Profiles Research Networking Software (Beta Version)
Installation Guide

November 10, 2010

Hardware and Operating System Requirements

Profiles is a Microsoft .NET 3.5 website that uses a Microsoft SQL Server 2005 (or 2008) database. You can use the same server (or a virtual machine) for the website and database; however, we recommend you se
parate the two. The database for Profiles is much more resource intense then the website. The database will require about 10 GB for 10,000 people. This is not a lot of space, but having a fast disk and as much CPU and RAM as possible for the database will benefit performance more than building a more robust web server. The website itself uses little disk space and bandwidth.

Outline of Installation Process

1. Install the database
2. Place basic HR data (name, email, address, affiliation, faculty rank, etc.) into database "loading" tables.
3. Run a series of processes that use the loading tables to populate the remaining database tables and automatically locate publications from PubMed.
4. Setup scheduled database jobs to keep the data up-to-date.
5. Install and configure the website code.

Database Components

There are three database components of Profiles:

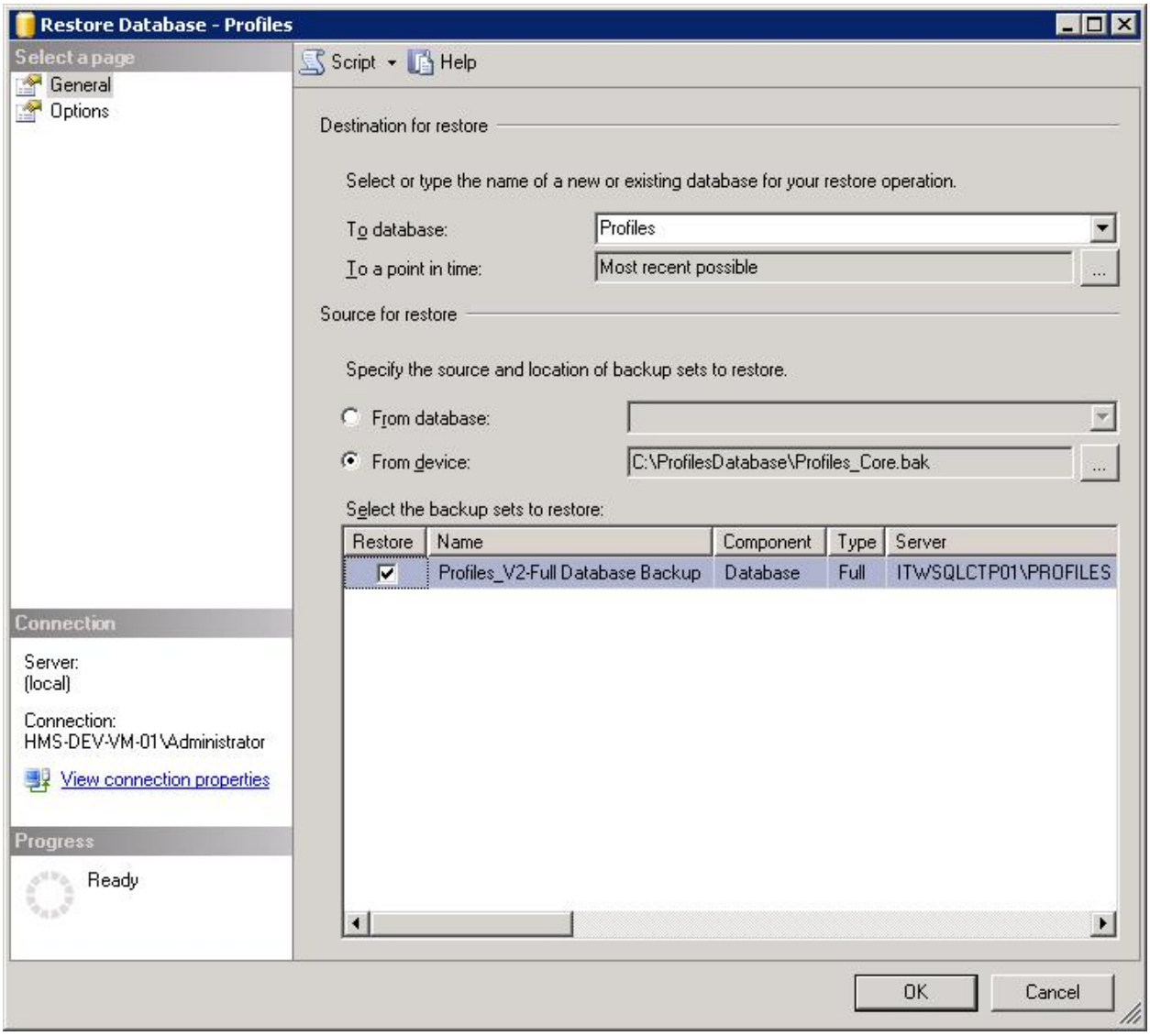
1. There is a profiles database that contains the data tables, stored procedures, and other database objects used by the website. It is distributed as a backup file, which you can restore into an empty SQL Server database. Some of the tables in the backup file are pre-populated with data, such as a set of tables that contain the Medical Subject Heading (MeSH 2010) vocabulary. You will need to provide the database with basic person information (e.g., names, titles, and affiliations), typically as a one-time data load or an automated feed from your human resource (HR) database.
2. There are three SSIS packages that are used to pull data from external sources. The first takes the person data you loaded into Profiles and calls a "disambiguation engine" web service hosted at Harvard to obtain Medline/Pubmed articles for those people. The second takes any address information you provided about people and calls a Google API to convert these to latitude and longitude coordinates.
3. To optimize performance, there are several "cache" tables in the database, which contain aggregate counts and denormalized copies of the raw data. To keep these cache tables up-to-date, Profiles contains a series of database jobs, which can be scheduled to run periodically to rebuild these tables.

Installing the Database

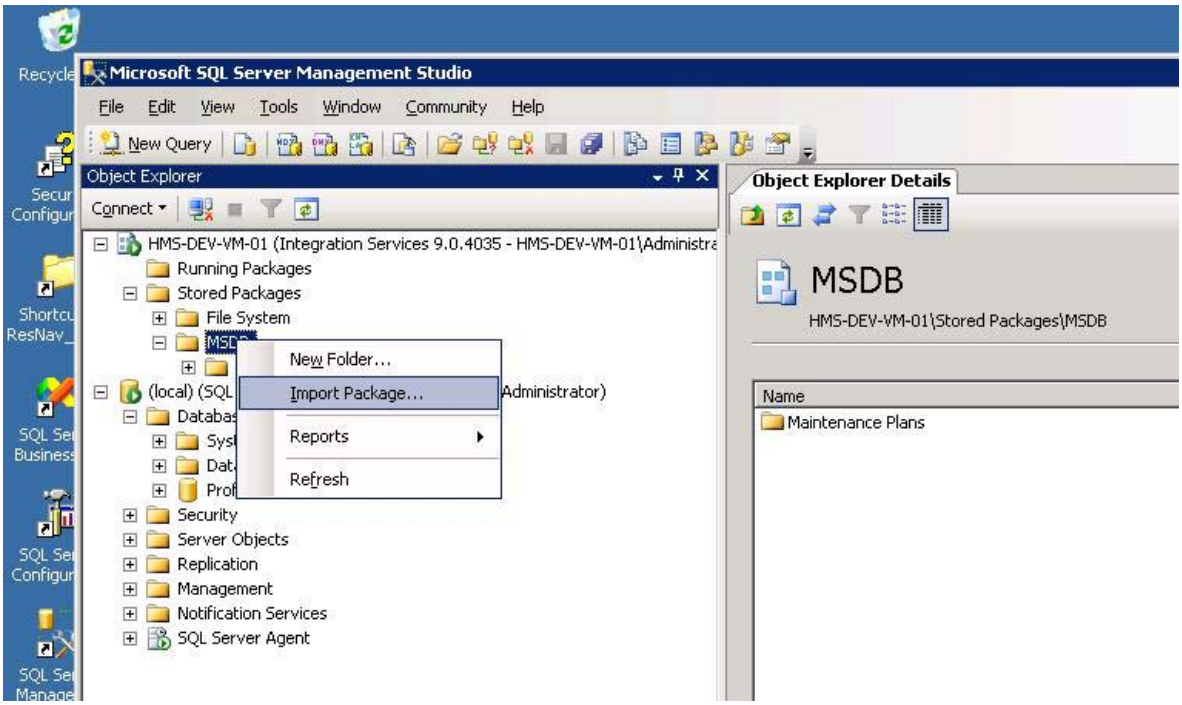
Follow the steps below to install the database:

1. Unzip the compressed database file.
2. Restore Profiles_Core.bak to an empty SQL Server database.

- The recommended database name is “Profiles”



3. Import the following SSIS packages into the SQL Server msdb database:
    - PubMedDisambiguation_GetPubs.dtsx
    - PubMedDisambiguation_GetPubMEDXML.dtsx
    - ProfilesGeoCode.dtsx



4. Create an admin account with sysadmin privileges that can be used to execute a series of scheduled jobs.
    - The script “ProfilesSetUserPermissions.sql” will create a default admin/application user account, AppProfilesUser, with all of the appropriate permissions.
    - If you want to use a database user account other than AppProfilesUser, then that account will need to be a sysadmin and you will have to grant execute permissions for all stored procedures in the Profiles database. (We recommend that you use the ProfilesSetUserPermissions.sql script as a template to help you do this. Just do a global search and replace of the AppProfilesUser, with your new database username. The default password can also be changed in the first part of this script. NOTE - AppProfilesUser is the recommended/default account.)
    - If you want to use a database user account other than the default account setup provided in the Profiles_Core.bak database, then remember to grant appropriate stored procedure execute permissions. The script ProfilesSetUserPermissions.sql is a template to help you do this. Just do a global search and replace of the AppProfilesUser, with your new database username. The default password can also be changed in the first part of this script. NOTE - AppProfilesUser is the recommended/default account.

5. The following scripts that create scheduled jobs need to be modified so that they work in your particular environment:
    - ProfilesYearlyJobs.sql
    - ProfilesNightlyJobs.sql
    - ProfilesWeeklyJobs.sql
    - ProfilesMonthlyJobs.sql
    - ProfilesHourlyJobs.sql

- PubMedDisambiguation_GetPubs.sql
- PubMedDisambiguation_GetPubMEDXML.sql
- ProfilesGeoCodeJob.sql

For each of the scripts, modify the following parameters in the sql code:

- @owner_login_name – the name of the sysadmin account created in step #4.
- @database_name – the name of the profiles database.
- @server_name – the name of the sql server instance.

6. The following scripts that call SSIS Packages need to be modified so that they work in your particular environment:
   - PubMedDisambiguation_GetPubs.sql
   - PubMedDisambiguation_GetPubMEDXML.sql
   - ProfilesGeoCodeJob.sql

   For each of the scripts listed in step #6, modify the following parameters in the sql code:
   - Replace `YourProfilesServerName` with the name of your Profiles database server.
   - Replace `YourProfilesDatabaseName` with the name of your Profiles database.

7. Execute the scripts you modified in steps #5 and #6.

Loading Person Data

Profiles requires that you provide basic demographic data about people. This can be a one-time import, or more likely you will want to setup a data feed from a human resources (HR) system that updates Profiles nightly. The general process is you place the data in a set of "loading" tables, and then Profiles will copy the data into the actual tables used by the website. During this step, Profiles will automatically create unique IDs for people and generate several lookup tables. There are several concepts to be aware of with how Profiles handles person data:

1. Profiles makes a distinction between the people who have profiles (Persons) and the people who can login to the website (Users). In general, Persons will be a subset of Users. At a typical academic institution, the Persons will be faculty, and the Users will be the faculty, staff, and students. Note that in order for someone to be able to use features of the site that require a login, such as "active networking" and "proxies", he or she will need to have a user account.
2. There are four main loading tables:
   a. The _person table has one row per person and includes fields such as first name, middle name, last name, name suffix, email, phone, fax, and address. The isactive and isvisible fields need to be set to 1 for the person to appear on the website.
   b. The _person_affiliations table lists a person's titles, institutions, departments, divisions, and faculty rank (e.g., "associate professor"). There can be multiple records in this table per person, reflecting the multiple jobs or roles a person has in your organization. One affiliation should be marked as the primary affiliation. Note that address is a person level attribute in Profiles, not an affiliation level attribute. The one address chosen should be the single best way that the person can be contacted. Set the departmentvisible flag in the _person_affiliations table to 1 for any department names that should appear in the Department drop-down on the Profiles website search form.

    c. The _person_filter_flags table allows you to extend the data model with custom Boolean flags that are relevant to your organization, such as “emeritus”, “visiting”, “student”, etc. A person can have multiple flags. Flags can be grouped into categories. For example, “faculty”, “staff”, and “student” can be grouped into a category named “job type”..
    d. The _user table has one row per user and includes basic user name and affiliation information. The _user table is used to load any person who needs access to the website, but won’t have their own profile. The Profiles Designated Proxy feature allows profile owners to grant editing permission to specific users. However, users who are eligible to be proxies must have the _user.CanBeProxy field set to 1.
3. The Profiles loading tables contain many fields. You do not need to populate all of these. None are required to run Profiles. However, certain functionality requires particular fields. For example, an office address is needed for Profiles to indicate a person’s location on a Google Map. If a data field is missing, Profiles will simply not show the feature. It won’t generate an error. You can also provide different amounts of data for different people. For author disambiguation (finding PubMed articles) to work best, include the firstname, middlename, lastname, suffix, and emailaddr fields.
4. We recognize that HR data is often “messy” and the institution, department, and division names stored in your HR database might be different than how you want these displayed on the Profiles website. Therefore, Profiles contains mapping tables that convert actual affiliation names to those used on the website. The fields for the actual names are “institutionfullname”, “departmentfullname”, and “divisionfullname”. The fields for the display names are just “institution”, “department”, and “division”.
5. Although Profiles generates it own unique IDs for people, the person and user tables contain database fields for Internal Identifiers. These are your local institutional identifiers for these individuals. They are not displayed on the Profiles website. Rather, they are used to help you create data feeds to update Profiles or to pull data from Profiles and merge with other systems.

The steps to loading person data are:

1. Populate the loading tables _person, _person_affiliations, _person_filter_flags, and _user. Fill in as many fields as you can, but none are required.
2. When you populate the _person_affiliations table, you can place actual institution, department, and division names in the “*fullname” fields; and, in the “institution”, “department”, and “division” fields, place the values that you want to appear on the website. If there is no difference between, simply place the affiliation names in both sets of fields.
3. There are four fields in _person_affiliations used for faculty (academic) rank. The main ones are “facultyrank”, which contains names like “Full Professor”, and “facultyrankorder”, which indicates the sort order of faculty ranks, starting (typically) at “0” for full professors, “1” for associate professors, etc. The “facultyrankcode” and “facultyrankfullname” are to store more precise job codes and faculty rank descriptions. These two fields are not displayed on the website. Like the “*fullname” affiliation fields, these are to help with back-end data flows.
4. Execute the stored procedure “usp_loadprofilesdata”. This will parse the data in the loading tables and populate the actual person, user, and related tables. This procedure takes 1 input parameter, @use_internalid_as_pkey. If this is set to 1, then Profiles will use the internalid fields in the _person and _user tables as the PersonID and UserID. Otherwise, Profiles will generate its own unique identifiers.

5. Profiles assumes that each person filter is its own independent flag. If certain filters can be grouped into categories, you can edit the PersonFilterCategory field in the person_filters table. You can also edit the PersonFilterSort field in the person_filters table to set the order in which filters appear in the Profiles search form.

Geocoding and Google Maps

In order for Profiles to display the locations of faculty on a map:

1. Profiles must convert person addresses to latitude and longitude coordinates. To do this conversion, after you have completed loading the person data, run the ProfilesGeoCode job. This will send each unique address in the database to a Google web service API, which will return the coordinates. Note that Profiles can only process 3 unique addresses per second because of Google's policy on how frequently its API can be called.
   In order to render a Google Map on the website, you will need to register a Google Map Key (http://code.google.com/apis/maps/signup.html). Google Map Keys are unique to the domain of the web page rendering the map. So, you might need to register multiple Google Map Keys, depending on how you setup Profiles in your environment. After you register the keys, insert them into the Profiles database table named "google_keys". This table has three fields. The first is the "app", which in this case should be set to the value "maps". The second is the "domain", which should look like "http://yourprofilesdomain". The third is "gkey", which is the sequence of characters gave you when you registered for the key. For example, "ABQIAAAAgvUCyYlXrLbSiYTl-Ja-8xTtdeBmLYMtQ2odxA5xusXMxYgP8xTfnaDW_yS9dIO1XknM7BVmuODxkQ".

   ```
   insert into google_keys (app, domain, gkey)
      values ('maps',
               'http://yourprofilesdomain',
               'ABQIAAA…7BVmuODxkQ')
   ```

2. When displaying Google Maps, Profiles lets the user select from several pre-defined zoom and centering configurations. For example, by default, Harvard's Profiles shows the city of Boston, but users can click a link to zoom out to show all of New England. Enter your list of map configurations in the google_map_preferences table. One row in this table should be flagged as DefaultLevel = 1. Note that http://maps.google.com has a new Google Labs feature that shows you the latitude and longitude of where your mouse is pointing. This can be helpful for editing the google_map_preferences table.

Obtaining Publications

In order for Profiles to automatically locate publications for people, you first need to provide a list of affiliation strings in the disambiguation_pm_affiliations table. These are phrases, which can include wildcard characters ("%"), that represent the most likely ways that your researchers will list their affiliations in Medline/Pubmed. Strings are not case sensitive. Selecting affiliation strings is somewhat of an art. The more precise the strings, the easier it is for Profiles to find publications. However, if the strings are too narrow in scope, Profiles might miss some articles. Examples of strings that we use at Harvard include:

```
%Harvard Medical School%
%Beth Israel Deaconess Medical Center%
%BIDMC%
```

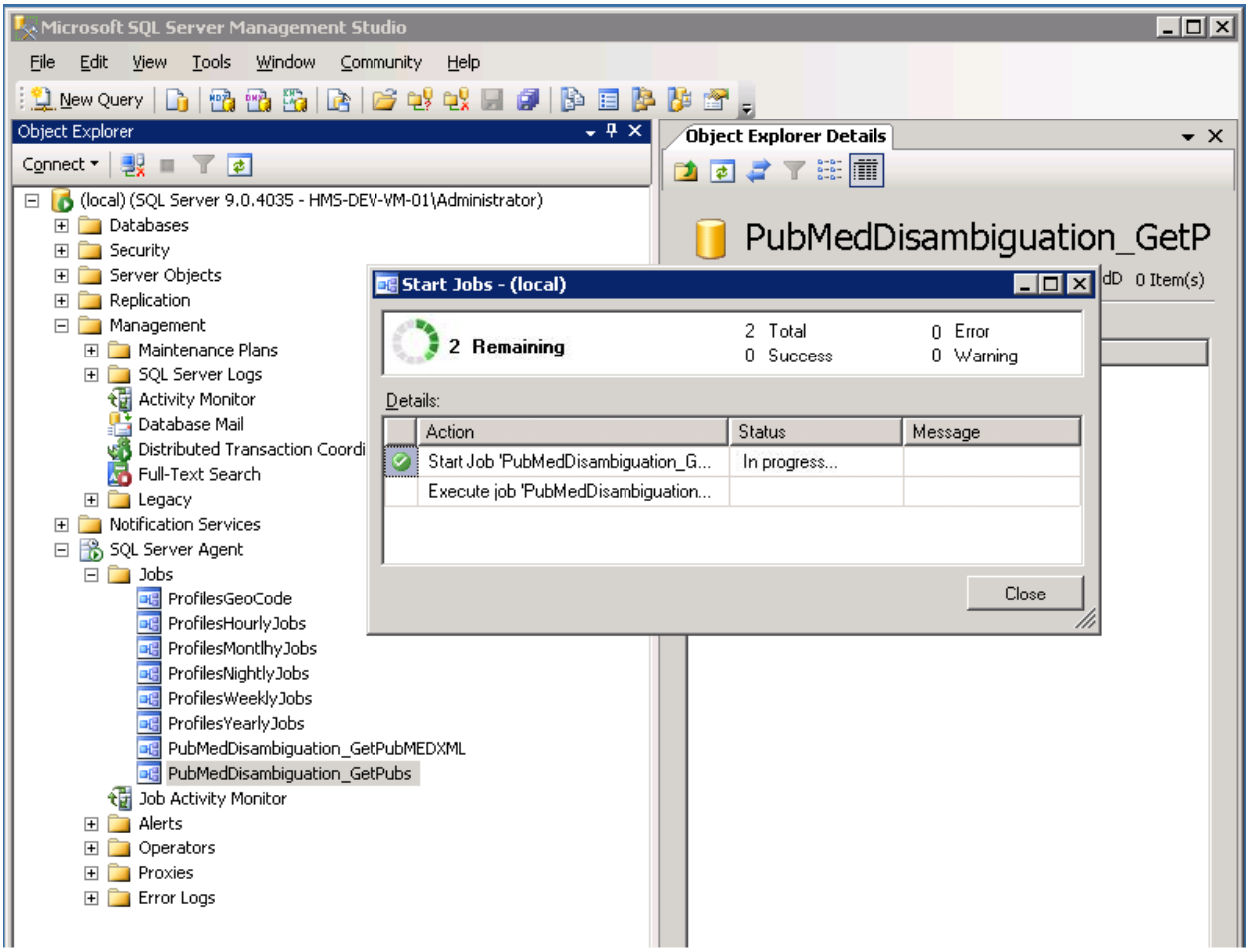
%@hms.harvard.edu%
%Children's Hospital%02115%

Example:

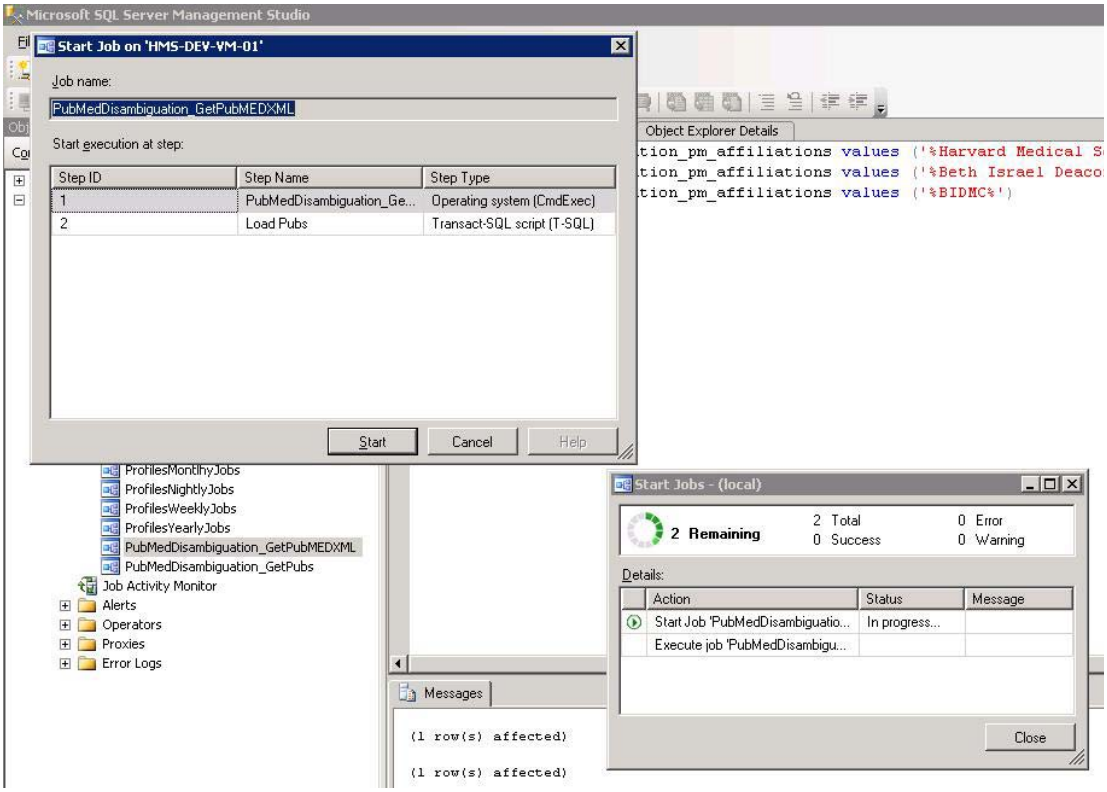
```
insert into disambiguation_pm_affiliations (affiliation)
   values ('%Harvard Medical School%')
insert into disambiguation_pm_affiliations (affiliation)
   values ('%Beth Israel Deaconess Medical Center%')
insert into disambiguation_pm_affiliations (affiliation)
   values ('%BIDMC%')
insert into disambiguation_pm_affiliations (affiliation)
   values ('%@hms.harvard.edu%')
insert into disambiguation_pm_affiliations (affiliation)
   values ('%Children's Hospital%02115%')
```

Examples of strings that we do <u>not</u> use at Harvard because they would be too broad are:
%Beth Israel%
%Department of Medicine%

Once the person data is loaded, and you have entered your affiliation strings, run the PubMedDisambiguation_GetPubs job to call the Profiles Disambiguation Engine web service to find Medline/Pubmed articles for people.



Once this job has completed (typically several hours), you should run the PubMedDisambiguation_GetPubMEDXML job. This job will retrieve the full xml for the pubmed articles and parse it in your local profiles instance.



There are a few important technical notes about the service:

1. The service will take about 5 seconds per person on average, provided you are the only one using the service. If another institution is calling the service at the same time, the run time will be slower.
2. The URL of the service is likely to change in the next few months. If you are unable to connect to the service, please contact us for details.
3. It is possible to host your own local instance of this service. However, its hardware and storage requirements are significantly greater than the main Profiles database and website. For example, you will need to have a local copy of the entire Medline database, which is several hundred gigabytes.

Below are a few general comments about the disambiguation engine:

1. Although the affiliation strings help the service find publications, it does not limit the search. The affiliation strings are used to identify “seed” publications. These are publications that are most likely correct matches. The disambiguation engine then searches all of Medline/Pubmed, using information about the seed publications, such as their titles, MeSH terms, coauthors, and journals, to find additional articles.

2. All publications are assigned a match probability. By default, the disambiguation engine uses a 98% probability threshold, meaning it will only return publications that are very likely correct matches. You have the option of lowering this threshold. This will reduce the chances that correct publications are missed, but it will increase the chances that incorrect publications are added to people's profiles. In general, select a low threshold if your goal is to create the "most accurate" profiles, meaning as many people as possible have close to correct publication lists. However, select a high threshold if your goal is to create the "cleanest" search results and passive networks. We set the default threshold high because it is easy for faculty (or their proxies) to add missing publications, but the website loses much of its value if the search results return the wrong people or if passive networks (e.g., top keywords, co-authors, similar people, etc.) contain meaningless information. Note that just a single incorrect publication can greatly alter a person's passive networks, but even multiple missing publications will have far less effect because an expert in a field will have many other publications in that same area. To change the threshold, modify the @threshold variable value defined within the usp_GetPersonInfoForPubMed_xml stored procedure.
3. Profiles will have the most difficulty with common names (e.g., J Smith), names with multiple parts (e.g., a hyphenated last name), names with foreign characters, and people who only recently joined your organization. We are continually working to improve the disambiguation engine to address these issues.
4. If two or more people in your Profiles database share the same first name and last name, then this will lower the publication match probabilities for those people. This logic is defined in the usp_GetPersonInfoForPubMed_xml stored procedure when it calculates the value for the XML tag "LocalDuplicateNames".
5. The disambiguation process includes an optional parameter, "RequireFirstName", which when set to true, will only find seed publications where the author's entire first name (not just the initial) is used. If two or more people in your Profiles database share the same last name and same first name initial, then this parameter is set to true. There are other use cases when you might want to use this option. For example, young investigators (e.g., post-docs) have few publications before 2002, the year when Medline began including author first names. By requiring a first name match for these people, it should have little effect on correct publication matches, but it has the potential to eliminate older publications that might be incorrect matches. To add this or other custom logic to control the RequireFirstName paramenter, modify the code in the usp_GetPersonInfoForPubMed_xml stored procedure.
6. Users or their proxies can manually edit their publication lists within Profiles. The disambiguation engine uses these modifications to improve its search. For example, if a publication was manually added, it will never be automatically removed. If a publication was manually deleted, it will never be automatically re-added. Manually added publications are used as additional seed publications for subsequent calls to the disambiguation engine. In other words, if users need to make corrections to their publication lists, Profiles will learn from this, and it will become more likely that future corrections will not be necessary.

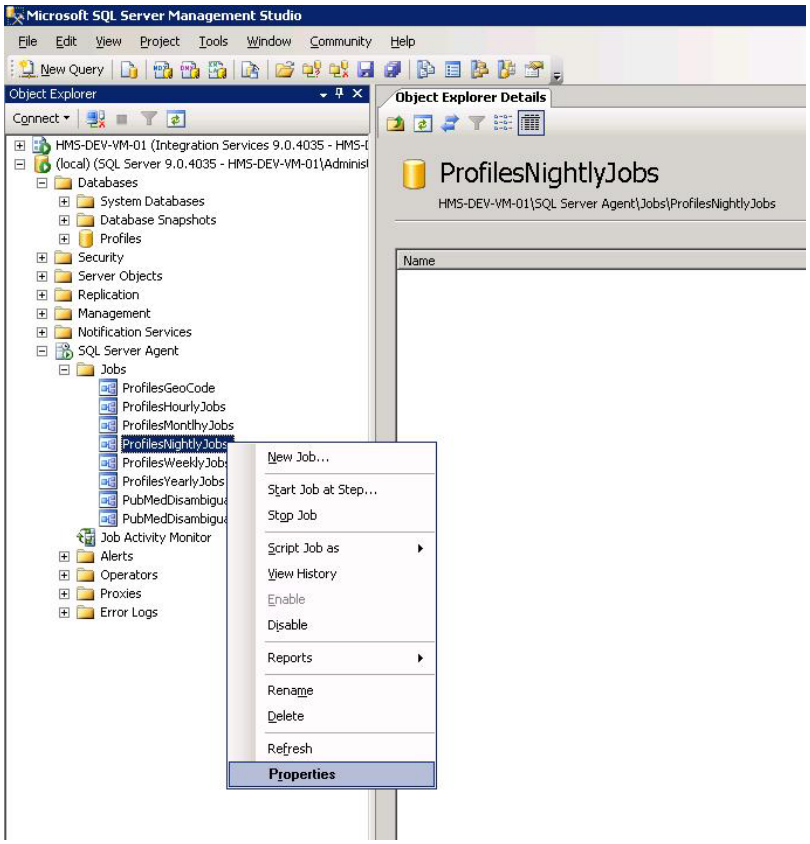
Scheduling Database Jobs

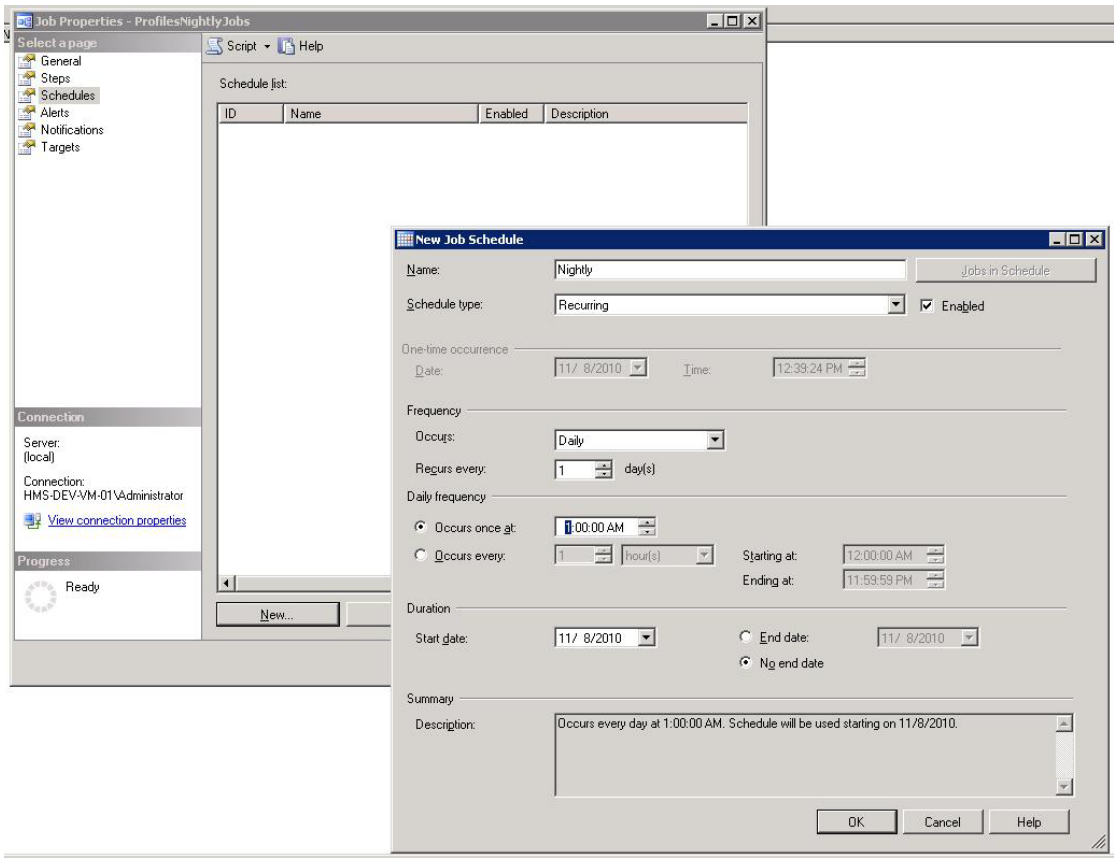
Run ProfilesYearlyJobs.sql manually once after the initial install of Profiles. It will only need to be run again if you update the mesh_* tables. NLM publishes a new version of MeSH once a year.

The following jobs should be scheduled to run regularly:

- ProfilesNightlyJobs.sql
- ProfilesWeeklyJobs.sql
- ProfilesMonthlyJobs.sql
- ProfilesHourlyJobs.sql

The names indicate suggested run frequencies. However, the more often you run these better since they are required to maintain consistency between the content in profiles (e.g., publications) and other aspects of the website, such as the search results and passive networks. ProfilesWeeklyJobs.sql and ProfilesMonthlyJobs.sql should ideally be run whenever ProfilesNightlyJobs.sql is run. The weekly and monthly jobs, though, take significantly longer to run than the nightly job, so it might not be practical to run all of these nightly if your database server is not fast enough.





The ProfilesHourlyJobs.sql runs independently of the others. However, there are dependencies among the remaining jobs, and they should be run in this order: ProfilesNightlyJobs.sql → ProfilesWeeklyJobs.sql → ProfilesMonthlyJobs.sql. Also, if you plan to schedule usp_loadprofilesdata to update your person and user data (see below in the section on updating person data after go-live), then this should be done before ProfilesNightlyJobs.sql is run.

Installing the Website

The Profiles beta website contains two main components that are comprised of a shared layer of developed and Microsoft class libraries: The two main components are the actual website, and a separate application for the Profiles web services. The entire solution should contain the main website (ProfilesWeb) and the web service (Connects.Profiles.Service) with each project referencing the required class libraries.

Solution name:

ResNavPeople

Class Libraries:

(Developed)
Connects.Profiles.Data
Connects.Profiles.BusinessLogic
Connects.Profiles.Common
Connects.Profiles.Utility
Connects.Profiles.Service.ServiceContracts

Connects.Profiles.Service.DataContracts
Connects.Profiles.ServiceImplementation

(Microsoft)
FUA (Ajax file uploader)
Microsoft Enterprise Library

Website:
ProfilesWeb

Web Service:
Connects.Profiles.Service (API and Network Browser Service)

Installation steps:

1. Note that Profiles requires you to download and install the two Microsoft libraries.
    - AspNetAjaxLibraryBeata0911.zip (http://www.codeplex.com/wikipage?ProjectName=AjaxControlToolkit)
    - Microsoft Enterprise Library 4.1 October 2008.msi (http://www.microsoft.com/downloads/details.aspx?FamilyId=1643758B-2986-47F7-B529-3E41584B6CE5&displaylang=en)
2. Load the solution into the IDE, and then perform a release build of the entire solution.
3. Review all messages and outputs of the compiler to ensure all references to required class libraries are in place.
4. Publish Connects.Profiles.Service to its target hosted location.
    - Edit the deployed copy of the web.config file for the correct connection string. Other settings should be left as default.
5. Publish ProfilesWeb to its target hosted location.
    - Edit the deployed copy of the web.config file for the following items:
        - Connection string (connectionStrings.ProfilesDB)
        - AppSettings.URLBase (needs to reflect the full URL of the profiles website)
        - AppSettings.HomepageUrl (needs to reflect the full URL of the default page for the profiles website)
        - AppSettings.ConnectsLoginURL (Login screen for the edit enabled profiles functions)
        - AppSettings.AfterLogoutURL (Redirect page you want your user to be taken to once they have logged out of profiles)
        - AppSettings.DefaultPersonImageURL (The default image that will show in the event a photo of a person does not exist.)
        - AppSettings.ProfileResponseXSD (Path of xml schema if schema validation is required.)
        - AppSettings.GoogleMapCenterLatitude (used to center the default of all network maps)
        - AppSettings.GoogleMapCenterLongitude (used to center the default of all network maps)
        - AppSettings.HideInstitutionSelectionForSearch (used to hide the institution field from the search options)
        - AppSettings.HideDepartmentSelectionFromMiniSearch (used to hide the department option from the mini search options)

- AppSettings.NetworkBrowserService (the full url of the deployed Connects.Profiles.Service.NetworkBrowserService/profiles)
- RoleManager.Providers (Change the application name to the full path of the ProfilesWeb)

Once the web config files are edited correctly, browse to the home page of profiles. If the error page displays in place of the home page, review the server event log for any details that will need to be addressed before proceeding.

Website Authentication Options

The Profiles web site utilizes the membership provider model built into Internet Information Server (IIS) as a means for authentication that is flexible and extensible to many different environments.

By default, Profiles uses basic IIS forms authentication that has been modified to utilize the [user] table in the Profiles database. This functionality is encapsulated in a default provider called "ProfilesDBMembershipProvider" that is defined in a class found in the App_Code folder of the Profiles web site.

Profiles installations that wish to integrate their own existing authentication mechanism can simply build their own C# class that inherits from System.Web.Security.MembershipProvider and include a reference to this provider in the web.config on their site. The settings can be found in the <membership> section of the web.config.

When developing a provider class that inherits from a built-in class, you typically override those methods that you wish to customize and control. Each installation will vary, but in most cases the ValidateUser and FindUserByName methods will need overrides in the custom provider class.

In the case of installations where a Single Sign-On (SSO) solution is desired, the authentication is performed outside of Profiles and what is typically required is a mechanism for handling the "callback" from the SSO system back to Profiles. The requirement here is that the SSO system validate the user and on the return, the core ASP.NET Profile attributes be filled with data from the SSO system. Depending on the SSO system, this information may be available on the QueryString, HTTP header or a cookie. The ASP.NET Profile attributes to be set can be found in the web.config in the <profile> section.

Modifying the Website's Look-And-Feel

The ProfilesWeb site is based on asp.net technology with a presentation layer comprised of .Net Master pages, HTML, CSS, .Net controls and .Net user controls.

- CSS: A folder called CSS exists under the ProfilesWeb project root. It contains 8 css files and one HTC support file for custom styles.
- Images: A folder called Images exists under the ProfilesWeb project root. This contains all image media content for the ProfilesWeb site. Many images are used for borders and navigation in conjunction to CSS styles.
- .Net controls: Pages and user controls will contain one or more controls that deal in the rendering of HTML.

- Master Pages: ProfilesWeb uses two Master Pages to define the layout of the site and each child page. ProfilesPageBase.master creates the layout for the header, footer and main body columns. ProfilesPage.master creates the layout of the details shown in each page (IE the menus,widgets and containers for each child page).

Updating Person Data After Go-Live

If you have frequently changing HR data, you can setup a recurring data load process. Each time you want to update Profiles, enter new data into the loading tables, and then execute the stored procedure usp_loadprofilesdata. For people or users who are already in Profiles, their data will be updated, but their PersonIDs and UserIDs will remain the same. If a person or user no longer appears in the loading tables, then the active flag in the Profiles database will be set to false, and their profiles will no longer be visible.

Proxies

Proxies are users who can permission to edit other people's profiles. There are two types of proxies, "designated" and "default". Designated proxies are people faculty assign to edit their profiles. Access is all or nothing, meaning if you assign someone as a designated proxy, he/she has full access to your profile.

Default proxies are "superusers" who have access to a large number of faculty by default. Default proxies can either be visible or hidden. When faculty go to their proxy page in Profiles, they can see any visible proxies who have access to their profiles. They cannot see the hidden default proxies. Default proxies can be assigned to multiple departments, institutions, or everyone. There are granular editing permissions for default proxies. For example, they can be allowed to edit everyone's publications, but only the photos for a particular department.

Default proxies must be entered directly into the database in the proxies_default table. The "proxy" field in this table is the proxy's Profiles UserID. The department and institution fields can either be empty strings, or a value can be given to either one to limit the population that can be edited by the proxy. The IsHidden field is "Y" if the person is a hidden proxies, and "N" otherwise. The "Edit" fields should be set to "1" if the proxy has permission to edit that content, and "0" otherwise. A single proxy can have multiple records in this table, where each record indicates scope and permissions for a different population of profiles.

Note that all proxies, both designated and default must have a record in the user table.

Software Updates

If you are applying this update to an existing install, we recommend that you do the following:

1. Back up your existing web.config files first, and then re-apply them after you copy the new web code to your server to preserve previous settings.
2. If you have previously made customizations to the code, re-apply those changes to the new code rather than simply replacing the new files with your modified ones.
3. Stop and restart the IIS Application Pool to ensure the cache from the previous version of Profiles has been cleared.

We will be releasing bug fixes as they are identified for the next few months, but no major new features are planned for this current beta codebase. Profiles 1.0, which will replace the beta software, will be a complete rewrite. It will have a more modular architecture, enabling us to include many new features and to begin collaborative development with others who would like to participate in the open source project.

Release Notes

5/5/10 Fixed bug in cache_person update process that did not calculate cache_person.NumPublications correctly.

5/10/10 Fixed bug in usp_LoadProfilesData related to how the department table is populated.

6/20/10 Bug Fixes:

1. The co-author radial network browser does not load.
2. When a collaborator is deleted from the active network details page, it doesn't get removed from the left sidebar.
3. Clicking the link to view all people in one's department in the right sidebar (passive network) returns all people in the entire database.
4. Keyword categories page does not display all categories.
5. Inconsistent behavior across browsers with auto-complete on the main search form. [This feature has been removed for now.]
6. Back link on a person's profile to return to search results is broken.
7. Connection strength information on search results detail page is confusing. [The connection strength value has been removed for now.]
8. Faculty rank, not the affiliation title, is appearing as a person's title.
9. Faculty ranks are hard-coded in the website instead of being dynamically read from the database.
10. usp_loadprofilesdata - departments table isn't populated correctly during the load process.
11. Permissions are not set correctly for the default database application user account.
12. API does not handle secure mode and the isvisible person flag correctly.
13. Pubmed disambiguation process crashes if data is requested for a publication that has been deleted from Medline.
14. Search breaks if API cache tables are not purged. [Clear API cache tables process was added to the nightly database job.]
15. CSS fixes are needed for IE 6.

New Features:

1. Added global error logging to server event log and uniform custom error message display.
2. Added weightcategory tag to keyword XML (for use on keyword cloud page).
3. Added academicrank tag to person XML (a derived person level indicator of that person's highest faculty rank).
4. Created optional script ProfilesSetUserPermissions.sql to grant execute permissions to new database accounts.

Cosmetic:

1. usp_getusersupporthtml - added a wrapper div when edit mode = 1.
2. usp_parsepubmedxml - cleaned up code related to calculating a publication date.
3. fn_getuserdepartmentpeople - cleaned up code to better use cache tables.
4. Changed default database application user account to AppProfilesUser with password Pass1234.

6/25/10 Bug Fixes:

1. ProfilesNightlyJobs.sql fails.
2. View all people in a department returns the wrong list in certain cases.
3. Radial network browser is not working on certain browsers.
4. Google Map visualizations does not display markers for all people.
5. Page layout is incorrect on IE6 and IE7 due to CSS handling differences from other browsers.
6. Number of matching keywords on the Search Results “Why?” Connection page is incorrect for multiple keyword searches.

7/15/10 Bug Fixes:

1. Pagination is not working on the Search Results page when more than 15 items per page is selected.
2. Searches for names or keywords containing an apostrophe (O’Brien, Baker’s Yeast) are not working.
3. IE 6 style conflict is causing Edit Profile page to indent far to the right.
4. Upload photo does not work on the Edit Profile page.
5. List view for Keywords Cloud tab redirects to Co-Authors List View tab.
6. usp_UpdateUnGeocodedAddress fails. [Changed to use addressstring, instead of derived address string.]
7. Exact Keyword search is not functioning as expected for multiple word phrases.
8. Clicking a “most viewed” item on the search page does not run an exact keyword search as expected.
9. The highest overall faculty/academic rank, rather than the rank of the primary affiliation should be displayed.

7/22/10 Bug Fixes:

1. usp_memebership_getpassword assumes that all users records have user.applicationname = ‘Profiles’. This results in login failures for all users.
2. A user who clicks the “Most Viewed” keywords lists in the passive network margin, will trigger double quotes to wrap around the keywords, causing the counts to be incorrect in the search_history_kw table.
3. Trying to upload a photo while editing a profile fails.

8/9/10 Bug Fixes:

1. Dynamic Javascript in ProfilesEdit.aspx.cs is causing Javascript browser errors for IE 6.0 clients viewing profiles from IIS 6.0 servers.
2. Incorrect physical neighbors. [Added latitude and longitude to logic for determining physical neighbors (usp_cache_physical_nieghbors). Added indexes to cache_person for these columns to improve performance.]
3. Changed column/primary key for person_add table to internalusername.

4. Search returned an error if the same query had been done more than 24 hours earlier and the nightly jobs were not run. [Removed a date check in API (usp_getpersonxml_v2), which was used to retrieve cached query results.]
5. Removed mpid foreign key constraint on publications_add and publications_exclude as there is the possibility of null values.
6. API code is not reading IsSecure appSettings key in web.config file.
7. API does not allow search by InternalID.

10/27/10 Bug Fixes:

1. <dataConfiguration defaultDatabase="ProfilesDB"/> in line number 35 of the web.config file was incorrectly listed as <dataConfiguration defaultDatabase="[DB Name Here]"/> [This value needs to be set to ProfilesDB and match the ConnectionString key name of ProfilesDB.]
2. CoAuthorNet.aspx.cs: the inline test for if an address was null or the Latitude equal to zero causes a null exception error in the event address was equal to null. [This was corrected by splitting the test into two 'if'' statements, first test if pList.Person[0].Address != null then inside this condition, test if pList.Person[0].Address.Latitude == 0.]
3. When calling the Profiles API web service, the Response message is not returning the InternalID list. [The bug was in the project Connects.Profiles.Service.DataContracts, file name InternalId.cs, line number 56. The DataContract name was spelled with the incorrect case. The name should be InternalIDList with a capital "ID". The same applies to the DataMember name and Serialization.XMLElementAttribute in this file.]
4. Editing a profile under Secure Socket (HTTPS) causes an error that prevents client side Ajax to execute and page content to display in the EditProfile.aspx page. [Added a secure page function to the ProfilePage.master.cs file to check if the page is under the SSL context.]
5. When you add a designated proxy to your profile and you search by institution, it never returns any records. [changed usp_searchproxy to remove the sub query select of {select name from institution_map where catalystname = @xInstitution} and changed it with {InstitutionFullName = @xInstitution}
6. usp_ProxySearch does not check for isActive status of the user. [Added AND isActive = 1 to the WHERE clause of usp_ProxySearch.

New:

1. vw_person preference defaults changed from Y to N for showphoto, showawards, shownarrative.
2. Added BSD license and Copyright header to all Web and Database code.
3. usp_GetPersonList_xml_V2 – added a join to the display_prefs table. Profiles will now use the display_prefs table directly instead of cache_person to enable user privacy settings to have an immediate effect.
4. usp_cache_sna_betweenness changed so that it is now calculating the exact betweenness score rather than an approximation.
5. usp_parseallpubmedxml proc changed to avoid foreign key conflicts when replacing pubmed data that already exists in publications_include.
6. usp_loadprofilesdata – changed default value of isactive column to 1 (active).

Known Issues:

1. Certain browser security configurations (we haven't determined the exact settings) disable the JavaScript used by the drop down menus on the main search page to select More Options.
2. Selecting "Display Columns:" in the search results by clicking on the column name and not the checkbox fails to trigger a repost of the page.
3. Long item names in Passive Networks (right sidebar) are truncated.
4. The height of the More Options drop down menu on the search page does not dynamically adjust based on the number of items displayed.
5. When two browser windows or tabs are opened at the same time to Profiles, it can create inconsistencies with internal session variables. This can be reproduced by running a search in window #1, then running a different search in window #2, then clicking the "Why?" link in the search results of window #1.
6. When editing a profile in Mac FireFox or Mac Chrome, users cannot add new awards.
7. Network View sometimes fails to load in Mac Opera.
8. Uploading of profile photos on ProfileEdit.aspx causes a javascript/AJAX permission denied error if the client browser is IE 6.0 and the web server is IIS 6.0
9. The PubMed disambiguation process can fail if it is run after users have manually added custom (non-PubMed) publications to their profiles.
10. Rarely, certain PubMed articles cause the disambiguation process to hang or fail.
11. Google maps does not display markers for all people.
12. Hidden proxies display in the list of a profiles known proxy accounts.
13. Login process does not correctly check for the isActive flag when executing the login process of a user.