.

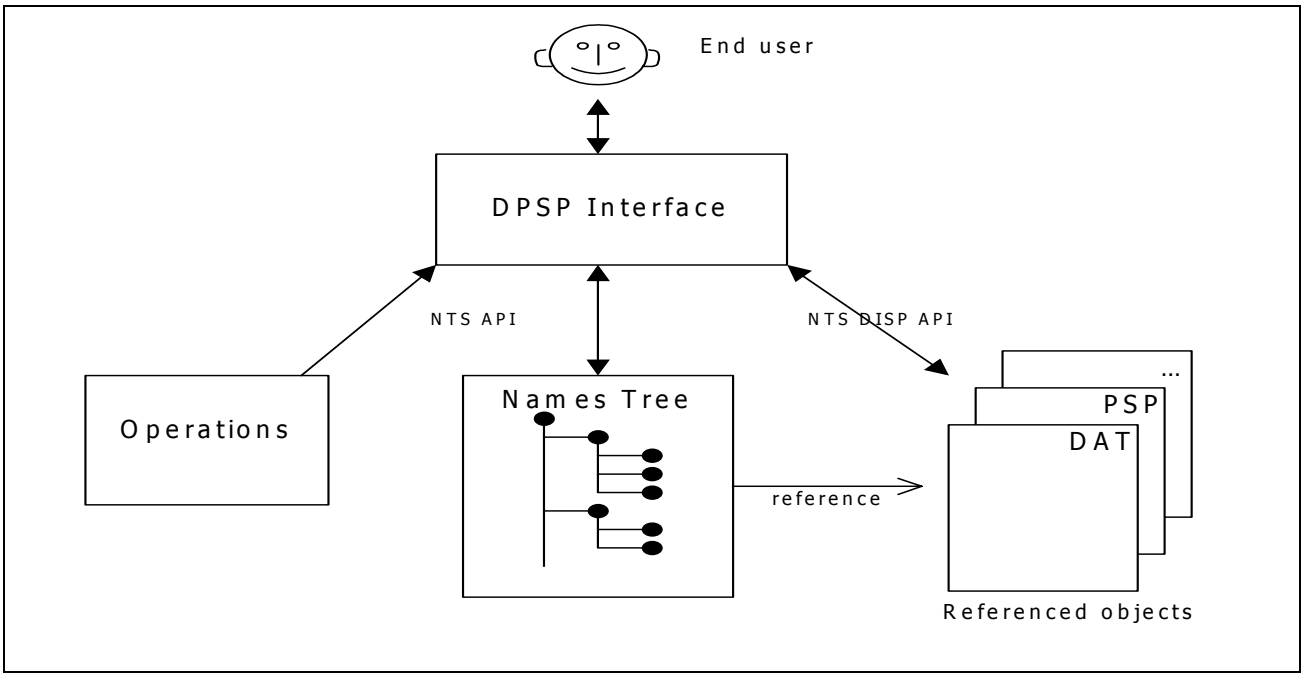
## THE NEED FOR NTS.

Most people are very comfortable with tree-like structures found in traditional file systems. Such structure starts with root folder, in which various numbers of other folders and files reside. Each folder may in turn contain other folders and files as well. Users can group files and folders naturally, by storing scripts and other files for some project in a common folder, for example, and easily find files and identify their purpose by looking at their container folder name. Default Dynamic PSP unit storage, on the contrary, is flat and provides only one level of hierarchy. All units for the site are stored on the same level. Also, there is no default storage for static site resources, like files or images, and DPSP developers have to implement their own storage and access mechanisms for static site resources – they may opt for external storage in web server file system, or they may implement the storage as BLOBs in a table within Oracle database using tools and techniques provided by PL/SQL Web Gateway (mod_plsql) and JOPA Gateway servlet. Either way, this requires additional work from developers, and usually requires additional ways into the web server or database server beyond HTTP protocol.

NTS was designed to address both issues at once. NTS is tightly coupled with Dynamic PSP Kernel and provides tree-like hierarchical naming system for both DPSP units and other resources, like files and images, support for customizable mechanisms for manipulating names and their associated resources, and a comprehensive extensible web interface for manipulating actual resources managed by NTS over HTTP using a web browser. In essence, NTS emulates traditional file system storage for DPSP sites. The main difference between real file systems and NTS is the fact that there is no actual file system with NTS – all resources are stored in Oracle tables, and NTS emulates file system hierarchy, manages 'folders', 'files' and 'links' within the tree, various resource types, and actions that may be performed on these resources. NTS also provides DPSP Kernel with advanced name-to-resource resolution mechanism that translates resource name to actual resource location in an Oracle table and identifies the code responsible for performing requested action on the resource; and an API that allows registering new resource types, and registering and programming various actions that may be performed on these resource types.

## NTS ARCHITECTURE AT A GLANCE.

The following diagram depicts basic architecture behind the NTS:



Key components of the system are:

- **Names Tree** – A hierarchical storage of NTS object names and reference tokens. Names Tree does not store referenced objects themselves, nor does it care how object reference tokens it stores are related to actual data or which property of that data they represent. NTS reference token may be a primary key into some table, or anything else fitting into `NUMBER`. Main purpose of NTS is to retrieve this reference token given the name in NTS hierarchy, and call corresponding operation code with this token as parameter.
- **NTS Dispositions API** – An API allowing to register new object types (dispositions) and corresponding operations that may be performed on these types. Object type and its corresponding operation set are key properties of data objects referenced by the Names Tree. When an object in a tree is requested, NTS determines its type and requested operation, locates code (Dynamic PSP unit) that performs the operation, and then executes the code passing object's reference token to it.
- **NTS API** – An API allowing custom PL/SQL code to interact with NTS and provide services through it. It is used in particular in NTS disposition operation implementation code.

When end user issues a request for an NTS-managed resource through Dynamic PSP, NTS system performs the following tasks:

1. Traverses through the Names Tree to locate the resource record,
2. Retrieves resource type (disposition), reference token and identifies the requested operation. If no specific operation was specified, default operation is implied.
3. Creates the URL to the unit registered as processor for the operation, and passes reference token to it along with other relevant parameters. Client is then redirected to this URL. The unit is responsible for performing the operation on the object identified by the reference token and communicating the result back to the client.

Operations on NTS nodes are divided into two groups: class operations and node operations. Node operations are performed on existing NTS nodes, while class operations are generic (usually they implement node constructors.) For example, create operation is class operation because it does not have an existing node to work with, but rather it creates a new node. Delete or display operations on the contrary are working with single existing node, thus they can be classified as node operations. For each resource type (disposition) there can be any number of class and node operations. One of node operations may be designated as default. For example, if you use NTS to manage documents, default operation for the document disposition may be 'display', and it will retrieve the document from server-side storage and present it to caller in some way.

#### DISPOSITIONS.

Dispositions in NTS are registered resource types, for which a set of operations is defined. Each disposition is identified by 3-letter abbreviation. The only disposition internal to NTS itself is `DIR`, and it represents a branch, or folder, in the Names Tree. All other dispositions are external to NTS and are registered through NTS Dispositions API. Two default dispositions are shipped with NTS – `DAT` and `PSP`. `DAT` disposition represents binary data (file) and `PSP` represents Dynamic PSP unit. Dynamic PSP developers may implement and register their own dispositions at will using Dynamic PSP for implementing operations and NTS Dispositions API for registering dispositions and their operations with NTS.

#### OPERATIONS.

Disposition operations are implemented as Dynamic PSP units and are registered through NTS Dispositions API. Each operation may work with existing tree node or implement a generic class operation, like creation of new node. NTS name resolver registered with Dynamic PSP Kernel automatically identifies which unit should be called for each requested operation and constructs the call URL for the corresponding implementation unit. Disposition operation implementation and registration will be discussed in detail in Chapter III.

# II CHAPTER II. MANAGING DPSP SITE UNITS AND RESOURCES WITH NTS.

This chapter describes the NTS module installation, configuration, web interface and how it can be used to manage a Dynamic PSP site units and resources, like files and images.

## INSTALLING NTS.

NTS module is installed into the Dynamic PSP Kernel schema with SQL*Plus script provided with distribution archive. The script is invoked as follows:

```
> sqlplus dpsp2owner/password[@TNSAlias] @create_nts_std
```

where `dpsp2owner` is Dynamic PSP Kernel owner user, `password` is that user's password and `TNSAlias` is optional Oracle instance TNS alias. `create_nts_std.sql` script is invoked with the above command. This script calls several other scripts, which actually create and populate NTS database objects and update Dynamic PSP configuration in DPSP Registry.

The installation script installs and enables NTS only for Dynamic PSP Kernel schema. To be able to use NTS from Dynamic PSP project schemas, each schema should be enabled for NTS. `enable_nts_project.sql` provided with the NTS should be used for enabling each particular Dynamic PSP project schema for NTS:

```
> sqlplus projectowner/password[@TNSAlias] @enable_nts_project
```

where `projectowner` is Dynamic PSP project schema owner, `password` is that user's password and `TNSAlias` is optional Oracle instance TNS alias. The script will prompt for the name of Dynamic PSP Kernel owner user (schema) and will create necessary synonyms in project schema to enable the project for NTS.

### CONFIGURING MOD_PLSQL DATABASE ACCESS DESCRIPTORS (DADS) FOR NTS.

Each DAD describing a database connection for mod_plsql should be additionally configured for NTS. In particular, *document table*, *document access path* and *document access procedure* should be specified to allow NTS to handle document access requests. To initially configure the DAD, follow instructions provided in Dynamic PSP 2 User's Guide and Reference. Once the DAD is configured according to those instructions, the following changes should be made to it to allow NTS to handle document access requests:

**Document Access Path** prefix should be specified. This can be any prefix of your choice.

**Document Table** should be set to **`<DPSPKernelSchema>.NN$T_DOWNLOAD`**, where `<DPSPKernelSchema>` is Dynamic PSP Kernel schema name.

**Document Access Procedure** should be set to **`NN$NTS_OWA_DOWNLOAD`**. This procedure will handle mod_plsql document download requests. You can also specify **`<DPSPKernelSchema>.NN$NTS_OWA_DOWNLOAD`**, where `<DPSPKernelSchema>` is Dynamic PSP Kernel schema name, as value for this parameter.

Once these modifications are done, you can access documents managed by the NTS with URLs similar to the following:

```
/pls/DAD/document_access_prefix/<NTS path>
```

where `pls` is mod_plsql location configured in Oracle HTTP Server's configuration (usually plsql.conf file), `DAD` is DAD name, `document_access_prefix` is document access prefix you specified for the DAD and `<NTS path>` is full NTS path to the node you are requesting.

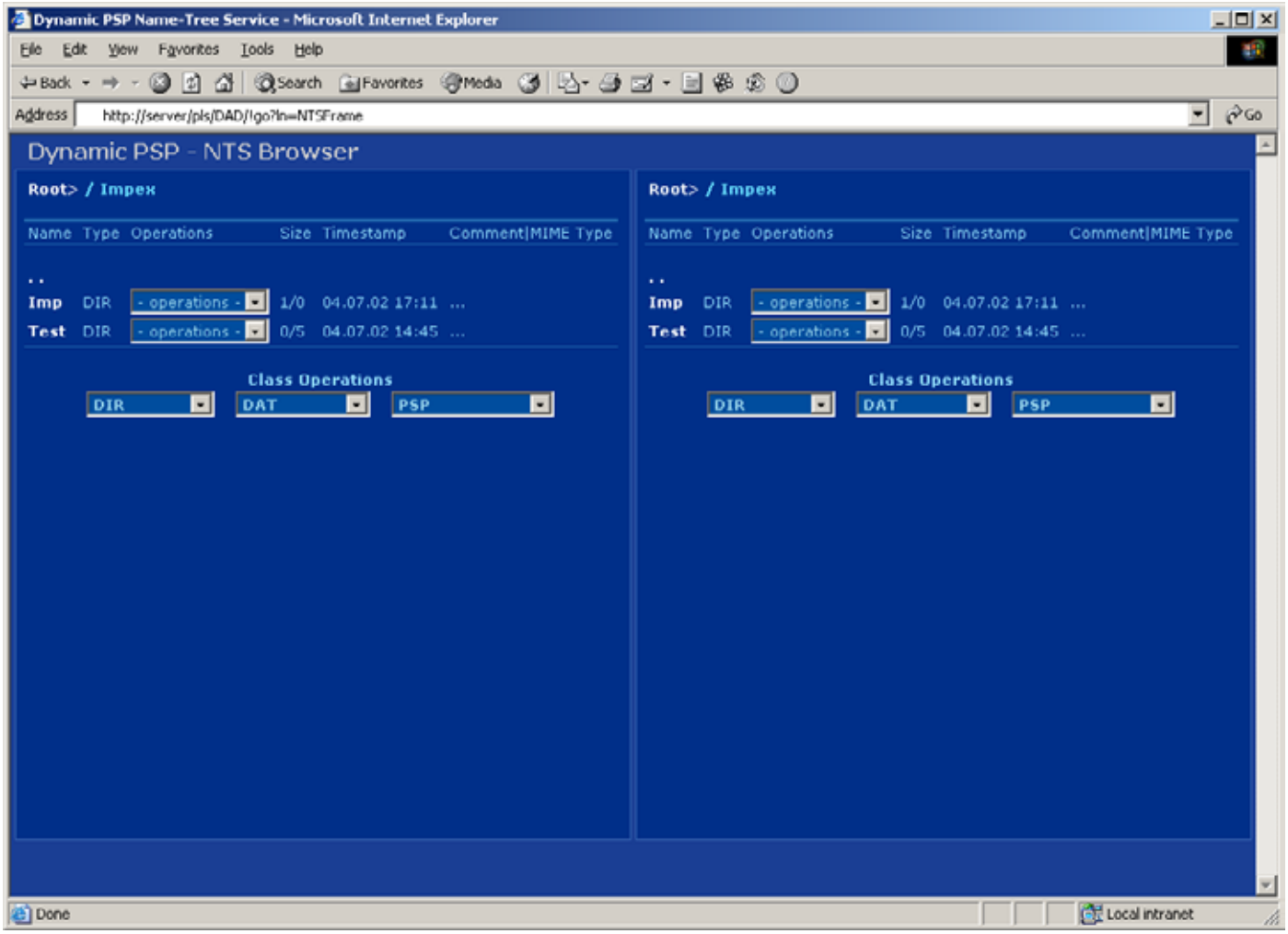
## ACCESSING AND NAVIGATING THE NTS BROWSER.

NTS Browser is web-based NTS tree browser written in Dynamic PSP. As soon as you add NTS module to the Dynamic PSP installation and configure the DAD as per installation instructions, NTS Browser is available at the following URL of your Dynamic PSP site:

`http://server/pls/DAD/!go?ln=NTSFrame`

(HTML 4.0 compatible browser with scripting enabled required. Microsoft Internet Explorer 5.5 or later is recommended, Netscape 6.2x or later and Mozilla 1.0.2 or later are supported.)

You will be prompted for login credentials (they are the same as for the rest of Dynamic PSP Development Interface and are registered and edited through DPSP Unit Commander). Then, a screen similar to the following will appear:




Two similar panels are displayed on this screen, and each panel displays a `DIR` node in the NTS tree. The node display may list links to sub-nodes of currently displayed node. Below the node display there are dropdown boxes for various generic operations (Class Operations) that do not operate on specific nodes – one dropdown for each disposition. On the sample screenshot above, three class operation dropdowns are displayed: for `DIR`, `DAT` and `PSP` dispositions. Each sub-node in the list also has an Operations dropdown associated with it. One of these operations is default and is performed when you click on the sub-node link (name), other non-default operations are only available through the associated Operations dropdown box.

### DIR DISPOSITION OPERATIONS.

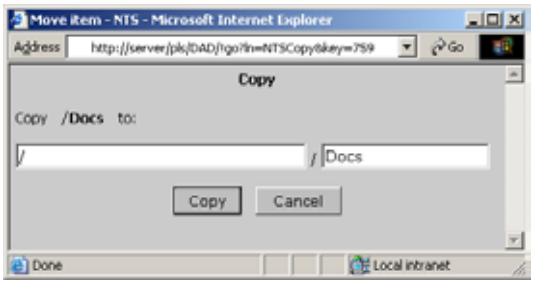
For `DIR` disposition there is one class operation and 7 node operations defined: **New Folder** class operation, **Open**, **Copy**, **Move**, **Delete**, **Rename**, **Comment** and **Secure** node operations.

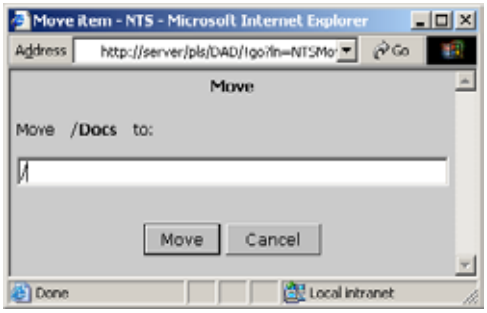
**New Folder** class operation creates new `DIR` node under current node:


**Open** operation opens a `DIR` node (this is default operation).

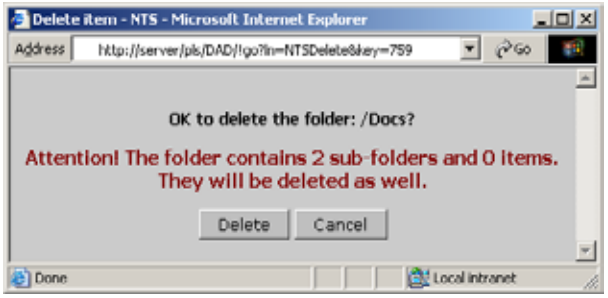
**Copy** creates a copy of the node under specified `DIR` node (by default, under the node currently displayed in the opposite NTS Browser panel):




**Move** moves the node under specified `DIR` node (by default, under the node currently displayed in the opposite NTS Browser panel):

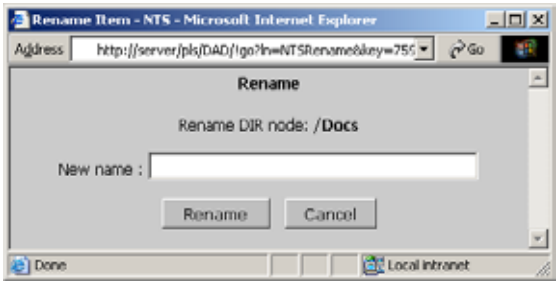



**Delete** deletes a node and all its children. If the node has any children, their count is displayed in the delete dialog. The user is always given an option to cancel delete:

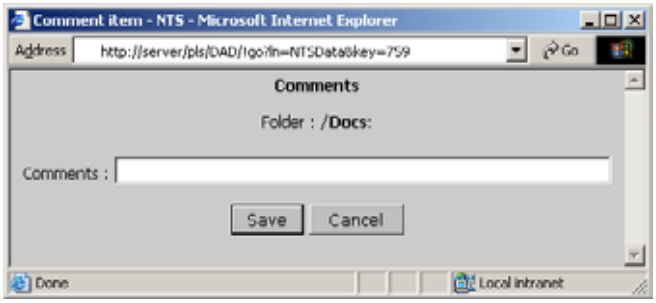


**Rename** renames a node:




**Comment** adds comment to the node.

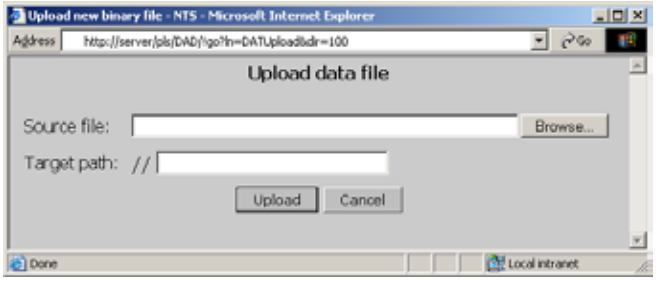



**Secure** allows assigning minimum access level a DPSP-authenticated user is required to have to access the node. All nodes with assigned access level higher than current user's access level will be inaccessible (invisible) to the user:


#### DAT DISPOSITION OPERATIONS.

For DAT disposition there is one class operation and 8 node operations defined: **Upload new** class operation, **Download**, **Copy**, **Move**, **Delete**, **Rename**, **Reload**, **Attributes** and **Secure** node operations.

**Upload new** class operation provides a way to upload new binary resource (file) and create corresponding NTS node referencing the resource under currently selected node. The file itself is uploaded into special table for binary resources:



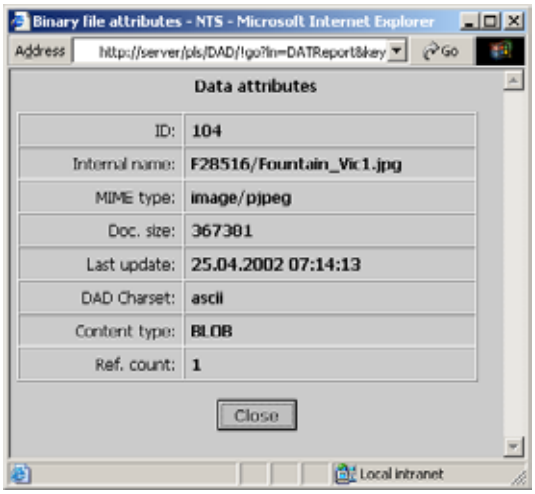

 

**Download** operation allows downloading the resource to the client PC (this is default operation).

**Copy**, **Move**, **Delete**, **Rename** and **Secure** operations act the same as for DIR disposition.

**Reload** operation allows replacing current resource with new resource while keeping it at the same position in the tree and maintaining all links to it. Reload window is the same as Upload new window, but it automatically receives the name of the node to be replaced.

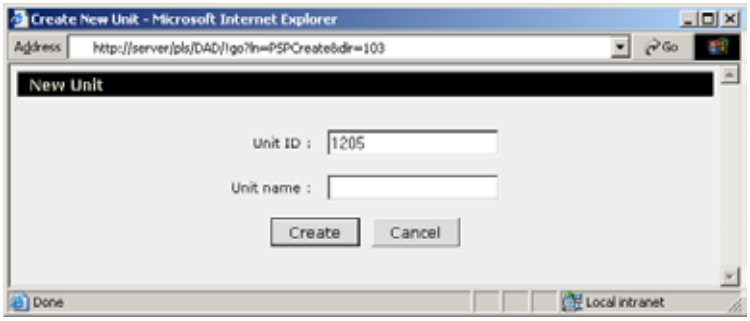
**Attributes** operation displays current node attributes:



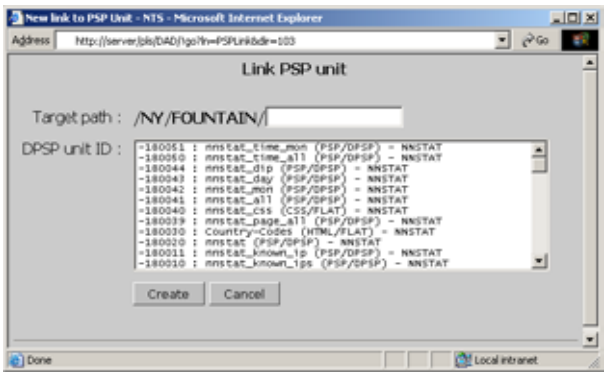

### PSP DISPOSITION OPERATIONS.

For PSP disposition there are 7 class operations and 14 node operations defined:

**Create unit** class operation creates new unit and adds a link to it into the NTS tree:



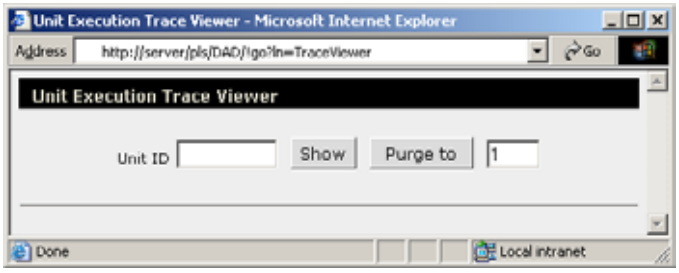

**New link** class operation creates new link to existing unit as NTS node. The same unit may have several associated links in the NTS tree; all these links are equivalent.



**Make** class operation attempts to compile all invalid units in the system.

**Tracer** class operation opens unit execution trace window where you can choose the unit for which to display trace information:




**Users** class operation opens Dynamic PSP user management console.

**Change passw.** class operation allows to change current Dynamic PSP user's password.

**Registry** class operation opens Dynamic PSP Registry editor (available only for DPSP2 Kernel schema).

Node operations:

**Execute** operation attempts to execute the referenced unit (this is default operation).

**Edit** opens the unit in DPSP Unit Editor.

**Attributes** opens Unit Attributes editor.

**Preview** opens Unit Code Preview window.

**Compile** attempts to compile the unit.

**Re-link** allows to link current node to a different existing unit.

**Trace** open Unit Tracer window.

**Errors** opens Unit Errors Log window.

**Copy**, **Move**, **Rename**, **Delete**, **Comment** and **Secure** act the same as for other dispositions.

As you may have noted, most Dynamic PSP Unit Commander functions are available through PSP disposition's class and node operations.