A Web-based System for Collaboratively Developed Ontologies

by

Clare Lee

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

September 1999

©1999 Clare Lee. All rights reserved.

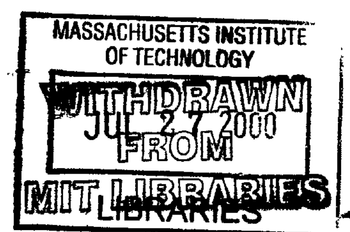
The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author ....
Department of Electrical Engineering and Computer Science
August 13, 1999

Certified by ....
Philip G. Greenspun
Thesis Supervisor

Accepted by ....(
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

ENG



A Web-based System for Collaboratively Developed Ontologies

by

Clare Lee

Submitted to the
Department of Electrical Engineering and Computer Science

August 13, 1999

In Partial Fulfillment of the Requirements for the Degrees of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This work describes the design, implementation, and test of a programming tool and user interface. A classified ads system is rewritten to experiment with the application programming interface for the tool and the conclusions derived from this experience are summarized.

This tool makes it easier to engineer a broad class of web services – any services that uses an ontology to organize content. This paper describes a system to manage categorizations for online content.

Thesis Supervisor: Philip G. Greenspun
Title: Research Scientist, Laboratory for Computer Science

Acknowledgments

Naturally, I must begin by thanking Dr. Philip Greenspun for his help in making this an exciting thesis and for his editorial contributions to this paper. His perspective is always enlightening. Thanks also to Alex for reminding me of the future. I also thank Prof. Harold Abelson for his advice regarding the direction of my research.

I would like to thank Tracy Adams for her guidance during the design and implementation of the ontology system. Her suggestions were timely and helpful, not to mention motivating. Thanks also go to the people at ArsDigita, LLC.

Thank you to all my friends and family. I thank Richard Li and Janet Liu in particular for their friendship that made the time palatable.

I dedicate this thesis to my parents. My grateful thanks to them for packing my room for me. I would not have been able to leave MIT without their help. I thank them for their love and long-suffering support.

Contents

1 Introduction **7**

1.1 The Problem 7

1.2 The Solution 8

2 Design **9**

2.1 Overview 9

2.2 A Domain 10

2.3 Categories 11

2.4 Items 11

2.5 Permissions 11

2.6 API 12

3 Implementation **13**

3.1 System Overview 13

3.2 Data Model 14

3.2.1 Domains 14

3.2.2 Categories 15

3.2.3 PL/SQL procedures 18

3.3 Tcl procedures 19

3.4 Administration Pages 19

3.5 User Pages 20

3.6 Rewriting the Classified Ads Module 20

4 Results **23**

4.1 Ontology Pages 23

4.2 Ontology API 24
4.3 Overall Effectiveness 24
4.4 Discussion Section 25
4.4.1 Multiple Parents 25

5 Conclusion **26**
5.1 Future work 26
5.1.1 Minimum depth for adding an item 26
5.1.2 Multiple hierarchies 26
5.1.3 Pay attention to user input 27
5.1.4 Data transferral 27
5.1.5 Performance improvements 27
5.1.6 More display options 27
5.1.7 Administration 28
5.2 Concluding remarks 28

A SQL Data Model **29**

B Online Ontology Documentation **33**
B.1 The Big Picture 33
B.2 The Medium-sized Picture 33
B.3 Permissions 36
B.4 The Steps 37
B.4.1 Applying this Package to an Existing Module 37
B.4.2 Categorizing New Content 39
B.5 Application Programming Interface 40

C Tcl Procedures and Application Programming Interface **43**

References **89**

List of Figures

3-1 System Architecture 13
3-2 ont_items_1 (created when domain_id = 1) 18

Chapter 1

Introduction

This work describes an application programming interface (API) for a general ontology system: Ontology Tool. This tool builds and manages hierarchies of categories.

Most web services today require ontologies. They range from the simple flat, eleven-category ontology like Salon Entertainment's [11], to eBay's [5] extensive four-level hierarchy. Categories are fundamental to eBay's business of online trading. Buyers and sellers can only find themselves through searching or browsing categories. If buyer and seller use the wrong title string or the wrong category, they will never meet, and eBay fails.

In the ACS toolkit alone, there are six separate systems, each with an category hierarchy: bulletin boards, neighbor-to-neighbor, calendar, general classified ads, static pages, and the ad server. All of these modules can benefit from the ontology tool.

1.1 The Problem

For each service, the categorization code must be designed from scratch. For example, when building the Virtual Compassion Corps [4], a web site allowing users to post listings requesting and offering help. The capability of this site is similar to the classified ads systems in the ArsDigita Community System (ACS) [13]. Why didn't we use this well-tested system? We needed a multi-level hierarchy to organize donations or requests. It was too difficult to separate the ontology from the rest of the system. Assumptions about a two-level hierarchy were embedded in dozens of classified ads. Programming resources should be applied towards the unique aspects of each application.

1.2 The Solution

I developed an ontology tool and application programming interface (API) for this purpose. Programmers can use this API to build systems that categorize their content and let administrators manipulate the categorizations. Web services built with this system can support multiple publishers. Users are allowed to add content, remove content, and move content into different categories. Additional features include dictating the depth of the ontology. If users would like to categorize already-existing content, they can use the ontology tool's Web interface directly to organize it, bypassing the API.

I rewrote the existing classified ads module in the ACS to evaluate the effectiveness of this architecture for the purposes of categorization. The rewritten system has the same external interface and equivalent capabilities. The following chapters explain the design of the ontology system, its implementation, and results and analysis stemming from the implementation and usage.

Chapter 2

Design

The goal is to develop a generic ontology system that faciliates software development and results in a higher quality, more consistent end-user experience. We want to make the process of programming a service shorter and maintaining the service easy as the needs of the service change. We plan to derive these benefits from the principles of code modularity and abstraction.

2.1 Overview

The design principles focused on saving the programmer time and making the resulting service easy to maintain. To save time, the API should do everything that the programmer would want to do. It should also be simple and easy to understand - the more abstraction at this point the better. The programmer shouldn't have to know how this is done.

To make the tool easy to maintain requires a good design and simple procedures to make modifications and enhancements. If it is difficult to understand the code, a programmer would probably rather write their own code rather than deal with someone else's.

The ontology system begins with a domain. The domain specifies the subset of information that the hierarchy is organizing. One domain might be a set of classified ads for photography-related listings. Another domain might be a forum about taking care of tropical fish. The domain defines a community of users.

The domain also contains a category hierarchy. Categories contain items defined by the end-user for that domain. The category hierarchy should be flexible enough to allow manipulation and movement of categories. Categories can be edited, moved, and deleted.

The depth of the hierarchy can be fixed at a particular level or left unlimited. If a hierarchy has three levels at which items can be posted, categories can be added at the top level and one level below.

Categories contain content; this can be anything that can be categorized online. Let us call this content "items." The publisher creates or supports this content.

2.2 A Domain

A domain represents a particular topic or focus; the rallying point for a collection of end-users. Without users there is no service, so there must be a reason for them to come back to the service. For example, a site could categorize both musical items and tropical fish information. Both of these topics may not belong in the same ontology, but can be supported by the same service.

Within the domain there are further categorizations appropriate for items under that topic. Many publisher choices belong at the domain-level of the ontology. A publisher specifies the title of the ontology, such as "Janet's Movie Reviews." Janet can limit the depth of the hierarchy to two levels deep, if she only wants one level of categories. Janet doesn't plan on writing a vast number of reviews, and only trusts reviews by a few other people, so a few top-level categories are all she needs. Richard wants to organize a collection of links to fun things to do in Japan. He doesn't know how much this collection of links will grow, as he is soliciting suggestions from all his friends. So he leaves the hierarchy unlimited.

With each domain comes one person or more who will administer it: creating and altering the ontology, managing end-users, promoting other users to help administer the domain. So we have administrators who perform this role in the system while ordinary end-users have limited permissions.

A domain dictates the form of content it supports. It specifies the interface between the ontology system and the content defined by the end-users. How can the domain display the content it organizes? By keeping a few key pieces of information to show how this content should be displayed. The domain will need to know what items are in what categories; an identifier for the items. A listing of items will be displayed, so we need to know where to look for a title or short description of the item. Finally, the domain needs to know where

to direct the end-user to see the full details regarding a particular item, a URL. This URL is written by the programmer specifically for the items being categorized.

The domain allows the programmer to specify a service for the domains being created. Janet and Richard may both want to use the classified ads system, but Janet is only interested in ads for movie premiere tickets, while Richard wants to handle ads for youth hostels in Japan. These two topics are quite distinct from each other, and a single ontology detailing both would be essentially 2 hierarchies slapped together. Thus there is a need for multiple domains to use the same service. As a result the ontology system can support multiple domains for multiple services.

2.3 Categories

Each category is part of a particular domain. A category can be something like "Documentaries" in Janet's Movie Reviews domain, or "Youth Hostels on the Chuo Line" in Richard's domain. A category's location is determined by the identity of its parent category. To simplify navigation, each child can only have one parent.

2.4 Items

The contents of the categories are designated ıitems. An item can be anything from a paragraph of text to a classified ad. An item is a programmer-defined unit of web-based content. The ontology system knows nothing about items other than the information specified in the domain: this is a substantial source of power and generality.

2.5 Permissions

There are two roles for people connected to the web interface of this ontology system: administrator and end-user. An administrator can create domains and construct and modify the ontology, in addition to the user privileges. The end-user can add and modify items, and browse the ontology.

2.6 API

The API must contain procedures to do the following:

- display the entire category hierarchy tree
- show the contents of a single category
- add, move, and delete categories
- show items in a category
- add, move, and delete items

Chapter 3

Implementation

This chapter describes the implementation of a basic ontology system using the design previously mentioned in Chapter 2. It will also discuss the specific data model chosen for the system and the reasoning behind it.

3.1 System Overview

To understand the ontology system, the supporting architecture must be explained.

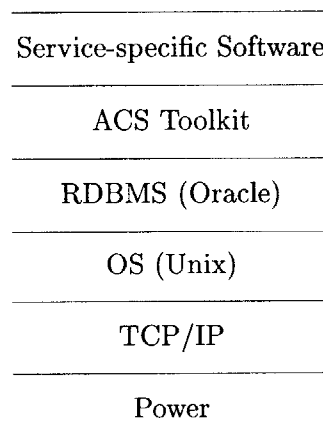


Figure 3-1: System Architecture

The tool is built on top of the ArsDigita Community System (ACS) [13], a toolkit for building online collaborative communities. The ACS is open-source and freely available for download. The toolkit runs on top of the Oracle Relational Database Management System

(RDBMS) and AOLserver, communicating through the AOLserver Tcl API.

The user groups module and the permissions module in the ACS both provide features that significantly simplified the work for the ontology package. The user groups module aggregates users for common user and administration functions. The ontology system takes advantage of this facility in managing user authorization issues. End-users are grouped with their associated roles. The permissions package builds upon the user groups module, creating user groups of type "administration" to grant administration privileges to users. Each domain has a group of this type. This enables us to ask "is user X authorized to perform action Y?"

3.2 Data Model

The Oracle SQL tables and the transactions they support are explained below.

3.2.1 Domains

The domain is represented with the following table in SQL.

```
create table ont_domains (
        domain_id            integer primary key,
        domain_pretty_name   varchar(100) not null,
        -- the full name of the domain, properly formatted, etc.
        hierarchy_depth_limitinteger,
        domain_key           varchar(100),
        -- link to an external table for augmenting information
        service              varchar(20),
        -- directory name for the service using this domain
        active_p             char(1) default 't' check (active_p in ('t', 'f')),
        creation_date        date,
        on_which_table       varchar(50),
        -- which table it is going to link to
        id_column_name       varchar(50),
        -- what is the column name of the id in this table
        pretty_name_column   varchar(50),
```

```
-- what is the pretty column name in this table
item_base_url       varchar(100),
-- the base url of the an item in the module
last_modified       date not null,
last_modifying_user not null references users,
modified_ip_address varchar(50) not null
);
```

The column on_which_table stores the name of the table holding the items. This table must have a primary key column so that individual items can be selected easily. The name of this column is stored in id_column_name. An item listing in a category will look uninteresting and indistinguishable if only a primary key is displayed; in particular if the primary key happens to be a numeric value. For this reason we store the name of a title column in pretty_name_column to use as a heading for the item.

A URL is stored for directing an end-user to the actual content display, taking the end-user into the publisher's system and away from the ontology display. This URL is stored in the column item_base_url. If the item_base_url is incorrect, the link provided to end-users will be broken. The item_base_url is passed the primary key for the desired item. Thus the page stored in item_base_url should not require any additional information passed in.

For each domain, an administration group is automatically created that can be managed from the ACS user group administration pages.

3.2.2 Categories

Categories are represented with the following two tables.

```
create table ont_categories (
       cat_id                 integer not null primary key,
       cat_pretty_name        varchar(1000) not null,
       -- the full name of the category, properly formatted, etc.
       extra_info             varchar(4000),
       domain_id              references ont_domains,
```

```
    -- identifying domain
    active_p              char(1) default 't' check (active_p in ('t', 'f')),
    creation_date         date,
    last_modified         date not null,
    last_modifying_user   not null references users,
    modified_ip_address   varchar(50) not null
    );
```

ont_categories contains the basic attributes of the categories.

The categories are connected in a tree structure, represented in a separate table. They are linked to each other in parent-child relationships which dictate the shape of the tree. Storing this hierarchy in a separate table frees the relationship between categories from any inherent structure. Categories can be in multiple parent-child pairs, depending on the implementation of the category code. The only restriction is that a category cannot be a parent or child of the same category twice.

```
create table ont_hierarchy (
    child_id              not null references ont_categories(cat_id),
    -- id number identifying child category
    parent_id             not null references ont_categories(cat_id),
    -- parent of the category specified by child (one level up in
    -- the hierarchy)
    last_modified         date not null,
    last_modifying_user   not null references users,
    modified_ip_address   varchar(50) not null,
    primary key(child_id, parent_id)
    );
```

All categories associated with a specific domain that do not show up as children in this hierarchy table are top level categories. These categories are the first ones the end-user will see in the hierarchy. Specifically, top-level categories are recognized by the lack of the existence of a row in the hierarchy table. So a category is in the top of the hierarchy for a specific domain if it is a row in ont_categories but does not exist in ont_hierarchy.

If a programmer decides that multiple parents for a category are vital to the service he or she wishes to provide, the current data model can support this with some changes in the Tcl procedures.

Category Hierarchy Representation

A key question in the design of the system was whether to use a table storing parent and child relationships or use a sort key in the `ont_categories` table to store hierarchy information. As explained in the bulletin board example in [9], a sort key indicates numerically the position of the corresponding category within the entire hierarchy. The top level categories would all use the category id as their sort key. When a sub-category is added to one of the parents, its sort key is calculated as the parent's sort key, with a period and 2 digits appended to it. For example, if a top level category has the sort key "000001", the first sub-category added to it will have the sort key "000001.00". An additional sub-category will be given "000001.01", "000001.02". If the ordering changes with the addition of the new sub-category then the sort keys among the sub-categories will have to be swapped. When ordering by the sort key, the categories are ordered by parentage in the hierarchy.

Issues with the sort key include the difficulty in moving a category within the hierarchy. Together, all the sort keys present a fixed encoding of the hierarchy. If a category is moved, all the sort keys for the moved category and any sub-categories will have to be updated to reflect the new hierarchy. Also, with the described sort key implementation, multiple parents do not make sense with this model. This model can only support single parent category trees.

In the context of this ontology system we need to order all the categories, and indicate which categories are sub-categories of others. With the parent-child hierarchy, the categories are looked up using a SQL "connect by" clause. The usage of "connect by" in Oracle is limited so that you cannot order the leaves, which is simple with the sort key. Additionally, with this clause we cannot join tables together and select information from both tables. To bypass this restriction, we use PL/SQL functions in their place. These functions are convenient but with overuse comes a performance cost. In this implementation the parent-child hierarchy was effective but the sort key would have worked just as well, depending on the needs of the end-users.

The contents of categories can be moved around as easily as the categories themselves.

```
create table ont_items_1 (
       cat_id               references ont_categories,
       item_id              references ont_domains.on_which_table,
       last_modified        date not null,
       last_modifying_user  not null references users,
       modified_ip_address  varchar(50) not null
       );
```

Figure 3-2: `ont_items_1` (created when `domain_id` = 1)

The information describing the items sits in a programmer-defined table. With a few facts about the item table specified in the domain, we can safely perform all categorization functions. Using the column name of the primary key for the item information, we generate a table referencing both the unique category key and the unique key for the item. There is one table per domain, whose name is generated from the domain identifier. A sample table is shown in Figure 3-2.

An alternative approach is to create a single table to map all items to all categories. It could reference the category information table and have a column to store the item keys. However, in a multiple domain environment this model has difficulties. The single global mapping table would have to maintain domain information for each row, whereas having one mapping table per domain implicitly maintains this information. Also, as the number of items grow, queries on the single table will take longer to return, slowing down the service. Searching the domain-specific table will automatically decrease performance penalties.

Items in the top level of a domain are recognized by the lack of a parent category in the mapping table. An item is in the top level if it is in the `ont_items_$domain_id` table with `cat_id` set to null. Placing the item in the `ont_items_$domain_id` table associates the item with the specific domain.

3.2.3 PL/SQL procedures

The category trees are obtained with the help of the SQL `connect by` clause. However, this clause has certain limitations, in particular that we cannot get information from more than one table while using it, nor can we order the items in the rows returned by a `connect by`. The category hierarchy information and category details are stored in two separate tables. To get around the restrictions of `connect by`, I built procedures in the Oracle-

specific language PL/SQL to get the category details for `cat_pretty_name`, `active_p`, and `domain_id`.

3.3 Tcl procedures

A collection of Tcl procedures form the API for the programmer.

User permissions are checked in the Tcl procedures. The alternative is examining permissions in the Tcl pages, where the programmer must decide who is allowed to do what. Placing the permissions checks at the API level does not give the programmer the same flexibility in choosing end-user roles, but does ensure consistency in determining who has the correct permissions. Rather than hope that the programmer will remember to check the permissions of end-users, the API hides this detail from the programmer. At the API level the user permissions are specifically set to divide end-users into the previously mentioned two groups: the user and the administrator.

Tcl procedures can also be memoized, saving database queries and providing major increases in performance.

Once a domain is created, none of the API procedures allow the depth of the hierarchy or the `item_base_url` to change. Changing the depth of the hierarchy will create inconsistencies in the display of the hierarchy if categories and items already exist below the specified depth. They would be counted as items in visible categories, but never displayed.

3.4 Administration Pages

The administration pages are located in a directory called `/admin/ontology/`. For each domain, an administration user group of module "ontology" and submodule with a value derived from the unique domain key `domain_id` is created. Administration of this user group can then be done using the Tcl pages in the above-mentioned directory.

As defined, there is one administration group per domain. With the two roles in the system, the only distinction is whether the end-user is an administrator or not. If a person is not an administrator, that makes him or her solely an end-user. Beyond defining who can do what, the administration pages are where the category hierarchy can be changed. If necessary, administrators can also edit domain attributes and move and delete items.

Appropriate user groups are set up automatically when a domain is created. They define the roles and corresponding permissions for each domain and each end-user in the domain. Each domain has its own user group with `module=ontology` and `submodule=domain_id`.

Having the checks at the API level has the significant advantage of hiding the details from the programmer. An end-user is any registered user of the ACS running on the virtual server on which the ontology system is running. Administrators are end-users in the administration group for the specified domain.

Initially, the administration strategy was to have two user groups: site administrators and category administrators. Category administrators can manage their category and child categories. For each domain there could be a user group, however for many uses, any random end-user should be able to browse and add content to a site. If permissions are more strict they can be modified in the programmer's system.

3.5 User Pages

The end-user pages are found in the `/ontology/` directory. These pages provide the normal user's view of the ontology. End users can add items, remove them, and browse the hierarchy; actions which are configurable by the programmer for user permissions. These pages demonstrate various API procedures, and some Tcl pages are reusable. In particular, the `category-move` pages display the ontology of categories that can provide a new home for the category to be moved. There is also a set of `item-select` pages for selecting an item to be moved.

3.6 Rewriting the Classified Ads Module

The classified ads system [7] as seen at photo.net shows classified ads. Some of the domains have fixed hierarchies. Other domains let end-users create a new category to hold their ad if they do not like any of the options available. All of the hierarchies have exactly one level of categories. Users can view ads by category or chronologically.

The primary SQL tables for the classified ads system are `ad_domains` and `classified_ads`. The table `ad_domains` specifies all sorts of details about a classified ads domain including default expiration date, auction options, and extra fields the classifieds will contain. The

system uses categories as attributes of an ad. This model is good for showing the same content in several different hierarchy views.

The overall idea in rewriting the module was to replace all the category and domain code and use the ontology system to manage the ontology details, while leaving the existing code to manage the classified ads and related details specific to the service itself.

Most of the calls to the API are straightforward. One conceptual complexity is in the frequent need for the `domain` value, the unique identifier for a domain. `ont_domains` uses `domain_id` as a unique identifier instead. All the API procedures use `domain_id` and all the Tcl code uses `domain`. To logically connect the ontology domain with the administrator-defined domain specifics, the `ont_domains` table has a `domain_key` column. The `domain` value for each `ad_domains` row is stored in the corresponding row for `ont_domains`. This lets the programmer grab the `domain_id` given the `domain` and vice versa, making it simple to switch between identifiers.

The unique identifier `domain_key` in `ont_domains` can be used as a reference to any additional domain information. Since the rewritten service is the new classified ads system, the directory for the new service is "gcnew", for "general classifieds new." Use "gcnew" as the name of the service, and pass it as the value for the `service` input to the `ont_domain_add`.

Use `ont_domain_id_new` to get a new `domain_id` when checking for double clicks on a submission page. Grab a new id on the first page in the sequence of submissions. If the end-user double clicks on a page, the code will try to enter two rows with the same unique identifier. In the even of a failure resulting from this double entry, the code can test whether the given `domain_id` already exists in the `ont_domains` table. If it exists, the row must already have been inserted. This procedure is also available for categories.

Categories in the classified ads system are part of the `classified_ads` table. The table maintains a `primary_category` column and two sub-category columns. With the ontology system these columns can be ignored. Category manipulation is done through the API calls and category and item display are taken care of using the many tree display procedures. These procedures can return HTML or Tcl lists. The Tcl lists are for the programmer to use if the HTML procedures do not provide enough interfaces or functions. `ont_item_list` returns a list of the identifier values for all the items in a specified category. `ont_cat_with_items_tree_html` lists the entire hierarchy in a nested list format. The programmer can provide URL, title pairs to be listed beside each category. For the new

classified ads, this procedure was used to provide edit and move category links beside each category. The URL for the edit page was listed with the title of the URL "Edit." This list was listed with the delete list, and passed to the procedure.

Chapter 4

Results

This chapter evaluates the stand-alone ontology system and the API. The ontology pages are evaluated by looking at the process of creating several domains. The experience of replacing code in the ACS classified ads module with the API points out some of its strengths and weaknesses. Overall the API design worked easily within the framework of the existing classified ads system.

4.1 Ontology Pages

The stand-alone ontology pages demonstrate that the design chosen can categorize generic content. The system has a default table where users can store basic information about an item, such as title, description, and URL. These items are easy to move about the ontology, and categories are simple to change. The interface with tables unknown to the ontology system was tested by creating domains for random existing tables in the Oracle database, such as `users`. There is a single Tcl page to display the public details about a specific user, where users were being categorized. The data model for `ont_domains` expects that the specified page can display information about any item when provided with the value of the unique identifier for the items table from which it gets information. The ontology pages pass the value of the item identifier to the specified page, dictating a portion of the interface between the system and external pages.

4.2 Ontology API

The rewritten classified ads package demonstrates more requirements that the data model imposes upon the programmer through the API. However it also shows that the API takes care of all necessary actions with reasonable effort on the part of the programmer.

As mentioned in 2.6, the programmer has procedures to switch between the ontology system's `domain_id` and the programmer's unique id for additional domain information. These procedures assume that the domains were created so that the unique identifier was stored in the `domain_key` of `ont_domains`. Switching between identifiers is necessary in the classified ads pages because the `domain` for the auxiliary domain information in `ad_domains` is used throughout the module. Most of the procedures in the ontology API use the `domain_id` as context for the procedure call. All of the procedures which display links pass the `domain_id` along with other variables to the listed Tcl pages. This means that any classified ads pages that are linked need to be able to handle getting a `domain_id` value instead of a `domain` value. The programmer has to translate the `domain_id` back to the `domain` in all cases, if the system being used has augmenting information. Another approach is to use the domain_id as the primary id in all the code.

4.3 Overall Effectiveness

The original classified ads system has about the same amount of code as the new system using the ontology API. A detraction from the API is the complexity of the interface. Unfortunately this was unavoidable given the myriad of options possible when displaying a hierarchy. The module eliminates the `ad_categories` table and removes the need for the category rows in the classified_ads table. The interface presented to the user has not changed with the addition of this API. Some additional functionality and alternate displays are available to the programmer for enhancements of the system. For example, the hierarchy depth can be greater than one level with the new system.

Improvements to the category code for the classified ads system can now be shared across any other services using the module. We can take advantage of the benefits of modularity. Multiple services and multiple domains within each service can all use this ontology system simultaneously.

In the effort to create a generally useful ontology system, we lose the conciseness of

code applied directly to the problem at hand. The code base for the ontology system is significant, but also more flexible to change than the built-in ontology system for classifieds.

4.4 Discussion Section

The major design decisions for this tool dealt with the hierarchy representation.

4.4.1 Multiple Parents

Given the use of the parent-child table, there was the option of having hierarchies that allowed multiple parents of a category. The tool could also allow the listing of an item in multiple categories. However, with multiple parents, various functions become very difficult to implement generically. More contextual information becomes necessary when calling category procedures. For example, if multiple parents are allowed, the current parent must be specified when moving a category. Otherwise, the system will not know which row to update in the hierarchy table. This would force the programmer to know more about the data model, a situation we would like to avoid in the interest of promoting modularity and maintainability.

There are similar difficulties with allowing an item to be posted in multiple categories. When deleting or moving an item, the behavior changes. As with moving a category, the procedure to move an item requires knowing its parent category in order to move the item. When deleting an item from a specific category, if it exists in multiple categories, the item should only be deleted from the category requested. However if it only exists in one category, it should be deleted from the system, including its entry in the `ont_categories`. Depending on the implementation of the system, this action becomes complex. Providing a feature such as multiple parents for a single item precludes the existence of any procedures which require a one-to-one mapping of items to categories.

Allowing multiple entries increases the range of capabilities available for the user and publisher. However, a single parent model simplified the design of the system.

Chapter 5

Conclusion

5.1 Future work

A software package is never complete. The following is a list of capabilities that would simplify the programmer's job and improve the experience for the end-user.

5.1.1 Minimum depth for adding an item

The present system allows items to be posted anywhere as long as they do not exceed the maximum depth of the domain hierarchy.

Publishers may want to specify where an item can be added to prevent unruly end-users from flooding the top of the hierarchy with self-serving content. Options could include allowing content to be placed at all levels of the ontology, or only past a certain depth, or only at the bottom level of the hierarchy. So Richard can say that end-users may only post in the second level of the tree, ensuring that only categories will be seen at the top level.

5.1.2 Multiple hierarchies

Occasionally it is difficult to determine the best hierarchy structure for a service. This is frequently an issue when location becomes important. Say that Richard is looking for a camera lens. He wants to buy it from someone in his area as he doesn't trust the parcel service. So he would like the listings to display only sellers in Boston. In contrast, Janet is looking for a rare movie camera. So she doesn't care about geographic location of the sellers, but rather their prices and quality ratings. Different hierarchy views of the same

content would let both Richard and Janet view the same listings using different axes of reference [6].

5.1.3 Pay attention to user input

How can we simplify navigation for end-users? It would be interesting to explore possible dynamic mutations of ontologies. Procedures could track end-users' clicking patterns and modify the site hierarchy to bring the most popular parts of the site to the top level. We could create a specialized view for each end-user without any specific personalization on the part of the end-user.

Various category policies can be established on top of the existing system. For example, let users make suggestions that will be visible on the hierarchy and implemented or deleted by administrators. The policies for adding categories can be more community-oriented also. End-users could vote on category suggestions, collectively deciding whether the suggestions should be followed or ignored.

5.1.4 Data transferral

Converting large bodies of data over to use this ontology tool requires an application to convert them into the appropriate category structure. Unfortunately, the category conversion will probably differ by individual system, as category implementations will vary.

5.1.5 Performance improvements

Performance could be improved by:

- adding indices to speed queries
- memoizing or caching to take advantage of common queries so they do not need to be reexecuted
- setting a limit on how many search results are listed on a single page to prevent a page from taking too long to display when searching for an item

5.1.6 More display options

The range of category displays can be expanded. In particular, an accordion-like display of the entire hierarchy of categories would be an interesting interface. With this procedure,

the end-user could click on a category and expand its contents, as a folder of files can be expanded or minimized.

5.1.7 Administration

In a site with heavy traffic and a complex ontology, category administrators might be needed to manage the content in individual branches. For advanced behavior the API can facilitate more sophisticated administration roles. This would allow more granularity in terms of who is allowed to do what. Statistics of how end-users go through the ontology and other behavior would also be useful for administrators. This could be implemented along with the features mentioned in 5.1.3.

5.2 Concluding remarks

Using the tool to rewrite a classified ads system, the API has been expanded to include general procedures that would replace more specific functions in the ads system. The process of rewriting the classified ads system with the ontology API shows that that the problem of coding an ontology can be abstracted sufficiently to form a categorization package.

The described ontology system design and implementation are effective alone and when integrated into the classified ads system. The set of API procedures duplicated the capability of the existing classified ads system and introduced more features.

Appendix A

SQL Data Model

---

at9.0pt

create sequence ont *at5.0pt at9.0pt at5.0pt at9.0pt at5.0pt at9.0pt at5.0pt at5.0pt at9.0pt at5.0pt at9.0pt at5.0pt at9.0pt at*

```
create table ont_domains (
        domain_id               integer primary key,
        domain_pretty_name      varchar(100) not null,
        -- the full name of the domain, properly formatted, etc.
        hierarchy_depth_limit   integer,
        domain_key              varchar(100),
        -- link to an external table for augmenting information
        service         varchar(20),
        -- directory name for the service using this domain
        active_p                char(1) default 't' check (active_p in ('t', 'f')),
        creation_date           date,
        on_which_table          varchar(50),
        -- which table it is going to link to
        id_column_name          varchar(50),
        -- what is the column name of the id in this table
        pretty_name_column      varchar(50),
        -- what is the pretty column name in this table
        item_base_url           varchar(100),
        -- the base URL of an item in the domain
        last_modified           date not null,
        last_modifying_user     not null references users,
        modified_ip_address     varchar(50) not null
);
```

```
create sequence ont_cat_id_sequence;

create table ont_categories (
        cat_id                  integer not null primary key,
        cat_pretty_name         varchar(1000) not null,
        -- the full name of the category, properly formatted, etc.
        extra_info              varchar(4000),
        domain_id               references ont_domains,
        -- identifying domain
        active_p                char(1) default 't' check (active_p in ('t', 'f')),
        creation_date           date,
        last_modified           date not null,
        last_modifying_user     not null references users,
        modified_ip_address     varchar(50) not null
);

create table ont_hierarchy (
        child_id            not null references ont_categories(cat_id),
        -- id number identifying child category
        parent_id           not null references ont_categories(cat_id),
        -- parent of the category specified by child (one level up in
        -- the hierarchy)
        last_modified           date not null,
        last_modifying_user     not null references users,
        modified_ip_address     varchar(50) not null,
        primary key(child_id, parent_id)
);

create index ont_hierarchy_by_parent_id
on ont_hierarchy (parent_id) tablespace ontology;

create index ont_hierarchy_by_child_id
on ont_hierarchy (child_id) tablespace ontology;

create sequence ont_default_item_id_sequence;

create table ont_default_items (
        item_id                 integer primary key,
```

```
    item_pretty_name      varchar(100),
    item_info             varchar(1000)
    domain_id             references ont_domains
);

-- For each domain this table is generated to link categories
-- with their contents.

-- create table ont_items_$domain_id (
--      cat_id                  references ont_categories,
--      item_id                 references ont_domains.on_which_table,
--      last_modified           date not null,
--      last_modifying_user     not null references users,
--      modified_ip_address     varchar(50) not null
--);
-- index on cat_id
-- index item_id


-- you can't do a JOIN with a CONNECT BY so we need a PL/SQL proc to
-- pull out category name, active_p, and domain_id from cat_id

create or replace function cat_pretty_name_from_cat_id(v_cat_id IN integer)
 return varchar
is
 pretty_name ont_categories.cat_pretty_name%TYPE;
BEGIN
  select cat_pretty_name INTO pretty_name
    from ont_categories
    where cat_id = v_cat_id;
   return pretty_name;
END cat_pretty_name_from_cat_id;
/
show errors

create or replace function active_p_from_cat_id(v_cat_id IN integer)
 return varchar
is
 active ont_categories.active_p%TYPE;
```

```
BEGIN
 select active_p INTO active
  from ont_categories
  where cat_id = v_cat_id;
 return active;
END active_p_from_cat_id;
/
show errors


create or replace function domain_id_from_cat_id(v_cat_id IN integer)
 return integer
is
 domain ont_domains.domain_id%TYPE;
BEGIN
 select domain_id INTO domain
  from ont_categories
  where cat_id = v_cat_id;
 return domain;
END domain_id_from_cat_id;
/
show errors
```

Appendix B

Online Ontology Documentation

- User directory: /ontology/
- Admin directory: /admin/ontology/
- data model: /doc/sql/ontology.sql
- procedures: /tcl/ontology-defs.tcl

B.1 The Big Picture

A standardized way of managing categorizations, or subsets of ontologies.

B.2 The Medium-sized Picture

This system can be used with several domains. Multiple publishers can use the same service. All of the information for a particular domain is specific to a particular publisher. A service using this package can support multiple publishers, but requires only one.

A domain can represent a particular topic or focus. Within the domain there are further categorizations appropriate for items under that topic. For example, a site could categorize both musical items and tropical fish information. Both of these topics may not belong in the same ontology, but can be supported by the same server.

```
create table ont_domains (
        domain_id               integer primary key,
        domain_pretty_name      varchar(100) not null,
        -- the full name of the domain, properly formatted, etc.
        hierarchy_depth_limitinteger,
        domain_key              varchar(100),
        -- link to an external table for augmenting information
        service                 varchar(20),
        -- directory name for the service using this domain
        active_p                char(1) default 't' check (active_p in ('t', 'f')),
        creation_date           date,
        on_which_table          varchar(50),
        -- which table it is going to link to
        id_column_name          varchar(50),
        -- what is the column name of the id in this table
        pretty_name_column      varchar(50),
        -- what is the pretty column name in this table
        item_base_url           varchar(100),
        -- the base URL of the an item in the module
        last_modified           date not null,
        last_modifying_user     not null references users,
        modified_ip_address     varchar(50) not null
        );
```

A publisher has the option of limiting the depth of the ontology to a specific value. `hierarchy_depth_limit` is blank if the depth is unlimited.

Taking categorized content and displaying user-intelligible information requires a few columns of information. The columns `on_which_table`, `id_name_column`, `pretty_name_column`, and `item_base_url` provide the details necessary to link to the content being categorized. `on_which_table` stores the name of the table which holds the content (collection of items). This table must have a primary key column so that individual items can be selected easily.

id_name_column holds the name of this column. An item listing in a category will look uninteresting and indistinguishable if only a primary key is displayed; in particular if the primary key happens to be a numeric value. For this reason pretty_name_column stores the name of a pretty name to use as a title or heading for the item. A last final bit of information is where a user should go to see the actual content, rather than these bits of summary details. Since this is web content, item_base_url stores the link to the actual content that the user wants to see.

Be warned that if the item_base_url is incorrect, there will be a broken link that users will click on unsuspectingly. Note also that the item_base_url is passed the primary key for the desired item. Thus the page stored in item_base_url should not require any additional information passed in.

```
create table ont_categories (
        cat_id                  integer not null primary key,
        cat_pretty_name         varchar(1000) not null,
        -- the full name of the category, properly formatted, etc.
        extra_info              varchar(4000),
        domain_id               references ont_domains,
        -- identifying domain
        active_p                char(1) default 't' check (active_p in ('t', 'f')),
        creation_date           date,
        last_modified           date not null,
        last_modifying_user     not null references users,
        modified_ip_address     varchar(50) not null
        );
```

Each category is a part of a particular domain.

```
create table ont_hierarchy (
        child_id               not null references ont_categories(cat_id),
        -- id number identifying child category
        parent_id              not null references ont_categories(cat_id),
        -- parent of the category specified by child (one level up in
```

```
-- the hierarchy)
last_modified         date not null,
last_modifying_user   not null references users,
modified_ip_address   varchar(50) not null,
primary key(child_id, parent_id)
);
```

This table defines the parent-child relationship for all categories, to dictate the structure of a hierarchy of categories. For the category to be in the hierarchy, it must be a child_id in at least one row of the table.

```
create table ont_items_$domain_id (
    cat_id                    references ont_categories,
    item_id                   references ont_domains.on_which_table,
    last_modified             date not null,
    last_modifying_user       not null references users,
    modified_ip_address       varchar(50) not null
    );
```

For each domain there is a table of this form, named after the specific domain, `ont_items_[$domain_id]`. This associates content with a particular category. Here, `item_id` references `on_which_table` in `ont_domains`. `item_id` is the primary key of the table that is holding the items (`on_which_table`).

B.3 Permissions

Appropriate user groups are set up automatically when a domain is created. They define the roles and corresponding permissions for each domain and each user in the domain.

- use the permissions package.
- Each domain has its own user group with `module=ontology` and `submodule=domain_id`.
- There is be a user group for each domain.
- Users in the group for a domain can administer that domain.

B.4 The Steps

B.4.1 Applying this Package to an Existing Module

Consider the classified ads system. Detailed below is the process of replacing the existing categorization scheme with the ontology package.

The classified ads system already has tables for maintaining domains and their corresponding categories. Since the ad_domains table maintains a significant amount of information that ont_domains does not, we keep it as an augmenting table for ont_domains, and reference it using the domain_key column in ont_domains. We can switch between the domain_id and domain_key using the functions `ont_domain_key_from_domain_id` and `ont_domain_id_from_domain_key`. The table classified_ads is the table that the domain references (on_which_table).

First, the admin pages /admin/gcnew/:

- In index.tcl, replace the select statement with calls to `ont_domain_list_html`.

- In domain-add-2.tcl, add a call to `ont_domain_key_exists` to the uniqueness check for the domain. Insert a call to `[ont_domain_add $db $full_noun "" "classified_ads" "classified_ad_id" "one_line" "/gcnew/view-one.tcl"]`. Remove the administration code, as `ont_domain_add` takes care of it. Place a request for the hierarchy depth in the form. Insert a call in domain-add-3.tcl to `[ont_domain_depth_edit $db $domain_id $hierarchy_depth]`.

  A simpler alteration to this set of pages is to insert the hierarchy depth request in the first domain-add page, and have a single call to `ont_domain_add` in the second page. Both approaches work equally well.

  Also, combine the two administrator links to both go to the user-groups page for the domain.

- insert calls to `ont_domain_toggle_active_p` in admin/gcnew/toggle-active-p.tcl. Use `ont_domain_id_from_domain_key` to get the domain_id.

- In domain-top.tcl, if the domain doesn't exist, call `ont_domain_key_from_domain_id`. Also, use `ont_domain_admin_group_id` get the group_id for the domain to be used for the user/helper administrators link.

- In domain-delete-2.tcl, insert a call to `ont_domain_delete`. Replace the count of ads in the domain with a call to `ont_domain_item_count`.

- In domain-edit2.tcl, insert a call to `ont_domain_edit` to update the title of the domain and domain_key.

- In manage-categories-for-domain.tcl, replace the category code with calls to `ont_cat_with_items_tree`.

- Modify category-add.tcl so the user also picks a parent category for the new category (using `ont_cat_all_select_box`). In category-add-2.tcl replace the code with a call to `ont_cat_add`. To guard against double clicks, use `ont_cat_id_new` to get a cat_id in the first category-add page.

  An alternative view could use `[ont_cat_tree_widget $db $domain_id "category-add-2.tcl?[expo cat_id]" "category-add-2.tcl?[export_url_vars cat_id]"]` to display a hierarchy tree for selecting the parent category first, then going to category-add.tcl and the existing chain of pages.

- Modify category-edit.tcl so that the user can also move a category by adding a link to the category-move.tcl pages. Copy over the category-move pages from /admin/ontology/ and modify them to fit your service. You can either change mentions of `domain` to `domain_id`, or use `ont_domain_id_from_domain_key` to extract the domain_id from the `domain` value.

- Insert a call to `ont_cat_edit` in category-edit-2.tcl, and use `ont_domain_id_from_domain_key`, `ont_cat_pretty_name`, `ont_domain_pretty_name`, and `ont_cat_extra_info` to extract any information you will need in category-edit.tcl

- In delete-category.tcl, insert a call to `ont_cat_deactivate`

- In ads-from-one-category.tcl, use the tcl list returned by `ont_item_list`, rewrite the code to loop through it and select information such as originating_ip, etc. Change URLs to pass cat_id to the delete page.

- Modify delete-ad-2.tcl to call `ont_item_delete` also. Be sure to specify the cat_id, as the procedure deletes the item in the specified category.

- Similarly, adjust delete-ads-from-one-user-2.tcl to use `ont_item_delete_spec_sql`.

- Modify edit-ad.tcl to call `ont_cat_all_select_box` and edit-ad-2.tcl to call `ont_item_move`.
- Modify ads.tcl to pass the value of domain to the delete and edit-ad pages.

Then, the user pages /gcnew/:

- in index.tcl, replace the domain listing with a call to `ont_domain_list_html`
- In domain-top.tcl, replace the listing of categories with a call to `ont_cat_with_items_tree_html`. Also extract the domain from the domain_id.
- in domain-all.tcl, the code allows for two ways to order the ads. For the listing by categories, replace the category selection with a call to `ont_cat_spec_items_tree_html`, and nested calls to `ont_subcat_list_html`.

  For the listing by items, use the code as normal.
- in place-ad.tcl replace the category listing with `ont_cat_list_all_html`. In place-ad-2.tcl replace mentions of primary_category with cat_id or cat_pretty_name. Place-ad-4.tcl insert a call to `ont_item_add`.
- In edit-ad-2.tcl, insert a call to `ont_cat_pretty_name_sub` to get a user-viewable category title. In edit-ad-4.tcl, replace the select box with a call to `ont_cat_all_select_box`. Insert a call to `ont_item_move` in edit-ad-5.tcl.
- Modify delete-ad-2.tcl to call `ont_item_delete` also.
- define-new-category-2.tcl, insert a call to `ont_cat_add`.
- In view-category.tcl, insert a call to `ont_item_list`.

B.4.2 Categorizing New Content

If the content you are categorizing already has a table to refer to it, skip the following step.

- Create a table to hold your data. Ideally the table has an id column, and a pretty name (like a title) column. And, since this is all web content, there should be a URL that takes the id column value and displays the information corresponding to that id. This is referred to as the item_base_url.

- Go to /ontology and click on Edit ontology domains.
- (user needs a password to get to the admin pages)
- Create a new domain called Pictures.
- Add categories as you would like.
- Go back to /ontology/ and add content.

B.5 Application Programming Interface

ont_cat_with_items_tree_html – Returns a nested HTML list showing the category hierarchy in a branched tree format. You can choose to get only categories that have items (or their sub-categories), or categories that don't have items.

ont_cat_id_new – Returns a new cat_id unused in the database.

ont_cat_add – Insert a new category with the specified name to the position specified by parent_id within the category hierarchy. Set parent_id to "" if adding the category to the top level of the hierarchy.

ont_cat_add_p – Returns 0 or 1 indicating whether a sub-category can be added at the specified position in the hierarchy. If cat_id is not supplied, it assumes that we are at the top of the hierarchy.

ont_cat_tree_widget – Returns the entire category hierarchy in a tree format - all active categories regardless of their contents.

ont_cat_edit – Edit the mutable attributes of a category: `cat_pretty_name` and `extra_info`.

ont_cat_deactivate – Remove a category and its contents from view.

ont_cat_items_count – Count the items in a category and its sub-categories.

ont_cat_pretty_name – The full name of the category.

ont_cat_extra_info – The value of the extra info field.

ont_cat_all_select_widget – A selection box containing all the categories in the domain.

ont_cat_list_all_html – list all active categories in the domain.

ont_subcat_list_html – A hyper-linked list of the sub-categories in the specified category, or the top of the hierarchy.

ont_cat_tree_widget – A nested hyper-linked list displaying the category hierarchy tree.

ont_cat_tree_options_widget – A nested hyper-linked list displaying the category hierarchy tree, with supplied options beside each category. This can be used to provide edit options for a category.

ont_context_fragment – Navigation (context) bar linked HTML fragment of the path down the hierarchy to the category specified.

ont_context_fragment_list – Context fragment Tcl list of the URL, title pairs for the path down the hierarchy to the category specified.

ont_domain_list_html List all the domains for the given web service, usually indicated by the name of the directory associated with it.

ont_domain_toggle_active_p – Deactivate or activate the domain depending on its current value.

ont_domain_add – Create a new domain.

ont_domain_edit – Edit the attributes of a domain (title, hierarchy depth, and item URL are the only edit-able attributes at present).

ont_domain_depth_edit – Update the hierarchy depth of the domain.

ont_domain_id – Look up the domain_id for the specified category.

ont_domain_id_from_domain_key – Look up the domain_id given the domain_key.

ont_domain_id_from_domain_key – Look up the domain_id given the domain_key.

ont_domain_pretty_name – Look up the full title for the domain.

ont_domain_delete – Permanently delete the specified domain.

ont_domain_id_from_domain_key – Look up the domain_id given the domain_key.

ont_domain_item_count – Count the items in the domain.

ont_hierarchy_depth_unlimited_p – Returns 1 if the specified domain has unlimited depth, 0 otherwise.

ont_item_add – Insert an item to the category specified. Set cat_id to "" or don't provide it if moving the item to the top of the hierarchy. User must be registered to add an item.

ont_item_add_p – Returns 1 if can add an item to the category specified by cat_id, 0 otherwise. Currently this returns 1 all the time if the user is part of the user-group

ont_item_delete – Delete an item from the category or top of the hierarchy.

ont_item_delete_spec_sql – Returns a sql statement that deletes the specified item from the category hierarchy.

ont_item_list – returns a Tcl list of all the item_id values in the specified category.

ont_item_list_html – Returns a linked HTML list of the items in the specified category.

ont_item_map_table_name – Name of the table that associates items with categories for a particular domain.

ont_item_move – Move the item to the category specified.

Appendix C

Tcl Procedures and Application Programming Interface

```
# ontology - defs . tcl
#
# by cslee @ alum . mit . edu , July 1999


proc_doc prep_tuple_list_for_args { list } "Appends ? or & to the url in the list of listed pairs of url, title elements, as appropriate." {
  set prepped_list ""

   foreach element $list {
     set url [lindex $element 0]
     set title [lindex $element 1]
     lappend prepped_list [list [prep_url_for_args $url] $title]
   }
   return $prepped_list
}


proc_doc prep_url_for_args { url } "Appends ? or & to the provided url as appropriate. If url contains arguments, a & will be appendec

   if { [string first "?" $url] > -1 } {
       # there is a ? so there should be at least one arg . passed in w / the url
       if { [string first "=" $url] > -1 } {
           # there appears to be an arg w / the url
```

```
            append url "&"
        }
    } else {
        # no ? so add one
        append url "?"
    }
    return $url
}


proc_doc ont_domain_key_exists { db domain_key } "Does the domain_key exist already? Returns number of occurrences in the table."

    return [database_to_tcl_string $db "select count(domain_key) from ont_domains where domain_key = '[DoubleApos $domain_key]
}


proc_doc ont_domain_id_exists { db domain_id } "Does the domain_id exist already? Returns number of occurrences in the table." {

    return [database_to_tcl_string $db "select count(domain_id) from ont_domains where domain_id = '[DoubleApos $domain_id]"']
}


proc_doc ont_domain_item_count { db domain_id } "Count the number of items in the domain." {

    set item_table [ont_items_map_table_name $domain_id]

    return [database_to_tcl_string $db "select count(*) from $item_table"]
}


proc_doc ont_domain_admin_group_id { db domain_id } "Look up the group_id of the administration group for the domain." {

    set group_name [ont_admin_user_group_name $db $domain_id]
    set helper_table [ad_user_group_helper_table_name "administration"]
    return [database_to_tcl_string_or_null $db "select group_id from user_groups where group_name='$group_name"' 0]
}
```

```
proc_doc ont_domain_delete { db domain_id } "Delete the domain" {

   set helper_table [ad_user_group_helper_table_name "administration"]
   set group_id [ont_domain_admin_group_id $db $domain_id]
   set user_id [ad_verify_and_get_user_id]

   set table_name [ont_item_map_table_name $domain_id]

   set list [ont_domain_item_info $db $domain_id]
   set id_column_name [lindex $list 1]
   set pretty_name_column [lindex $list 2]

   ns_db dml $db "delete from user_group_map_queue where group_id = $group_id"
   ns_db dml $db "delete from user_group_map where group_id = $group_id"

   ns_db dml $db "delete from ont_hierarchy
where parent_id in (select cat_id
              from ont_categories
              where domain_id = $domain_id)"

   ns_db dml $db "drop table $table_name"


   ns_db dml $db "delete from user_group_map where group_id=$group_id and user_id=$user_id"
   ns_db dml $db "delete from administration_info where group_id=$group_id"
   ns_db dml $db "delete from $helper_table where submodule = '$domain_id'"
   ns_db dml $db "delete from user_groups where group_id = $group_id"
   ns_db dml $db "delete from ont_domains where domain_id='$domain_id'"
}




proc_doc ont_domain_list_html { db active domain_url extra_list { service "" } } "list all domains, either active or inactive, as specified.

    if { $active } {
        set active "t"
    } else {
        set active "f"
```

```
}

set list ""
set counter 0

set domain_url [prep_url_for_args $domain_url]
set extra_urls [prep_tuple_list_for_args $extra_list]

set restriction ""
 if { ![empty_string_p $service] } {
     set restriction "and service = '[DoubleApos $service]"'
 }

set selection [ns_db select $db "select domain_id, domain_pretty_name from ont_domains where active_p = '$active' $restriction"]

 while { [ns_db getrow $db $selection] } {

   set_variables_after_query

   incr counter

   append list "<li><a href=\"$domain_url[export_url_vars domain_id]\">$domain_pretty_name</a>"

    foreach url_title $extra_urls {
      set url [lindex $url_title 0]
      set title [lindex $url_title 1]
      append list " | <a href=\"$url[export_url_vars domain_id]\">$title</a>"
   }
   append list "\n"
 }

 if { $counter == 0 } {
     append list "no domains found."
 }

 return $list
}

proc_doc ont_item_list { db domain_id top_p { input_cat_id "" } { sql_table "" } { sql_restriction "" } } " Returns a tcl list of the items
```

```
# Sql_table and sql_restriction allow you to further qualify the
# basis on which items are listed . Sql_table holds any table names
# ( separated by commas ) that are needed for the added restriction
# specified in sql_restriction .
# Sql_restriction holds any additions to the sql select , following
# a where clause . All column names in sql_restriction should be
# fully qualified with the corresponding table names . It should not
# contain where , as it follows a where clause , but if it is an
# addition to the where statement it should have an and .
# top_p : if value is 1 then cat_id is not expected , and a list of the
# top level items is returned .

set item_list ""

set item_map_table [ont_item_map_table_name $domain_id]

 if { [string first "," $sql_table] != 0 && ![empty_string_p $sql_table] } {
     set sql_table ", $sql_table"
 }

  set selection [ns_db 1row $db "select on_which_table, id_column_name, pretty_name_column, item_base_url
from ont_domains $sql_table
 where domain_id = '$domain_id'
 $sql_restriction
"]

  set_variables_after_query

   if { $top_p } {

     set selection [ns_db select $db "select $on_which_table.$id_column_name, $pretty_name_column
from $on_which_table, $item_map_table
 where cat_id is null
 and $item_map_table.item_id = $on_which_table.$id_column_name
"]

  } else {
     set selection [ns_db select $db "select $on_which_table.$id_column_name, $pretty_name_column
```

```
from $item_map_table, $on_which_table
 where cat_id = $input_cat_id
  and $item_map_table.item_id = $on_which_table.$id_column_name
"]
  }

   while { [ns_db getrow $db $selection] } {
     set_variables_after_query

     lappend item_list [set $id_column_name]
  }

   return $item_list
}

proc_doc ont_item_list_html { db domain_id top_p { input_cat_id "" } { sql_table "" } { sql_restriction "" } { url_args "" } } "Returns a

  # Sql_table and sql_restriction allow you to further qualify the
  # basis on which items are listed . Sql_table holds any table names
  # ( separated by commas ) that are needed for the added restriction
  # specified in sql_restriction .
  # Sql_restriction holds any additions to the sql select , following
  # a where clause . All column names in sql_restriction should be
  # fully qualified with the corresponding table names . It should not
  # contain where , as it follows a where clause , but if it is an
  # addition to the where statement it should have an and .
  # The base url for the hyperlinks is specified by the domain definition .
  # url_args : the value is appended to the end of the base
  # url , then the variables domain_id and the value of id_column_name
  # ( also specified in the domain definition ) are added too .
  # url_args should not begin with &
  # top_p : if value is 1 then cat_id is not expected , and a list of the
  # top level items is returned .

  set item_text ""

  set item_map_table [ont_item_map_table_name $domain_id]
```

```
  if { [string first "," $sql_table] != 0 && ![empty_string_p $sql_table] } {
      set sql_table ", $sql_table"
  }

  set selection [ns_db 1row $db "select on_which_table, id_column_name, pretty_name_column, item_base_url
from ont_domains $sql_table
 where domain_id = '$domain_id'
 $sql_restriction
   "]

  set_variables_after_query

  set item_base_url [prep_url_for_args $item_base_url]

   if { $top_p } {

     set selection [ns_db select $db "select $on_which_table.$id_column_name, $pretty_name_column
from $on_which_table, $item_map_table
 where cat_id is null
   and $item_map_table.item_id = $on_which_table.$id_column_name
"]

   } else {
     set selection [ns_db select $db "select $on_which_table.$id_column_name, $pretty_name_column
from $item_map_table, $on_which_table
 where cat_id = $input_cat_id
   and $item_map_table.item_id = $on_which_table.$id_column_name
"]
   }

    while { [ns_db getrow $db $selection] } {
      set_variables_after_query

      append item_text "<li><a href=\"$item_base_url[export_url_vars domain_id $id_column_name]\">[set $pretty_name_column]</
   }

    if { $item_text != "" } {
        return "\n <ul> \n$item_text \n</ul> \n"
   } else {
```

```
        return ""
    }

}


proc_doc ont_item_map_table_name { domain_id } "Returns name of the table that maps items to categories." {
    return ont_items_$domain_id
}

proc_doc ont_items_table_create { db domain_id } "Creates ont_items_\$domain_id table and indices on the cat_id and item_id. Return
   set selection [ns_db 1row $db "select on_which_table, id_column_name from ont_domains where domain_id = $domain_id"]

   set_variables_after_query

   set table [ont_item_map_table_name $domain_id]

   set create_sql "create table $table (
cat_id   references ont_categories,
item_id references $on_which_table\($id_column_name\)
)
"

   ns_db dml $db $create_sql
   ns_db dml $db "create index [set table]_cat_id_idx on $table (cat_id)"
   ns_db dml $db "create index [set table]_item_id_idx on $table (item_id)"
}

proc_doc ont_item_add_p { db domain_id { cat_id "" } } "Returns 1 if can add an item to the category specified by cat_id, 0 otherwise.

   set user_id [ad_verify_and_get_user_id]
    if { [empty_string_p $user_id] } {
        return 0
   }

    return 1
}
```

```
proc_doc ont_item_add { db domain_id item_id { cat_id "" } } "Adds the item_id specified to the category specified.   Set cat_id to \"'

   set user_id [ad_verify_and_get_user_id]

    if { [empty_string_p $user_id] } {
        return "Please register first."
    }

   set item_map_table [ont_item_map_table_name $domain_id]

   set insert_sql "insert into $item_map_table (cat_id, item_id)
values('$cat_id', '$item_id')
"

   ns_db dml $db $insert_sql
}

proc_doc ont_item_delete { db domain_id item_id { cat_id "" } } "Removes the item from the specified category.  If user does not have
   set user_id [ad_verify_and_get_user_id]

    if { [empty_string_p $user_id] } {
        return "Please register first."
    }

   set item_map_table [ont_item_map_table_name $domain_id]

   set selection [ns_db 1row $db "select on_which_table, id_column_name from ont_domains where domain_id = $domain_id"]

   set_variables_after_query

   ns_db dml $db "delete from $item_map_table where item_id = '$item_id"'
}

proc_doc ont_item_delete_spec_sql { db domain_id item_sql } "Returns sql statement to delete the items specified by item_sql from the

   set item_map_table [ont_item_map_table_name $domain_id]

    return "delete from $item_map_table $item_sql"
```

```
}

proc_doc ont_item_move { db domain_id item_id { cat_id "" } } "Moves the item_id specified to the category specified." {

  set item_map_table [ont_item_map_table_name $domain_id]

  set update_sql "
update $item_map_table
set cat_id = '$cat_id'
 where item_id = '$item_id'
"

  ns_db dml $db $update_sql
}

proc_doc ont_item_move_sql { domain_id item_id { cat_id "" } } "Returns the sql for moving the item_id specified to the category speci

  set item_map_table [ont_item_map_table_name $domain_id]

   return "update $item_map_table
set cat_id = '$cat_id'
 where item_id = '$item_id'
"
}

proc_doc ont_cat_id_from_item_id { db domain_id item_id } "Select the cat_id associated with the item_id in the specified domain. Retu
  set item_table [ont_item_map_table_name $domain_id]
  set selection [ns_db select $db "select cat_id from $item_table where item_id = '$item_id"']

   while { [ns_db getrow $db $selection] } {
    set_variables_after_query
     break
  }
  ns_db flush $db

   if { [info exists cat_id] } {
      return $cat_id
```

```
  } else {
    return 0
  }

}

proc_doc ont_cat_all_select_box { db domain_id { default_val "" } { select_name "category" } { size_subtag "size=10" } } "Returns a s

  set widget_value "<select name=\"$select_name\">\n"

  if { $default_val == "" } {
      append widget_value "<option value=\"none\" SELECTED>Choose a Category</option>\n"
  }

  set domain_name [ont_domain_pretty_name $db $domain_id]

  append widget_value "<option value=\"\">$domain_name</option>\n"

  set selection [ns_db select $db "select cat_id, cat_pretty_name from ont_categories where domain_id = '$domain_id' and active_p = '

  while { [ns_db getrow $db $selection] } {

    set_variables_after_query

    if { $default_val == $cat_id } {
        append widget_value "<option value=\"$cat_id\" SELECTED>[ns_quotehtml $cat_pretty_name]</option>\n"
    } else {
      append widget_value "<option value=\"$cat_id\">[ns_quotehtml $cat_pretty_name]</option>\n"
    }
  }

  append widget_value "</select>\n"
  return $widget_value
}

proc_doc ont_cat_tree_move_cat_widget { db domain_id domain_url cat_url move_cat } "returns a nested hyperlinked list displaying a hi

  set db_sub [ns_db gethandle subquery]
```

```
set domain_pretty_name [ont_domain_pretty_name $db $domain_id]

set widget ""

set counter 0

set unlimited_p [ont_hierarchy_depth_unlimited_p $db $domain_id]

# top of the hierarchy is level = 0

set limit [ont_domain_hierarchy_depth_limit $db $domain_id]

 if { $unlimited_p || $limit > 2 } {

    incr counter
    set prev_level -1

    # construct domain_url , cat_url

    set domain_url [prep_url_for_args $domain_url]
    set cat_url [prep_url_for_args $cat_url]

    append widget "<li><a href=\"$domain_url[export_url_vars domain_id]\">$domain_pretty_name</a> \n"

    # select all categories at top level , then connect by to find all their children
    set selection [ns_db select $db "select cat_id as top_cat, cat_pretty_name
from ont_categories
 where domain_id = $domain_id
   and active_p = 't'
   and cat_id <> $move_cat
   and not exists (select 1
                     from ont_hierarchy
                   where ont_hierarchy.child_id = ont_categories.cat_id)
 order by cat_pretty_name
"]

      while { [ns_db getrow $db $selection] } {

         set_variables_after_query
```

```
# top level categories are at level 0
set level 0

if { $prev_level < $level } {
    append widget "\n<ul>\n"
}
set prev_level $level

set cat_id $top_cat

append widget "<li><a href=\"$cat_url[export_url_vars domain_id cat_id]\">$cat_pretty_name</a>\n"

if { $unlimited_p } {

    set sub_selection [ns_db select $db_sub "select child_id as cat_id, level,
cat_pretty_name_from_cat_id(child_id) as pretty_name, active_p_from_cat_id(child_id) as active_p
from ont_hierarchy
 where active_p_from_cat_id(child_id) = 't'
   and child_id <> $move_cat
 start with parent_id = $top_cat
 connect by parent_id = prior child_id
"]

} elseif { $limit > [expr $level+2] } {

    set sub_selection [ns_db select $db_sub "select child_id as cat_id, level, cat_pretty_name_from_cat_id(child_id)
as pretty_name, active_p_from_cat_id(child_id) as active_p
from ont_hierarchy
 where active_p_from_cat_id(child_id) = 't'
   and level < $limit - 2
   and child_id <> $move_cat
 start with parent_id = $top_cat
 connect by parent_id = prior child_id
"]

} else {
    # skip subquery
    continue
```

```
        }

        while { [ns_db getrow $db_sub $sub_selection] } {
          set_variables_after_subquery

          if { $prev_level < $level } {
               append widget "\n<ul>\n"
          }

          append widget "<li><a href=\"$cat_url[export_url_vars domain_id cat_id]\">$pretty_name</a>\n"

          for {set i $level} {$i < $prev_level} {incr i} {
             append widget "<ul>\n\n"
          }

          set prev_level $level
        }
      }

      if { $counter > 0 } {
           while { $prev_level > -1 } {
           append widget "</ul>\n"
           set prev_level [expr $prev_level-1]
        }
      }

   }

   ns_db releasehandle $db_sub

    if { $counter == 0 } {
        append widget "no suitable categories found"
   }

    return "<ul>\n $widget </ul>"
}

proc_doc ont_cat_spec_items_tree_html { db domain_id { item_sql "" } { item_string ""} } "Returns a nested html list of the category h
```

```
# item_sql : should have a line in it saying where [ id_column_name ] = $ item_id , where id_column_name is the

set db_sub [ns_db gethandle subquery]

# select first level categories ( categories at the top of the hierarchy )
# for each first level cat that has items select all active children that have items
# save a list of cats , cat_pretty_name , and levels
set cat_list [ont_cat_with_items_level_list $db $domain_id]

set tree_html ""

set prev_level 0

 foreach id_name_level $cat_list {

   set branch_parent [lindex $id_name_level 0]
   set cat_id $branch_parent
   set cat_pretty_name [lindex $id_name_level 1]
   set level [lindex $id_name_level 2]

    if { $prev_level < $level } {
        append tree_html "\n<ul>\n"
   }

   append tree_html "<h3>$cat_pretty_name</h3>\n"

   # go through and select the items for each category
   set selection [ns_db select $db "select item_id from [ont_item_map_table_name $domain_id] where cat_id = $cat_id"]

    while { [ns_db getrow $db $selection] } {
      set_variables_after_query

      set sub_selection [ns_db 0or1row $db_sub [eval_sql $item_sql]]

       if { ![empty_string_p $sub_selection] } {
           set_variables_after_subquery
           append tree_html "<li>[eval_sql $item_string]\n"
      }
```

```
        }

        for { set i $prev_level } { $i < $level } { incr i } {
          append tree_html "\n</ul>\n"
        }

        set prev_level $level
    }

    ns_db releasehandle $db_sub

    while { $prev_level ≥ 0 } {
        append tree_html "\n</ul>\n"

        set prev_level [expr $prev_level−1]
    }

    return $tree_html
}

proc eval_sql {item_sql} {

    upvar eval_sql_sql_statement eval_sql_sql_statement

    set eval_sql_sql_statement $item_sql

    uplevel {
        eval $eval_sql_sql_statement
    }
}


proc_doc ont_cat_with_items_level_list { db domain_id } "Returns a tcl list of triples consisting of cat_id, cat_pretty_name, and level, rep

    # select all categories at top level ( cat_id and pretty_name )
    set top_cat_list [ont_cat_id_name_top_list $db $domain_id]

    set unweeded_cat_list ""
```

```
# get the count ( level ) for each category
foreach id_name_pair $top_cat_list {
  set branch_parent [lindex $id_name_pair 0]
  set cat_pretty_name [lindex $id_name_pair 1]
  set level 0

  lappend unweeded_cat_list [list $branch_parent $cat_pretty_name $level]

  set selection [ns_db select $db "select child_id as cat_id, level, cat_pretty_name_from_cat_id(child_id) as pretty_name
from ont_hierarchy
 where active_p_from_cat_id(child_id) = 't'
 start with parent_id = $branch_parent
 connect by parent_id = prior child_id
"]

    while { [ns_db getrow $db $selection] } {
      set_variables_after_query
      lappend unweeded_cat_list [list $cat_id $pretty_name $level]
    }
}

set total_cat_list ""
# weed out categories that don ' t have any items ( count = 0 )
foreach id_name_pair $unweeded_cat_list {

  set cat_id [lindex $id_name_pair 0]
  set cat_pretty_name [lindex $id_name_pair 1]
  set level [lindex $id_name_pair 2]

  set count [ont_cat_items_count $db $cat_id]

   if { $count > 0 } {
      lappend total_cat_list [list $cat_id $cat_pretty_name $level]
  }
}

 return $total_cat_list
}
```

```
proc_doc ont_cat_with_items_tree_html { db domain_id cat_url { with_items_p 1 } { show_domain_p 0 } { domain_url "" } { url_title_list

  # url_title_list : [ url1 title1 ] [ url2 title2 ] . . .

  set db_sub [ns_db gethandle subquery]

  set tree_html ""

  set prev_nesting_level 0

  if { $show_domain_p } {
      set domain_pretty_name [ont_domain_pretty_name $db $domain_id]

    set domain_url [prep_url_for_args $domain_url]

    set tree_html "<li><a href=\"$domain_url[export_url_vars domain_id]\">$domain_pretty_name</a> \n"

    # domain is at level - 1
    set prev_nesting_level -1
  }

  set cat_url [prep_url_for_args $cat_url]

  set prepped_url_pair_list [prep_tuple_list_for_args $url_title_list]

  # select all categories at top level , then connect by to find all their children
  set cat_id_name_list [ont_cat_id_name_top_list $db $domain_id]

  set counter 0

  foreach id_name_pair $cat_id_name_list {

    set level 0
    incr counter

    if { $level > $prev_nesting_level } {
        append tree_html "\n<ul>\n"
```

```
} else {
    for { set i $level } { $i < $prev_nesting_level } { incr i } {
        append tree_html "\n</ul>\n"
    }
}

set prev_nesting_level $level

set branch_parent [lindex $id_name_pair 0]
set cat_id $branch_parent
set cat_pretty_name [lindex $id_name_pair 1]

set count [ont_cat_items_count $db $cat_id]

if { $with_items_p && $count > 0 } {
    append tree_html "<li><a href=\"$cat_url[export_url_vars domain_id cat_id]\">$cat_pretty_name</a> ($count)"
} elseif { !$with_items_p && $count == 0 } {
    append tree_html "<li><a href=\"$cat_url[export_url_vars domain_id cat_id]\">$cat_pretty_name</a>"
} else {
    # skip this category
    continue
}

# tack on the extra hyperlinks

foreach url_title $prepped_url_pair_list {
    set url [lindex $url_title 0]
    set title [lindex $url_title 1]
    append tree_html " | <a href=\"$url[export_url_vars domain_id cat_id]\">$title</a>"
}

set selection [ns_db select $db "select child_id as cat_id, level, cat_pretty_name_from_cat_id(child_id) as pretty_name,
active_p_from_cat_id(child_id) as active_p
from ont_hierarchy
 where active_p_from_cat_id(child_id) = 't'
 start with parent_id = $branch_parent
 connect by parent_id = prior child_id
"]
```

```
while { [ns_db getrow $db $selection] } {
  set_variables_after_query

  set count [ont_cat_items_count $db_sub $cat_id]

   if { $with_items_p && $count > 0 } {
        if { $level > $prev_nesting_level } {
            append tree_html "\n<ul>\n"
     } else {
          for { set i $level } { $i < $prev_nesting_level } { incr i } {
            append tree_html "\n</ul>\n"
          }
     }
     set prev_nesting_level $level

     append tree_html "<li><a href=\"$cat_url[export_url_vars domain_id cat_id]\">$pretty_name</a> ($count)"

  } elseif { !$with_items_p && $count == 0 } {
      if { $level > $prev_nesting_level } {
          append tree_html "\n<ul>\n"
     } else {
          for { set i $level } { $i < $prev_nesting_level } { incr i } {
            append tree_html "\n</ul>\n"
          }
     }
     set prev_nesting_level $level

     append tree_html "<li><a href=\"$cat_url[export_url_vars domain_id cat_id]\">$pretty_name</a>"
  } else {
     # skip this category
      continue
  }

  # tack on the extra hyperlinks

   foreach url_title $prepped_url_pair_list {
     set url [lindex $url_title 0]
     set title [lindex $url_title 1]
```

```
            append tree_html " | <a href=\"$url[export_url_vars domain_id cat_id]\">$title</a>"
        }

        append tree_html "\n"
      }
    }

    ns_db releasehandle $db_sub

    if { $counter > 0 } {
        while { $prev_nesting_level > 0 } {
        append tree_html "\n</ul>\n"
        set prev_nesting_level [expr $prev_nesting_level-1]
      }
    } else {
      append tree_html "no categories defined currently"
    }

    if { $show_domain_p } {
        append tree_html "\n</ul>\n"
    }

    return "<ul>\n$tree_html\n</ul>"
}



proc_doc ont_cat_tree_widget {db domain_id domain_url cat_url { extra_url_title_list "" } } "Widget returns a nested hyperlinked list dis

    # extra_url_title_list : [ url1 title1 ] [ url2 title2 ] . . .

    set db_sub [ns_db gethandle subquery]

    set domain_pretty_name [ont_domain_pretty_name $db $domain_id]
    set domain_url [prep_url_for_args $domain_url]
    set cat_url [prep_url_for_args $cat_url]
    set prepped_url_pair_list [prep_tuple_list_for_args $extra_url_title_list]

    set widget "<li> <a href=\"$domain_url[export_url_vars domain_id]\">$domain_pretty_name</a>"
```

```
# tack on the extra links

 foreach element $prepped_url_pair_list {
   set url [lindex $element 0]
   set title [lindex $element 1]
   append widget " | <a href=\"$url[export_url_vars domain_id]\">$title</a>"
}

append widget "\n"

# select all categories at top level , then connect by to find all their children

set selection [ns_db select $db "select cat_id as branch_parent, cat_pretty_name from ont_categories
where domain_id = $domain_id
and active_p = 't'
and not exists (select 1
                from ont_hierarchy
                where ont_hierarchy.child_id = ont_categories.cat_id)
order by cat_pretty_name
"]

set prev_level -1

set counter 0

 while { [ns_db getrow $db $selection] } {
   set_variables_after_query

   # top level categories are at level 0
   set level 0

   incr counter

    if { $prev_level < $level } {
        append widget "\n<ul>\n"
   } else {
       for {set i $level} {$i < $prev_level} {incr i} {
         append widget "\n</ul>\n"
      }
```

```
}

set prev_level $level

set cat_id $branch_parent

append widget "<li><a href=\"$cat_url[export_url_vars domain_id cat_id]\">$cat_pretty_name</a>\n"

set sub_selection [ns_db select $db_sub "select child_id as cat_id, level, cat_pretty_name_from_cat_id(child_id) as pretty_name,
active_p_from_cat_id(child_id) as active_p
from ont_hierarchy
 where active_p_from_cat_id(child_id) = 't'
 start with parent_id = $branch_parent
 connect by parent_id = prior child_id
"]

  while { [ns_db getrow $db_sub $sub_selection] } {
    set_variables_after_subquery

     if { $level > $prev_level} {
        append widget "\n<ul>\n"
    } else {
       for {set i $level} {$i < $prev_level} {incr i} {
         append widget "\n</ul>\n"
      }
    }
    set prev_level $level

    append widget "<li><a href=\"$cat_url[export_url_vars domain_id cat_[id]\">$pretty_name</a>"

    # tack on the extra links

     foreach element $prepped_url_pair_list {
       set url [lindex $element 0]
       set title [lindex $element 1]
       append widget " | <a href=\"$url[export_url_vars domain_id cat_id]\">$title</a>"
    }

    append widget "\n"
```

```
        }
    }

    ns_db releasehandle $db_sub

    if { $counter > 0 } {
        while { $level > -1 } {
        append widget "\n</ul>\n"
        set level [expr $level-1]
        }
    }

    return "<ul>\n$widget</ul>"
}



proc_doc ont_context_fragment_list {db domain_id domain_url { ending_context "" } {input_cat_id ""} {cat_url ""} } "Returns a list of

    set domain_pretty_name [database_to_tcl_string_or_null $db "select domain_pretty_name from ont_domains where domain_id = '$dom

    set ancestor_list ""

    if { [string first "?" $domain_url] > -1 } {
        # there is a ? so there should be at least one arg . passed in w / the url
        if { [string first "=" $domain_url] > -1 } {
            # there appears to be an arg with the url
            append domain_url "&"
        }
    } else {
        # no ? so add one
        append domain_url "?"
    }

    # set up cat_url
    if { [string first "?" $cat_url] > -1 } {
        if { [string first "=" $cat_url] > -1 } {
            append cat_url "&"
        }
```

```
} else {
   append cat_url "?"
}

 if { ![empty_string_p $input_cat_id] } {

   lappend ancestor_list [list "$domain_url[export_url_vars domain_id]" "$domain_pretty_name"]

   set sql_query "select parent_id as cat_id, level, cat_pretty_name_from_cat_id(parent_id) as pretty_name
from ont_hierarchy
 start with child_id = $input_cat_id
 connect by child_id = prior parent_id
 order by level desc
"

    set selection [ns_db select $db $sql_query]

     while { [ns_db getrow $db $selection] } {
       set_variables_after_query

       set url "$cat_url[export_url_vars domain_id cat_id]"

       lappend ancestor_list [list "$url" "$pretty_name"]
    }

    set cat_pretty_name [database_to_tcl_string $db "select cat_pretty_name from ont_categories where cat_id = $input_cat_id"]

     if { ![empty_string_p $ending_context] } {
        # tack on the last category on the list ( linked )
        set cat_id $input_cat_id

      set url "$cat_url[export_url_vars domain_id cat_id]"

      lappend ancestor_list [list "$url" "$cat_pretty_name"]

      lappend ancestor_list "$ending_context"

    } else {
      # tack on the category unlinked
```

```
            lappend ancestor_list "$cat_pretty_name"
        }
    } else {
        if { ![empty_string_p $ending_context] } {
            lappend ancestor_list [list "$domain_url[export_url_vars domain_id]" "$domain_pretty_name"]

            lappend ancestor_list "$ending_context"
        } else {
            lappend ancestor_list "$domain_pretty_name"
        }
    }

    return $ancestor_list
}

proc_doc ont_context_fragment {db domain_id domain_url { ending_context "" } {input_cat_id ""} {cat_url ""} } "Returns a hyperlink

    set domain_pretty_name [database_to_tcl_string_or_null $db "select domain_pretty_name from ont_domains where domain_id = '$don

    set ancestors ""

    set domain_url [prep_url_for_args $domain_url]
    set cat_url [prep_url_for_args $cat_url]

    if { ![empty_string_p $input_cat_id] } {

        lappend ancestors [list "$domain_url[export_url_vars domain_id]" "$domain_pretty_name"]

        set sql_query "select parent_id as cat_id, level, cat_pretty_name_from_cat_id(parent_id) as pretty_name
from ont_hierarchy
 start with child_id = $input_cat_id
 connect by child_id = prior parent_id
 order by level desc
"

        set selection [ns_db select $db $sql_query]

        while { [ns_db getrow $db $selection] } {
            set_variables_after_query
```

```
        set url "$cat_url[export_url_vars domain_id cat_id]"

        lappend ancestors [list "$url" "$pretty_name"]
    }

    set cat_pretty_name [database_to_tcl_string $db "select cat_pretty_name from ont_categories where cat_id = $input_cat_id"]

     if { ![empty_string_p $ending_context] } {
        # tack on the last category on the list ( linked )
        set cat_id $input_cat_id

        set url "$cat_url[export_url_vars domain_id cat_id]"

        lappend ancestors [list "$url" "$cat_pretty_name"]

        lappend ancestors "$ending_context"

    } else {
        # tack on the category unlinked
        lappend ancestors "$cat_pretty_name"
    }
  } else {
     if { ![empty_string_p $ending_context] } {
         lappend ancestors [list "$domain_url[export_url_vars domain_id]" "$domain_pretty_name"]

        lappend ancestors "$ending_context"
    } else {
        lappend ancestors "$domain_pretty_name"
    }
  }
  set test_string "ad_context_bar $ancestors"

  set context_bar [ eval $test_string]

   return $context_bar
}

proc_doc ont_hierarchy_depth_unlimited_p { db domain_id } "Returns 1 if the hierarchy of the specified domain is unlimited, 0 otherwis
```

```
  set limit [ont_domain_hierarchy_depth_limit $db $domain_id]
   if { [string compare $limit ""] == 0 } {
        return 1
  } else {
      return 0
  }
}

proc_doc ont_domain_hierarchy_depth_limit { db domain_id } "Returns 1 if the specified domain has unlimited depth, 0 otherwise." {
   set limit [database_to_tcl_string_or_null $db "select hierarchy_depth_limit from ont_domains where domain_id = '$domain_id"' 0]
    return $limit
}

proc_doc ont_domain_id { db cat_id } "Look up the domain_id of the specified category." {
    return [database_to_tcl_string $db "select domain_id from ont_categories where cat_id = '$cat_id"']
}

proc_doc ont_domain_item_info { db domain_id } "Get the on_which_table, id_column_name, pretty_name_column, item_base_url values

   set selection [ns_db 1row $db "select on_which_table, id_column_name, pretty_name_column, item_base_url
   from ont_domains
   where domain_id = '$domain_id'
   "]

   set_variables_after_query
    return [list $on_which_table $id_column_name $pretty_name_column $item_base_url]
}


proc_doc ont_domain_id_from_domain_key { db domain_key } "Returns the domain_id for the domain_key specified, or -1 if there is no
    return [database_to_tcl_string_or_null $db "select domain_id from ont_domains where domain_key = '[DoubleApos $domain_key]"'
}


proc_doc ont_domain_key_from_domain_id { db domain_id } "Returns the domain_key for the domain_id specified, or -1 if there is no s
    return [database_to_tcl_string_or_null $db "select domain_key from ont_domains where domain_id = '$domain_id"' -1]
}
```

```
proc_doc ont_domain_id_new { db } "Return a new domain_id." {
    return [database_to_tcl_string $db "select ont_domain_id_sequence.nextval from dual"]
}


proc_doc ont_domain_pretty_name { db domain_id } "Look up the domain_pretty_name of the specified domain." {
    return [database_to_tcl_string_or_null $db "select domain_pretty_name from ont_domains where domain_id = '$domain_id"' "No N
}

proc_doc ont_domain_edit { db domain_id title { domain_key "" } { url "" } } "Edit the title, domain_key, and item url for the domain

    if { ![ad_permission_p $db "Ontology" $domain_id] } {
        return "You do not have permission to add an item."
    }

    # check input values
    set update_domain "update ont_domains set "

    set title_p 0

    if { ![empty_string_p $title] } {
        append update_domain "domain_pretty_name = '[DoubleApos $title]"'
        set title_p 1
    }

    set url_p 0

    if { ![empty_string_p $url] } {
        if {$title_p } {
            append update_domain ",\n"
        }
        append update_domain "item_base_url = '[DoubleApos $url]"'
        set url_p 1
    }

    set key_p 0

    if { ![empty_string_p $domain_key] } {
```

```
        if { $title_p || $url_p } {
            append update_domain ",\n"
        }
        append update_domain "domain_key = '[DoubleApos $domain_key]"'
        set key_p 1
    }

    set user_id [ad_get_user_id]

    set client_ip_address [ns_conn peeraddr]

    if { $title_p || $url_p || $key_p } {
        append update_domain ",\n"
    }

    append update_domain "last_modified = sysdate,
last_modifying_user = $user_id,
modified_ip_address = '$client_ip_address'
  where domain_id='$domain_id' "

    if [ catch { ns_db dml $db $update_domain } errmsg] {
        return $errmsg;
    }
}

proc_doc ont_domain_depth_edit { db domain_id depth } "Edit the hierarchy depth for the domain." {

    if { ![ad_permission_p $db "Ontology" $domain_id] } {
        return "You do not have permission to add an item."
    }

    # Error Count and List
    set exception_count 0
    set exception_text ""

    # check input values
    if { [regexp {([^0-9])} $depth] } {
        incr exception_count
```

```
        append exception_text "<li>The value you entered for the hierarchy depth limit is not a valid number.\n"
    }

    if { $exception_count > 0 } {
        return $exception_text
    }

    set user_id [ad_get_user_id]

    set client_ip_address [ns_conn peeraddr]

    set update_domain "update ont_domains set
hierarchy_depth_limit = '$depth',
last_modified = sysdate,
last_modifying_user = $user_id,
modified_ip_address = '$client_ip_address'
 where domain_id='$domain_id' "

    ns_db dml $db $update_domain
}


proc_doc ont_domain_toggle_active_p { db domain_id } "Toggle whether the domain is active or not." {
    if { ![ad_permission_p $db "Ontology" $domain_id] } {
        return "You do not have permission to edit the domain."
    }

    ns_db dml $db "update ont_domains set active_p = logical_negation(active_p) where domain_id = '$domain_id"'
}


proc_doc ont_domain_add { db domain_pretty_name hierarchy_depth_limit table_name id_column_name pretty_name_column item_base_u

    # Error Count and List
    set exception_count 0
    set exception_text ""

    if [empty_string_p $table_name] {
        if ![info exists Table] {
```

```
        incr exception_count
        set exception_text "You did not enter a table name"
    } else {
        set table_name $Table
    }
}

if ![ns_table exists $db $table_name] {
    incr exception_count
    set exception_text "The table named $table_name does not exist in the database.\n"
}

if ![ns_column exists $db $table_name $id_column_name] {
    incr exception_count
    set exception_text "The column $id_column_name does not exist in the table $tablename.\n"
}

if ![ns_column exists $db $table_name $pretty_name_column] {
    incr exception_count
    set exception_text "The column $pretty_name_column does not exist in the table named $table_name.\n"
}

# error - check input : hierarchy_depth_limit is a number
if { [regexp {([^0-9])} $hierarchy_depth_limit] } {
    incr exception_count
    append exception_text "The value you entered for the hierarchy depth limit is not a valid number.\n"
}

if {$exception_count > 0} {
    return $exception_text
}


if { [empty_string_p $domain_id] } {
    set domain_id [ont_domain_id_new $db]
}

set user_id [ad_get_user_id]
set client_ip_address [ns_conn peeraddr]
```

```
set insert_domain "insert into ont_domains
(domain_id,
domain_pretty_name,
hierarchy_depth_limit,
domain_key,
service,
creation_date,
on_which_table,
pretty_name_column,
id_column_name,
item_base_url,
last_modified,
last_modifying_user,
modified_ip_address)
values
($domain_id,
'[DoubleApos $domain_pretty_name]',
'[DoubleApos $hierarchy_depth_limit]',
'[DoubleApos $domain_key]',
'[DoubleApos $service]',
sysdate,
'[DoubleApos $table_name]',
'[DoubleApos $pretty_name_column]',
'[DoubleApos $id_column_name]',
'[DoubleApos $item_base_url]',
sysdate,
$user_id,
'$client_ip_address')"

set module_name "Ontology"

ns_db dml $db $insert_domain

set group_name [ont_admin_user_group_name $db $domain_id]

ad_administration_group_add $db $group_name $module_name $domain_id "/admin/ontology" "f"

ad_administration_group_user_add $db $admin_user_id "administrator" $module_name $domain_id
```

```
    ont_items_table_create $db $domain_id
}

proc_doc ont_cat_with_contents_list_html { db domain_id url { sql_table "" } { sql_restriction "" } { url_title_list "" } } "Returns a hype

    set url [prep_url_for_args $url]

    set prepped_url_pair_list ""

     foreach element $url_title_list {
        set url [lindex $element 0]
        set title [lindex $element 1]
        lappend prepped_url_pair_list [list [prep_url_for_args $url] $title]
    }

     if { ![empty_string_p $sql_table] && [string first "," $sql_table] != 0 } {
         set sql_table ", $sql_table"
    }

    set db_sub [ns_db gethandle subquery]

    set active_cats ""

    set sql_query "select cat_id, cat_pretty_name
from ont_categories $sql_table
  where domain_id = $domain_id
    and active_p = 't'
  $sql_restriction
  order by cat_pretty_name
    "

    set selection [ns_db select $db $sql_query]

     while { [ns_db getrow $db $selection] } {
        set_variables_after_query

        set count [ont_cat_items_count $db_sub $cat_id]
         if { $count > 0 } {
```

```
          append active_cats "<li><a href=\"$url[export_url_vars domain_id cat_id]\">$cat_pretty_name</a>"

        # tack on the extra hyperlinks

        foreach element $prepped_url_pair_list {
          set url [lindex $element 0]
          set title [lindex $element 1]
          append widget " | <a href=\"$url[export_url_vars domain_id cat_id]\">$title</a>"
        }

      }
   }

   ns_db releasehandle $db_sub

   return "\n <ul> \n$active_cats \n</ul> \n"
}


proc_doc ont_cat_id_new { db } "Returns a unique cat_id for use." {
    return [database_to_tcl_string $db "select ont_cat_id_sequence.nextval from dual"]
}



proc_doc ont_cat_parent_id { db cat_id } "Returns the cat_id of the parent of this category (the first one found if there are multiple par
   set selection [ns_db select $db "select parent_id from ont_hierarchy, ont_categories
where child_id = $cat_id
  and cat_id = $cat_id
  and active_p = 't' "]

    while { [ns_db getrow $db $selection] } {
      set_variables_after_query
       break
   }
   ns_db flush $db

    if { [info exists parent_id] } {
        return $parent_id
```

```
    } else {
        return 0
    }
}


proc_doc ont_cat_id_name_top_list { db domain_id } "Returns a tcl list of lists of cat_id, cat_prettyname for all the categories in the top

    set list ""

    set sql_query "select cat_id, cat_pretty_name
from ont_categories
  where domain_id = $domain_id
    and active_p = 't'
    and not exists (select 1 from ont_hierarchy where child_id = cat_id)
  order by cat_pretty_name
    "

    set selection [ns_db select $db $sql_query]

     while { [ns_db getrow $db $selection] } {

        set_variables_after_query

        lappend list [list $cat_id $cat_pretty_name]
    }

     return $list
}


proc_doc ont_cat_list_all_html { db domain_id url } "Returns a hyperlinked list of all the categories in the domain.  The base url for the

     if { [string first "?" $url] > -1 } {
         # there is a ? so there should be at least one arg . passed in w / the url
          if { [string first "=" $url] > -1 } {
              # there appears to be an arg with the url
              append url "&"
      }
```

```
  } else {
    # no ? so add one
    append url "?"
  }

  set db_sub [ns_db gethandle subquery]
  set active_cats ""

  set sql_query "select cat_id, cat_pretty_name from ont_categories
 where domain_id = $domain_id
  and active_p = 't'
 order by cat_pretty_name
  "

  set selection [ns_db select $db $sql_query]

   while { [ns_db getrow $db $selection] } {
     set_variables_after_query

     set count [ont_cat_items_count $db_sub $cat_id]

     append active_cats "<li><a href=\"$url[export_url_vars domain_id cat_id]\">$cat_pretty_name</a> ($count)"
  }

  ns_db releasehandle $db_sub

   return "\n <ul> \n$active_cats \n</ul> \n"
}

proc_doc ont_cat_pretty_name { db cat_id } "Look up the cat_pretty_name of the specified cat_id." {
   return [database_to_tcl_string $db "select cat_pretty_name from ont_categories where cat_id = '$cat_id"']
}

proc_doc ont_cat_pretty_name_sub { cat_id } "Look up the cat_pretty_name of the specified cat_id - grabs a db handle from the subqu

  set db_sub [ns_db gethandle subquery]

   return [database_to_tcl_string $db_sub "select cat_pretty_name from ont_categories where cat_id = '$cat_id"']
```

```
    ns_db releasehandle $db_sub
}


proc_doc ont_cat_extra_info { db cat_id } "Look up the extra_info for the specified cat_id." {
     return [database_to_tcl_string $db "select extra_info from ont_categories where cat_id = '$cat_id"']
}


proc_doc ont_cat_items_count { db cat_id } "Count the number of items in a category and any subcategories." {
    set domain_id [ont_domain_id $db $cat_id]
    set table_name [ont_item_map_table_name $domain_id]
    set child_count [database_to_tcl_string $db "select count(*)
from $table_name
 where cat_id in (select child_id
                  from ont_hierarchy
                  start with parent_id = $cat_id
                  connect by parent_id = prior child_id)
"]

     return [database_to_tcl_string $db "select count(*)+$child_count from $table_name where cat_id = $cat_id"]
}

proc_doc ont_subcat_list_html { db domain_id url top_p {input_cat_id ""} } "Returns a hyperlinked list of the subcategories in the spec


     if { [string first "?" $url] > -1 } {
         # there is a ? so there should be at least one arg . passed in w / the url
          if { [string first "=" $url] > -1 } {
              # there appears to be an arg with the url
              append url "&"
         }
    } else {
         # no ? so add one
         append url "?"
    }

    set db_sub [ns_db gethandle subquery]
    set active_cats ""
```

```
    if { $top_p } {

        set sql_query "select cat_id, cat_pretty_name
from ont_categories
  where not exists (select 1
                      from ont_hierarchy
                  where ont_hierarchy.child_id = ont_categories.cat_id)
    and domain_id = $domain_id
    and active_p = 't'
  order by cat_pretty_name
"
    } else {
        set sql_query "select cat_id, cat_pretty_name
from ont_categories, ont_hierarchy
  where active_p = 't'
    and parent_id = $input_cat_id
    and cat_id = child_id
  order by cat_pretty_name
"
    }

    set selection [ns_db select $db $sql_query]

     while { [ns_db getrow $db $selection] } {
        set_variables_after_query

        set count [ont_cat_items_count $db_sub $cat_id]
        append active_cats "<li><a href=\"$url[export_url_vars domain_id cat_id]\">$cat_pretty_name ($count)</a>"
    }

    ns_db releasehandle $db_sub

     if { $active_cats != "" } {
          return "\n <ul> \n$active_cats \n</ul> \n"
    } else {
        return ""
    }
```

```
}

proc_doc ont_cat_move { db domain_id cat_id parent_id new_parent_id} "move the category to the specified parent category. If new_pa

  set user_id [ad_verify_and_get_user_id]
   if { ![ad_permission_p $db "Ontology" $domain_id $user_id] } {
        return "Sorry, you do not appear to be authorized to access this page."
  }

   if { [empty_string_p $parent_id] } {
      # cat_id is a top level category - no parent in hierarchy
      set cur_parent_is_top_cat_p 1
  } else {
     set cur_parent_is_top_cat_p 0
  }

   if { [empty_string_p $new_parent_id] } {
      # moving to new parent in hierarchy
      set new_parent_is_top_cat_p 1
  } else {
     # - move cat_id to top of hierarchy
     set new_parent_is_top_cat_p 0
  }

  set modified_ip_address [ns_conn peeraddr]

   if { !$cur_parent_is_top_cat_p && !$new_parent_is_top_cat_p } {
      set sql "update ont_hierarchy
set parent_id = $new_parent_id,
last_modifying_user = '$user_id',
modified_ip_address = '$modified_ip_address',
last_modified = sysdate
 where child_id = $cat_id
  and parent_id = $parent_id"

  } elseif { !$cur_parent_is_top_cat_p && $new_parent_is_top_cat_p } {
     # moving cat to top of hierarchy
```

```
        set sql "delete from ont_hierarchy where child_id = $cat_id and parent_id = $parent_id"

    } elseif { $cur_parent_is_top_cat_p && !$new_parent_is_top_cat_p } {

        # add category to hierarchy

        set sql "insert into ont_hierarchy
(child_id, parent_id, last_modified, last_modifying_user, modified_ip_address)
values
($cat_id, $new_parent_id, sysdate, $user_id, '$modified_ip_address')
"
    } else {
        # no change
        return
    }

    if [ catch { ns_db dml $db $sql } errmsg] {
        return $errmsg
    }
}

proc_doc ont_cat_edit { db cat_id cat_pretty_name { extra_info "" } } "Edit the attributes of a category. " {

    if { [empty_string_p $cat_pretty_name] } {
        return "Category title missing"
    }

    set user_id [ad_get_user_id]

    set ip_address [ns_conn peeraddr]

    if { ![empty_string_p $extra_info] } {
        set update_sql "update ont_categories
set cat_pretty_name = '[DoubleApos $cat_pretty_name]',
extra_info = '[DoubleApos $extra_info]',
last_modifying_user = '$user_id',
modified_ip_address = '$ip_address',
last_modified = sysdate
  where cat_id = $cat_id"
```

```
    } else {
        set update_sql "update ont_categories
set cat_pretty_name = '[DoubleApos $cat_pretty_name]',
last_modifying_user = '$user_id',
modified_ip_address = '$ip_address',
last_modified = sysdate
  where cat_id = $cat_id"
    }

    if [ catch { ns_db dml $db $update_sql } errmsg] {
        return $errmsg
    }
}


proc_doc ont_cat_activate { db cat_id } "activate the specified category.  Returns errmsg if there is a problem activating the category."

    set domain_id [ont_domain_id $db $cat_id]

    if { ![ad_permission_p $db "Ontology" $domain_id] } {
        return "You do not have permission to activate a category."
    }

    set category_exists_p [database_to_tcl_string_or_null $db "select 1 from ont_categories where cat_id = $cat_id" 0]

    set user_id [ad_get_user_id]
    set client_ip_address [ns_conn peeraddr]

    ns_db dml $db "update ont_categories set active_p = 't',
last_modified = sysdate,
last_modifying_user = $user_id,
modified_ip_address = '$client_ip_address'
where cat_id = $cat_id"
}

proc_doc ont_cat_deactivate { db cat_id } "Deactivate the specified category.  Returns errmsg if there is a problem deactivating the ca
    # see if there is anything in this category : subcategories / items
```

```
set domain_id [ont_domain_id $db $cat_id]

 if { ![ad_permission_p $db "Ontology" $domain_id] } {
     return "You do not have permission to deactivate a category."
 }

set category_count [database_to_tcl_string_or_null $db "select count(*)
from ont_hierarchy, ont_categories
where active_p = 't' and parent_id = $cat_id" 0]

set item_count [database_to_tcl_string_or_null $db "select count(*) from [ont_item_map_table_name $domain_id] where cat_id = $cat

 if { $category_count > 0 || $item_count > 0 } {
     return "This category holds subcategories or items.  Please move or deactivate them before deactivating this category."
 }

set user_id [ad_get_user_id]
set client_ip_address [ns_conn peeraddr]

ns_db dml $db "update ont_categories set active_p = 'f',
last_modified = sysdate,
last_modifying_user = $user_id,
modified_ip_address = '$client_ip_address'
where cat_id = $cat_id"
}

proc_doc ont_cat_add_p { db domain_id { cat_id "" } } "Returns 1 if can add a sub-category to the specified cat_id, 0 otherwise.  If ca

 if { ![ad_permission_p $db "Ontology" $domain_id] } {
     return 0
 }

# see if we have reached limit specified by hierarchy_depth_limit
set limit [ont_domain_hierarchy_depth_limit $db $domain_id]

 if { [empty_string_p $cat_id] } {
     set current_level 1
 } else {
    set current_level [database_to_tcl_string_or_null $db "select greatest(level) from ont_hierarchy start with child_id = $cat_id connec
```

```
    }

    if { [ont_hierarchy_depth_unlimited_p $db $domain_id] || $limit > $current_level } {
        return 1
    } else {
        return 0
    }
}

proc_doc ont_cat_add { db domain_id cat_pretty_name extra_info parent_id { cat_id "" } } "Adds a new category with the specified nan

    if { ![ont_cat_add_p $db $domain_id $parent_id] } {
        return "Cannot add a category - either you do not have permission or this is the limit of the hierarchy depth."
    }

    if { [empty_string_p $cat_id] } {
        set cat_id [ont_cat_id_new $db]
    }

    set user_id [ad_get_user_id]

    set client_ip_address [ns_conn peeraddr]

    if { [empty_string_p $parent_id] } {
        set add_to_top_of_hierarchy_p 1
    } else {
        set add_to_top_of_hierarchy_p 0
    }

    set insert_category "
insert into ont_categories
(cat_id, cat_pretty_name, extra_info, domain_id, creation_date, last_modified, last_modifying_user, modified_ip_address)
values
($cat_id, '[DoubleApos $cat_pretty_name]', '[DoubleApos $extra_info]', '$domain_id', sysdate, sysdate, $user_id, '$client_ip_address')
"

    if { $add_to_top_of_hierarchy_p } {
        # add to ont_categories but don ' t add to hierarchy ( its parent is
        # domain , so it has no parent category )
```

```
if [ catch { ns_db dml $db $insert_category } errmsg] {

   if { [database_to_tcl_string $db "select count(*) from ont_categories where cat_id = $cat_id"] } {
      # user double clicked , this category already exists .
      return
   } else {
      return $errmsg
   }
} else {
   return
}
}

# otherwise , also insert into hierarchy , since it has a parent category
# Error Count and List

set exception_count 0
set exception_text ""

set child_parent_pair_exists_p [database_to_tcl_string_or_null $db "select count(*) from ont_hierarchy where child_id = $cat_id and (p

 if { $child_parent_pair_exists_p == 1 } {
     incr exception_count
     append exception_count "<li>The specified category is already in the category specified."
}

 if { $exception_count > 0 } {
      return "<ul>$exception_text</ul>"
}

set insert_hierarchy "
insert into ont_hierarchy
(child_id, parent_id, last_modified, last_modifying_user, modified_ip_address)
values
($cat_id, $parent_id, sysdate, $user_id, '$client_ip_address')
"

ns_db dml $db "begin transaction"
```

```
if [ catch { ns_db dml $db $insert_category } errmsg] {

  ns_db dml $db "end transaction"

  set cat_id_exists_p [database_to_tcl_string $db "select count(cat_id) from ont_categories where cat_id = $cat_id"]

   if { $cat_id_exists_p == 1 } {
      # user double clicked , this category already exists .
       return
  } else {
      return $errmsg
  }
}

 if [ catch { ns_db dml $db $insert_hierarchy } errmsg] {
    ns_db dml $db "rollback"

    return $errmsg
 }
 ns_db dml $db "end transaction"
}

proc_doc ont_admin_user_group_name { db domain_id } "Returns the name of the administration user_group for the specified domain"
  set domain_pretty_name [ont_domain_pretty_name $db $domain_id]
   return "$domain_pretty_name \($domain_id\) Administrators"
}
```

References

[1] Tracy Adams, "Permission Package," *http://photo.net/doc/permissions.html.*

[2] America OnLine, "AOLserver," *http://aolserver.com*, 1995.

[3] Andover.Net, "Slashdot homepage," *http://www.slashdot.org*, 1997-1999.

[4] Michael Bryzek, Richard Li, et. al., "Virtual Compassion Corps homepage," *http://www.CompassionCorps.org*, 1999.

[5] eBay, "eBay homepage," *http://www.ebay.com*, 1995-1999.

[6] Philip Greenspun, "Developers Guide," *http://photo.net/doc/developers.html.*

[7] Philip Greenspun, "Generic Classifieds," *http://photo.net/gc/.*

[8] Philip Greenspun, "Chapter 3: Scalable Systems for Online Communities," *http://photo.net/wtr/thebook/community.html*, 1997.

[9] Philip Greenspun, "Chapter 13: Case Studies," *http://photo.net/wtr/dead-trees/53013.htm*, 1997.

[10] Philip Greenspun, "User Groups," *http://photo.net/doc/user-groups.html.*

[11] Salon Entertainment, "Salon Entertainment homepage," *http://salon.com*, 1999.

[12] Yahoo!, "Yahoo! homepage," *http://www.yahoo.com*, 1999.

[13] ArsDigita LLC, "The ArsDigita Community System," *http://software.arsdigita.com*, 1999.