Profiles Research Networking Software Architecture Guide

Documentation Version: August 5, 2014

Software Version: ProfilesRNS_2.5.0

Table of Contents

Introduction 2
Conceptual Models 3
Profiles, Networks, and Connections 3
Passive & Active Networks 4
Website Architecture 5
Website Framework 5
Applications 6
Modules 7
Database Architecture 10
Schemas Overview 10
Core Schemas 10
Extended Schemas 11
Core Database Objects 11
Security Groups 15
Node and Triple Tables 16
Data Flow 16
Ontology Schema Tables 18
Loading Data Using ProcessDataMap 21
Loading Data Using ProcessTriples 22
Extending Profiles RNS 23

Introduction

This document contains three parts:

1) Conceptual Models – Both the Profiles RNS website and database are organized around of concept of Profiles, Networks, and Connections, which build on the triple structure of RDF. Networks are further divided into ones that are automatically derived (Passive) and those which users create themselves (Active). These conceptual models provide the rationale for much of the architectural design of Profiles RNS.

2) Website Architecture – This section provides a basic overview of how the website Framework is designed. It also describes Applications, which extend the Framework, and Modules, which the Applications use to provide specific functionality.

3) Database Architecture – As an ontology-based system, the database plays a greater role in Profiles RNS than in most software. The database includes not only the RDF triple store, but also the ontology, data feeds, and business logic. This section of this guide describes the key components of the database and concludes with a brief tutorial.

For information about the VIVO Ontology, which Profiles RNS uses to encode data in RDF, see the ProfilesRNS_APIGuide.pdf file.

Conceptual Models

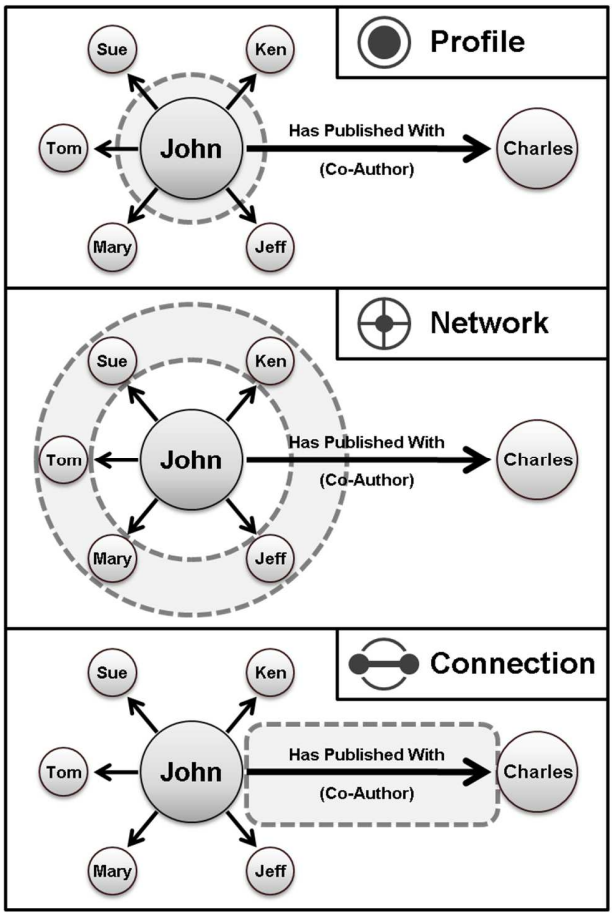
Profiles RNS pioneered two conceptual models, “Profiles-Networks-Connections” and “Passive & Active Networks”, which make the software unique among research networking platforms by (1) providing three ways of viewing and exploring RDF data, and (2) providing two ways of generating new triples.

Profiles, Networks, and Connections

Profiles RNS introduces a novel concept called Profiles, Networks, and Connections. Consider the RDF triple in which “John”-“has published with”-“Charles”. In other words, John and Charles are co-authors. By itself, this is a simple fact, but for a user of the Profiles website, it leads to three types of more complex questions:

1. Who is John? To answer this question, Profiles contains “profile” pages dedicated to each entity, which lists its various RDF properties. John’s profile page might include properties such as his name, title, affiliation, contact information, photograph, and a research narrative.
2. Who are John’s co-authors? This question explores one of John’s RDF properties, “has published with”, in more depth. A profiles “network” page lists an entity and all the other entities that are connected to it through a particular property, along with additional information about those connections. In other words, it displays all RDF triples that have a given subject and predicate. There are many ways to present a network to users, depending on exactly what they want to know about that network. For example, a geospatial visualization of the network can show whether John’s co-authors are mostly located in one city or spread across different geographical regions, and a temporal view of John’s co-authors show how his collaborations have changed over time.
3. How are John and Charles connected? This question is about the particular co-authorship relation between John and Charles. How many articles have they published together? When were these articles published? Who was the first author on those articles? What were the topics of those articles? Profiles contains “connection” pages, which enable users to view any metadata associated with a single RDF triple. This is especially useful for Profiles’ derived “passive” networks. For example, Profiles automatically creates a “similar research” connection between investigators if their publications have a certain number of Medical Subject Headings (MeSH) in common. The connection page lists those subject headings.

The diagram below illustrates the differences between Profiles, Networks, and Connections. On the Profiles website, the three circular icons indicate to users which of the three types of pages they are viewing. The Profile icon has a large center circle with a thin outer ring around it. This represents the fact that the page is primarily about the entity, with only a sample of the surrounding networks being shown. The Network icon has a small center circle with lines connecting it to the outer ring. This represents the fact that the page is more about the surrounding entities. The Connection icon has two circles connected by a thick line to show that the page is about a particular triple.



Passive & Active Networks

"Passive" and "Active" networks build upon the raw RDF data loaded into Profiles by creating new types of triples. This not only enables the website to be a more useful and exciting tool on day one, but it also allows users to expand its content with information about social networks that only they know.

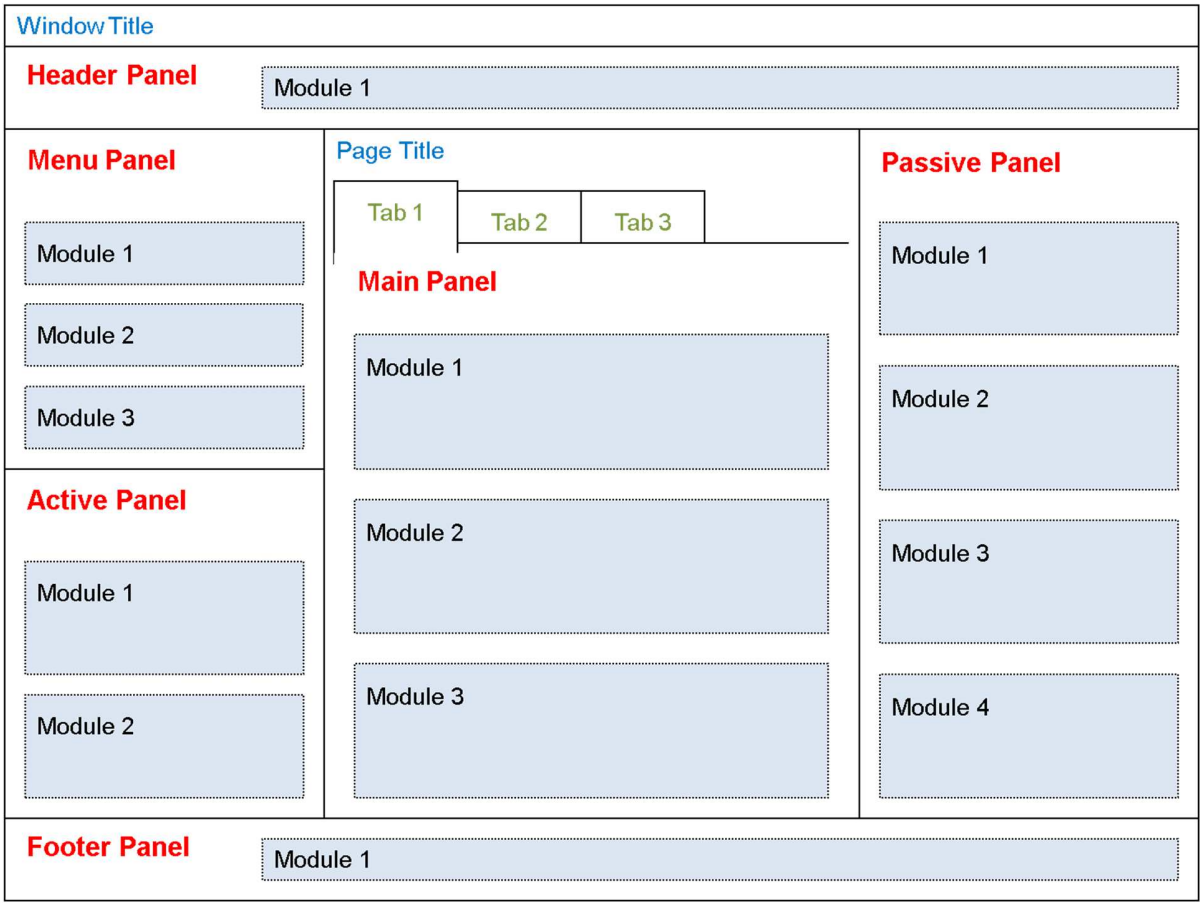
- Passive networks are automatically derived from existing RDF data, such as co-authorship history, organizational relationships and geographic proximity. It extends these networks by discovering new connections, such as identifying "similar people" who share related keywords. Offering these additional suggestions, Profiles RNS can lead users to unexpected opportunities for collaboration and new sources of expertise.

- Users can manually create active networks by identifying advisor, mentor and collaborator relationships with colleagues. Profiles RNS will soon support the OpenSocial standard, which will let researchers use the same types of plug-in collaboration gadgets found on LinkedIn and Google within their active networks.

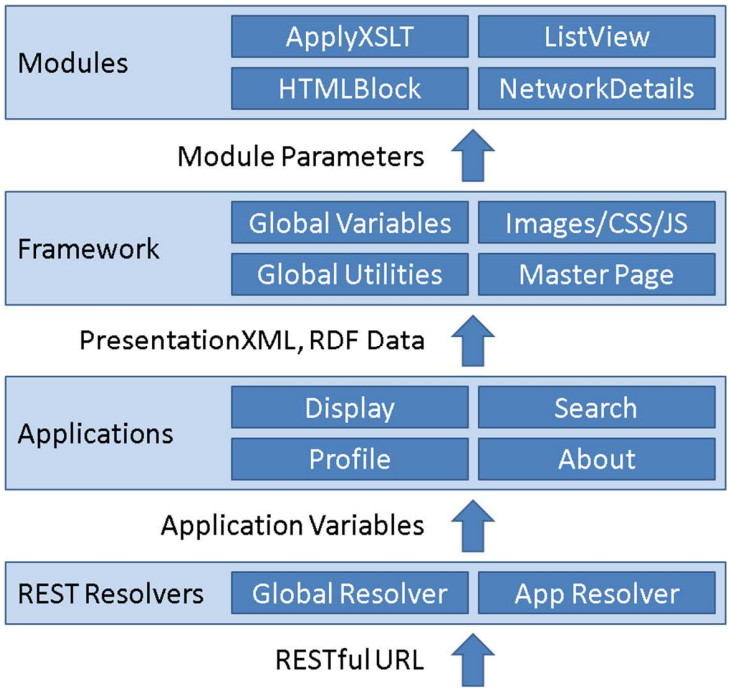
Website Architecture

Website Framework

The website's graphical user interface (GUI) is divided into sections called "panels". Each panel contains a list of "modules". Modules are implemented as .NET controls. Most modules function by requesting RDF data and applying an XSLT file to render it as HTML. However, some perform more complex tasks. The website "Framework" creates the HTML "shell" illustrated below. The shell provides the page layout for the panels. The Framework uses the data in an XML file called the PresentationXML to determine the window and page titles and which modules to load into each panel.



Below is a box diagram of the website components. When the server receives a RESTful URL, a set of "resolvers" parse the query string path to determine which application is being requested and what variables should be passed to the application. Each application performs different functions. The Profile application takes a URI and returns an RDF document. The Display application takes a URI and renders it as HTML. Applications that have a user interface component, such as Display, Search, and About send the Framework an application-specific PresentationXML and an optional RDF Data object. The Framework's Master Page creates the HTML shell, which includes images, CSS, and JavaScript files. The Framework uses the data in the PresentationXML to select which modules to load, and it passes parameters listed in the PresentationXML to the modules.



Modules typically obtain data by performing an HTTP POST to a URI that returns an RDF document. That URI can exist anywhere on the internet. If the URI happens to be in Profiles, then the HTTP POST will pass through the REST Resolver and be handled by the Profile application. Thus, the Display application usually calls the Profile application multiple times indirectly via the Modules that it tells the Framework to render.

When a user enters a Profiles URI into the web browser, the following events occur:

1. A 303 redirect sends the user to a URL that corresponds to the Display application.
2. The Display application retrieves a PresentationXML specific to the RDF class of the URI from the Profiles database and sends it to the Framework.
3. The Framework parses the PresentationXML and loads the appropriate modules.
4. The modules request RDF data via an HTTP POST to a URI. The request header will contain an “application/rdf+xml” content type.
5. If it is a Profiles URI, then Profiles will recognize the “application/rdf+xml” content type in the request header and perform a 303 redirect to a URL that corresponds to the Profile application.
6. The Profile application will first see if the RDF data for the requested URI exists in its cache. If so, it will return the data from cache. Otherwise, it will retrieve the RDF from the Profiles database and store it in cache before returning it.
7. The modules apply XSLT to the RDF to render HTML.

Applications

Below is a list of default applications in Profiles RNS. The source code for a new application should be placed within its own folder, and the application must be “registered” in the ApplicationModuleCatalogue.xml file.

Name	Description
Profile	Returns the RDF document for a URI.
Display	Renders a URI as HTML.
Search	Search identifies all RDF nodes that have a property whose value matches a search string. It displays a list of those nodes and links to their URIs. Faceting allows users to narrow the search results by type (class group) or subtype (class). Any property can be used to sort search results. Search incorporates stemming (to match different parts of speech), removal of stop words (e.g., “the”, “of”), and term expansion through the use of a thesaurus (e.g., “cancer” -> “neoplasm”).
About	Displays general information about the Profiles RNS website.
SPARQL	This is an interface to test the Profiles RNS SemWeb SPARQL engine. Users can enter an arbitrary SPARQL query and view the results. In the final Profiles RNS 1.0.0 release, this front-end tool will only be available to administrators by default, though the ability to pass SPARQL queries to the SemWeb web services can remain open to the public.
Edit	This application allows users to manage the content on their profiles.
Direct	Direct2Experts is a federated search tool that locates experts across multiple institutions using Profiles RNS and other research networking products.

Modules

Below is a list of default modules in Profiles RNS. The source code for a new module should be placed within its own folder within an application’s modules folder, and the module must be “registered” in the application’s modules/ModuleCatalogue.xml file.

Application	Module	Description
ActiveNetwork	MyNetwork.ascx	Displays the active network of a user.
DIRECT	DirectSearch.ascx	Federated search across multiple institutions using the Direct2Experts network.
Edit	CustomEditAwardOrHonor.ascx	Custom edit module that simplifies creating or updating an AwardOrHonor.
Edit	CustomEditMailingAddress.ascx	Custom edit module that simplifies creating or updating a Mailing Address.
Edit	CustomEditMainImage	Custom edit module that simplifies creating or updating a MainImage.
Edit	CustomEditAuthorInAuthorship	Custom edit module that simplifies creating or updating an AuthorInAuthorship.
Edit	EditDataTypeProperty.ascx	Default module for editing a DataType property.
Edit	EditObjectTypeProperty.ascx	Default module for editing an ObjectType property.
Edit	EditPropertyContainer.ascx	This is a wrapper for all modules on the profile editing pages.
Edit	EditPropertyList.ascx	Displays a list of properties on a profile that a user can edit.
Edit	EditVisibilityOnly.ascx	Enables a user to change the visibility settings of a property, but not the value of the property.
Error	ErrorMessage.ascx	Displays the global error message from the global.asax.cs file.
Framework	ApplyXSLT.ascx	Applies a specified XSLT file to RDF data.
Framework	HelloWorld.ascx	Displays a "Hello World" message to help with development/debugging.
Framework	HTMLBlock.ascx	Displays a specified block of HTML.
Framework	MainMenu.ascx	Displays the main menu on the left sidebar.

Login	Login.ascx	Profiles login module.
Profile	CustomViewPersonGeneralInfo.ascx	Custom display module that shows a person's name, phone, fax, email, mailing address, and main image.
Profile	CustomViewAuthorInAuthorship	Custom display module for the authorInAuthorship property.
Profile	CustomViewPersonSameDepartment.ascx	Custom display module that lists people in the same department as the person whose profile is being viewed.
Profile	CustomViewInformationResource	Custom display module for an InformationResource profile.
Profile	NetworkCategories.ascx	Displays a network by grouping items according to category names.
Profile	NetworkDetails.ascx	Displays a network listing each item with additional summary information.
Profile	NetworkList.ascx	Displays a network listing each item in 1, 2 or 3 columns with an enumerated or bulleted style.
Profile	NetworkMap.ascx	Displays a network by indicating the location of items on a Google map.
Profile	NetworkRadial.ascx	Displays a network using an egocentric radial graph (e.g., person in the center surrounded by coauthors).
Profile	PassiveHeader.ascx	Displays instructional text at the top of the right sidebar.
Profile	PassiveList.ascx	Displays a list of items in the right sidebar.
Profile	PassiveText.ascx	Displays a block of text in the right sidebar.
Profile	PropertyList.ascx	Default view of properties on a profile page.
Proxy	ManageProxies.ascx	Enables a user to view/change the people who have proxy permissions to edit his or her profile(s).
Proxy	SearchProxies.ascx	Enables a user to search for people who can be added as designated proxies.
Search	MiniSearch.ascx	Mini search module at the top of the left sidebar.
Search	SearchConnection.ascx	Displays the details of why an item was returned by a keyword search.
Search	SearchCriteria.ascx	Displays the search criteria on the right sidebar of a search results page.
Search	SearchEverything.ascx	Search for Everything form.
Search	SearchEverythingFacets.ascx	Displays drill down options on the search results page of a Search Everything search.
Search	SearchOptions.ascx	Displays additional search options on the search results page.
Search	SearchPerson.ascx	Search for People form.
Search	SearchResults.ascx	Displays the results of a Person Search and an Everything Search.
Search	TopSearchPhrase.ascx	Displays the most commonly used search terms over a day, week, or month period.
SPARQL	SPARQLSearch.ascx	Enables administrators to run a SPARQL search.

We try to minimize the amount of C# code in Profiles RNS. The complexity of how to store and process RDF exists in the database, and the details of how to render a page are coded as PresentationXML and XSLT files. As a result, little or no C# programming should be needed to

configure and customize Profiles. Extending the functionality of Profiles can be done in several ways: (1) adding new classes or properties to the ontology, (2) editing the PresentationXML files for existing applications, (3) creating a new application, or (4) creating a new module.

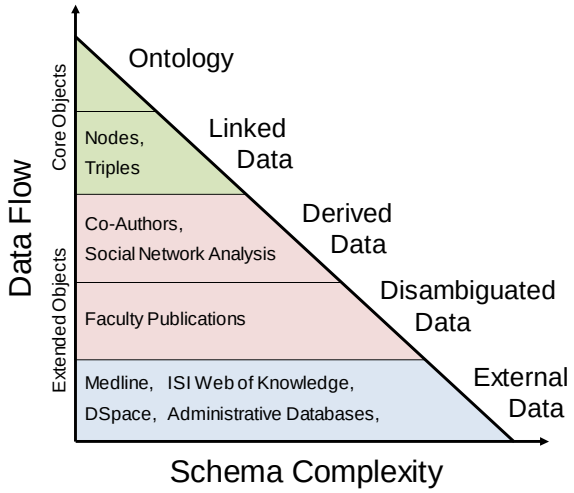
Database Architecture

Schemas Overview

The Profiles database is organized in a hierarchy, with the top two levels defined by schemas. We use a convention where each schema name has two parts, separated by a period. Note that because of the period, the schema name must always be written in brackets. For example, the table [ProfilesRNS].[RDF.Stage].[Log.Job] has the table name "[Log.Job]", it is in the schema "[RDF.Stage]", and it is in the database "[ProfilesRNS]". When writing queries, you do not need to include the database name if you are already connected to the database, but you will always need to include the schema name.

The schemas are divided into two categories: (1) The "core" schemas are essential for Profiles to function properly. Every instance of Profiles must contain the database components in these schemas. (2) The "extended" schemas facilitate the data loading process or provide other types of functionality that are specific to a domain. For example, there are tables to store attributes associated with Pubmed articles; however, these are only relevant in biomedicine. A financial company using Profiles to build networks among its global employees might create different extended schema tables that store information about countries and markets.

There is a general direction of data flow in Profiles, as illustrated in the diagram below, which begins with external data from many different sources. This is then parsed and disambiguated to identify distinct objects and their relationships. Derived data may be obtained from the disambiguated data through computational methods such as social network analysis. Finally, disambiguated and derived data is described as linked data, which consists of nodes and triples. An ontology describes the classes and properties in the linked data. At each step in the data flow, there is a reduction in the number and complexity of database objects, as the model used to represent the data becomes simpler and more generalizable.



Core Schemas

Schema	Description
[Framework.]	Handles global functions, such as resolving RESTful URLs and managing

	scheduled jobs.
[Ontology.]	Contains the semantic web ontology used by the website.
[Ontology.Import]	Contains tools to import and process OWL files.
[Ontology.Presentation]	Contains the "presentation" ontology, which describes how content should be displayed on the website.
[RDF.]	Contains the RDF nodes and triples specific to an instance of Profiles.
[RDF.Security]	Contains information about who can access secure/private nodes and triples.
[RDF.SemWeb]	Used to format [RDF.] data so that it can be used by the SemWeb SPARQL engine.
[RDF.Stage]	Used by the bulk data loading process to store temporary data before it is loaded into the [RDF.] tables.
[User.Account]	Contains information about authorized users of the website.
[User.Session]	Contains information about website sessions. A public user of the website will have a session even if she has not logged in and linked the session to a specific user account.
[Utility.Application]	Contains functions and procedures that are used in a variety of contexts.
[Utility.Math]	Contains mathematical lookup tables and functions.
[Utility.NLP]	Contains lookup tables and functions related to support natural language processing for search and other features.

Extended Schemas

The Profiles website is structured as a collection of “applications”. In the extended schemas, the first part of the schema name corresponds to the primary application that uses it, such as “Profile”. Below are the default extended schemas included in Profiles.

Schema	Description
[Direct.*]	Supports Direct2Experts functionality--federated search across multiple institutions using Profiles and other research networking products.
[Edit.*]	Allows users to edit profile content.
[Login.*]	Allows users to login to the website.
[Profile.Cache]	Contains the results of bibliometric and social network analyses.
[Profile.Data]	Stores copies of certain types of RDF data in relational tables to help with data loads or to improve performance of particular kinds of queries.
[Profile.Framework]	Used by the Profile application to interact with the Framework.
[Profile.Import]	Used to place person and other types of data during an initial load of Profiles RNS and in subsequent updates.
[Search.]	Provides basic search functionality for Profiles RNS.
[Search.Cache]	Improves the performance of the Profiles RNS search tool by pre-processing the RDF data through scheduled jobs.
[Search.Framework]	Used by the Search application to interact with the Framework.

Core Database Objects

Below are selected objects within the core database schema. The object Types are table (T), view (V), stored procedure (P), and function (F). The Uses are: Administrative (A) objects are used during the initial software installation, modifying the ontology, or debugging. They are not used during normal operation of the website. Job (J) objects are used as part of scheduled processes to load data into Profiles RNS or to analyze existing data (e.g., update search

cache). Web (W) objects are called directly by the .NET code. Helper (H) objects are used by other objects, but are generally not called directly.

Object	Type	Use	Description
[Framework.].[InstallData]	T	A	Used for bulk import of ontology data during the installation process.
[Framework.].[Job]	T	J	Lists steps that are executed during scheduled data updates.
[Framework.].[JobGroup]	T	J	Describes related sets of jobs.
[Framework.].[Log.Job]	T	J	Keeps a log of each time a step in the [Ontology.].[Job] table is run.
[Framework.].[Parameter]	T	W	Global parameters used by Profiles.
[Framework.].[RestPath]	T	H	Lists URL prefixes that should be treated as RESTful paths and contains pointers to optional stored procedures that can map paths to actual files.
[Framework.].[vwDatabaseCode]	V	A	Lists the code for stored procedures and functions.
[Framework.].[vwDatabaseObjects]	V	A	Lists all database objects.
[Framework.].[ChangeBaseURI]	P	A	Facilitates moving Profiles from one environment to another by changing the base URI path throughout the database.
[Framework.].[CreateInstallData]	P	A	Combines all non-OWL ontology data into a single XML object. This is used only when building a Profiles install package.
[Framework.].[LICENCE]	P	A	Contains the open source license for the software.
[Framework.].[LoadInstallData]	P	A	Parses the InstallData.xml file and populates all non-OWL ontology tables.
[Framework.].[LoadXMLFile]	P	A	Imports data from an external file into a specified table and column.
[Framework.].[ResolveURL]	P	W	Maps RESTful URLs to their actual file names.
[Framework.].[RunJobGroup]	P	J	Runs a series of data load or update steps.
[Ontology.].[ClassGroup]	T	H	Lists top-level RDF Class Groups used in search and browse.
[Ontology.].[ClassGroupClass]	T	H	Maps Class Groups to individual RDF Classes.
[Ontology.].[ClassProperty]	T	H	Defines which RDF properties should be returned and expanded when data is requested.
[Ontology.].[ClassTreeDepth]	T	H	Contains the class hierarchy. Used by Search.
[Ontology.].[DataMap]	T	J	Maps extended schema data to the ontology.
[Ontology.].[Namespace]	T	W	Lists namespaces and their prefixes.
[Ontology.].[PropertyGroup]	T	H	Lists the broad groups of related properties.
[Ontology.].[PropertyGroupProperty]	T	H	Lists the properties within each group.
[Ontology.].[vwMissingClassProperty]	V	A	Lists items in the ontology not represented in the ClassProperty table.
[Ontology.].[vwMissingPropertyGroup Property]	V	A	Lists items in the ontology not represented in the PropertyGroupProperty table.
[Ontology.].[vwPropertyTall]	V	A	Summarizes properties in [Ontology.].[Triple].
[Ontology.].[vwPropertyWide]	V	A	Summarizes properties in [Ontology.].[Triple].
[Ontology.].[AddProperty]	P	A	Adds a new property to the ontology tables.
[Ontology.].[CleanUp]	P	A	Helps with populating the ontology tables.
[Ontology.].[GetClassCounts]	P	H	Returns the number of nodes associated with Class Groups and their classes. This is used primarily for search and browse in the website.

[Ontology.].[UpdateCounts]	P	J	Updates ontology tables with counts from the [RDF.] tables.
[Ontology.].[UpdateDerivedFields]	P	A	After the Ontology is loaded into the [RDF.] tables, this procedure updates [Ontology.] tables with the NodeIDs that were created during the loading process.
[Ontology.Import].[OWL]	T	A	Contains RDF ontologies in OWL XML format.
[Ontology.Import].[Triple]	T	A	Presents OWL data as triples.
[Ontology.Import].[vwOwlTriple]	V	A	Extracts triples from [Ontology.].[OWL].
[Ontology.Import].[ConvertOWL2Triple]	P	A	Parses the [Ontology.].[OWL] data and stores the triples in [Ontology.].[Triple].
[Ontology.Import].[ConvertTriple2OWL]	P	A	Creates a record in the [Ontology.].[OWL] table by combining triples in [Ontology.].[Triple]. This is used only when building a Profiles install package.
[Ontology.Import].[fnGetClassTree]	F	A	Returns a hierarchical list of classes defined in [Ontology.].[Triple].
[Ontology.Import].[fnGetPropertyTree]	F	A	Returns a hierarchical list of properties defined in [Ontology.].[Triple].
[Ontology.Presentation].[General]	T	A	A temporary table used to help with editing PresentationXML.
[Ontology.Presentation].[Panel]	T	A	A temporary table used to help with editing PresentationXML.
[Ontology.Presentation].[XML]	T	H	Templates for how different types of profiles, networks, and connections should be displayed on the website.
[Ontology.Presentation].[ConvertTables2XML]	P	A	Creates PresentationXML from data in temporary tables.
[Ontology.Presentation].[ConvertXML2Tables]	P	A	Parses data in PresentationXML to temporary tables.
[RDF.].[Alias]	T	H	Lists alternative names for particular nodes.
[RDF.].[Node]	T	H	Lists RDF nodes.
[RDF.].[Triple]	T	H	Lists RDF triples.
[RDF.].[vwClass]	V	A	Summarizes classes from data in [RDF.].[Triple].
[RDF.].[vwPropertyTall]	V	A	Summarizes properties from data in [RDF.].[Triple].
[RDF.].[vwPropertyWide]	V	A	Summarizes properties from data in [RDF.].[Triple].
[RDF.].[vwTripleValue]	V	A	Lists the node values of the subject, predicate, and object of the triples in [RDF.].[Triple].
[RDF.].[DeleteNode]	P	W	Deletes a node and its associated triples.
[RDF.].[DeleteTriple]	P	W	Deletes a triple and its dependencies.
[RDF.].[GetDataRDF]	P	W	Returns RDF/XML for a profile, network, or connection.
[RDF.].[GetPresentationXML]	P	W	Returns the PresentationXML template associated with a profile, network, or connection.
[RDF.].[GetPropertyList]	P	W	Combines DataRDF and PresentationXML.
[RDF.].[GetPropertyRangeList]	P	W	Returns classes that can be linked to by a property.
[RDF.].[GetStoreNode]	P	W	Creates or updates a single node.
[RDF.].[GetStoreTriple]	P	W	Creates or updates a single triple.
[RDF.].[SetNodePropertySecurity]	P	W	Changes the security group for a single node and property.

[RDF.].[fnTripleHash]	F	H	Returns the SHA1 hash of a triple.
[RDF.].[fnURI2NodeID]	F	H	Returns the NodeID of a URI.
[RDF.].[fnValueHash]	F	H	Returns the SHA1 hash of the language, data type, and value of a node.
[RDF.Security].[Group]	T	H	Lists Security Groups, which are groups of people with certain access rights.
[RDF.Security].[Member]	T	H	Lists the users that belong to a group.
[RDF.Security].[NodeProperty]	T	H	Lists custom security groups for a property of a specific node.
[RDF.Security].[GetSessionSecurityGroup]	P	H	Returns the primary security groups of a user.
[RDF.Security].[GetSessionSecurityGroupNodes]	P	H	Returns all the individual security groups to which a user has been assigned.
[RDF.Stage].[InternalNodeMap]	T	J	Used to map RDF NodeIDs to IDs used in the extended schema tables.
[RDF.Stage].[Log.DataMap]	T	J	Keeps a log of each time the [Ontology.].[DataMap] table is used to convert extended schema data into RDF.
[RDF.Stage].[Log.Triple]	T	J	Keeps a log of each time data in the [RDF.Stage].[Triple] table is converted to RDF.
[RDF.Stage].[Triple.Map]	T	J	Keeps a record of which TripleIDs were created from StageTripleIDs.
[RDF.Stage].[Triple]	T	J	Used as a temporary table to store information about triples before they are copied to the [RDF.] tables.
[RDF.Stage].[LoadAliases]	P	J	Populates the [RDF.].Alias table.
[RDF.Stage].[LoadTriplesFromOntology]	P	A	Loads data from the ontology into [RDF.Stage].[Triple].
[RDF.Stage].[ProcessDataMap]	P	J	Creates nodes and triples from extended schema data.
[RDF.Stage].[ProcessTriples]	P	J	Loads data from [RDF.Stage].[Triple] into [RDF.].[Node] and [RDF.].[Triple], creating new nodes and triples as needed.
[Search.].[History.Phrase]	T	H	Stores a history of the keyword phrases that were searched.
[Search.].[History.Query]	T	H	Stores a history of the full queries that were searched.
[Search.].[GetConnection]	P	W	Returns the details of why a node matched a search query.
[Search.].[GetNodes]	P	W	Search for nodes based on keywords and other parameters.
[Search.].[GetTopSearchPhrase]	P	W	Determines which search phrases were used most often.
[Search.].[LookupNodes]	P	H	A simplified version of the search algorithm, which does not rely on the cache tables.
[Search.].[ParseSearchString]	P	H	Extracts phrases from the keyword search string entered by the user.
[Search.Cache].[History.TopSearchPhrase]	T	J	Lists the most commonly used search phrases.
[Search.Cache].[History.UpdateTopSearchPhrase]	P	J	Determines the most commonly used search phrases.
[Search.Framework].[ResolveURL]	P	H	Parses parameters out of a RESTful search URL.
[User.Account].[DefaultProxy]	T	H	Lists users who can edit other user's content based on affiliation.

[User.Account].[DesignatedProxy]	T	H	Lists users who have been designated by other users to edit their content.
[User.Account].[Relationship]	T	H	Lists “active network” relationships.
[User.Account].[User]	T	H	Lists authorized users of Profiles.
[User.Account].[Authenticate]	P	W	Provides default user authentication.
[User.Account].[Proxy.AddDesignatedProxy]	P	W	Adds a designated proxy to a user.
[User.Account].[Proxy.DeleteDesignatedProxy]	P	W	Deletes a designated proxy from a user.
[User.Account].[Proxy.GetProxies]	P	W	Gets a list of proxies related to a user.
[User.Account].[Proxy.Search]	P	W	Searches for users who can be added as designated proxies.
[User.Account].[Relationship.GetRelationship]	P	W	Returns a user’s active network.
[User.Account].[Relationship.SetRelationship]	P	W	Stores a new active network connection.
[User.Session].[Bot]	T	H	Contains a list of UserAgents known to be web crawlers.
[User.Session].[History.ResolveURL]	T	H	Contains a log of pages accessed during a session.
[User.Session].[Session]	T	H	Contains basic information about each website session.
[User.Session].[CreateSession]	P	W	Creates a new session.
[User.Session].[UpdateSession]	P	W	Updates the data associated with an existing session.

Security Groups

View and edit permissions in Profiles RNS are managed through “security groups”. A security group is bigint number. Security groups with negative values correspond to user roles (e.g., “public” or “admin”). Security groups with positive values correspond to specific users. A given user can be linked to multiple security groups. Every node and triple has a ViewSecurityGroup. A user must be linked to that security group in order to view the node or triple. Every node also has an EditSecurityGroup, which defines who can make changes to the properties of that node. Triples do not have EditSecurityGroups—triples can be edited if the user is linked to the EditSecurityGroup of the subject of the triple.

A user who is not logged into the site has a role of “public”, which is security group -1. A logged in “user” is security group -20, and an “admin” is security group -50. Every role has access to everything a role with a higher number has. For example, admins can see and edit everything users can, and users can see everything the public can. Therefore, a “curator”, which has security group -40, can do everything a user can, but not necessarily everything an admin can.

In Profiles RNS, a user is not the same as a profile. A user might own multiple profiles. For example, a department chair might manage both a person profile for herself as well as an organization profile for her department. Every user account has a unique URI, though this account profile is not visible on the website. What are visible are the profiles of the URIs that the user manages. This is important for understanding how security groups work. Suppose Mary Smith’s user account corresponds to NodeID 1234, but her online profile corresponds to NodeID 5678. In order for Mary Smith to edit her own profile, the EditSecurityGroup of NodeID 5678

must be 1234. If you want to give Mary Smith additional editing rights to a different profile, then again, the user's NodeID, not the user's profile's NodeID, should be the EditSecurityGroup.

No user has access to security group 0. That is reserved for "deleted" items that cannot be physically deleted from the database. In other words, when a node or triple is given a ViewSecurityGroup of 0, then it can no longer be accessed through the website.

Node and Triple Tables

The [RDF.].[Node] and [RDF.].[Triple] tables store the RDF data in Profiles RNS. The data for every profile in the website comes from these tables, and it is used by the SemWeb API for SPARQL queries.

The unique identifier in the [RDF.].[Node] table is the NodeID. Each node has a Value and optional Language and DataType. The [RDF.].[fnValueHash] function combines the Value, Language, and DataType into a binary(20) value stored in the ValueHash column. The ValueHash is used by SemWeb in SPARQL queries. A ValueHash cannot be repeated. In other words, the combination Value/Language/DataType must be unique in [RDF.].[Node]. The ObjectType = 0 if the node is an entity, and ObjectType = 1 if the node is a literal. The value of an entity is a URI. The ViewSecurityGroup and EditSecurityGroup indicate who has permission to view and edit the node. The InternalNodeMapID is a reference to a corresponding record in the [RDF.Stage].[InternalNodeMap] table, which is described in more detail in the Loading Data Using ProcessDataMap section.

The unique identifier in the [RDF.].[Triple] table is the TripleID. Each triple has a Subject, Predicate, and Object values, which correspond to NodeIDs. The [RDF.].[fnTripleHash] function combines the Subject, Predicate, and Object into a binary(20) value stored in the TripleHash column. A TripleHash cannot be repeated. In other words, the combination Subject/Predicate/Object must be unique in [RDF.].[Triple]. The ObjectType indicates whether the object of the triple is an entity (0) or literal (1). The ViewSecurityGroup indicates who has permission to view the triple. There is no EditSecurityGroup in this table. The EditSecurityGroup of the Subject node determines who can edit the triple. The Graph value groups triples that were created as part of the same process, such as importing an OWL file. Every triple has a Weight, whose value is a float between 0 and 1, which indicates the "strength" of the triple, or the probability that the triple is correct. The concept of a triple weight is not part of RDF. It is a Profiles RNS extension of RDF, which is used for automatically derived triples (e.g., passive networks). The idea is that there can be some uncertainty about whether the relationship described by a derived triple really exists, and the weight reflects that uncertainty. Profiles RNS also adds a SortOrder value, which determines the default order in which the triples for a given subject-property combination are listed in various places in the website. The SortOrder values are sequential integers starting at 1. The Reitification value links a triple to a node whose properties provide additional information about the triple, such as provenance.

Data Flow

Profiles RNS is designed to be a dynamic website, with multiple data feeds regularly updating the system. There are several different processes that load and update data. Understanding the direction of data flow and the entry points is essential for adding new types of data.

1) Loading person data from an external (e.g., HR) source.

   [Profile.Import] → [Profile.Data] → [Profile.Cache] → [RDF.]

   Person data such as names, contact information, and affiliations are loaded into the [Profile.Import] tables. The [Profiles.Import].[LoadProfilesData] procedure uses this to update the [Profile.Data] tables. The Nightly/Weekly/Monthly jobs, as defined in the [Framework.].[JobGroup] and [Framework.].[Job] tables, as well as the geocoding and author disambiguation services, then populate the [Profile.Cache] tables. The [Profile.Cache].[Process.Audit] table stores a log of the steps that were run to update the [Profile.Cache] tables. The [RDF.Stage].[ProcessDataMap] procedure uses the [Ontology.].[DataMap] table to map selected items in the [Profile.Data] and [Profile.Cache] tables to the ontology and to generate RDF nodes and triples in the [RDF.] tables.

2) Loading user account data from an external source.

   [Profile.Import] → [User.Account] → [RDF.]

   Information about users is also loaded into [Profile.Import]. The [Profiles.Import].[LoadPersonData] process uses this to update the [User.Account] tables. The [RDF.Stage].[ProcessDataMap] procedure generates RDF nodes and triples for this data.

3) Creating RDF data from an extended data table.

   [Profile.Data] → [RDF.]

   When editing profiles, some data is stored directly into the [RDF.] tables as nodes and triples. However, other types, such as publications, are first stored in [Profile.Data] tables and then copied to the [RDF.] tables using the [Profiles.Stage].[ProcessDataMap] procedure.

4) Loading data as triples.

   [RDF.Stage] → [RDF.]

   Data can be loaded as triples (subject URI, predicate URI, and object URI or literal value) into the [RDF.Stage].[Triple] table and then loaded into the [RDF.] tables using the [RDF.Stage].[ProcessTriples] procedure.

5) Adding new classes or properties to the ontology.

   [Ontology.Import] → [Ontology.] → [RDF.Stage] → [RDF.]

   OWL files are loaded into the [Ontology.Import].[OWL] table. The [Ontology.Import].[ConvertOWL2Triple] procedure parses the OWL file and stores triples in the [Ontology.Import].[Triple] table. The [RDF.Stage].[LoadTriplesFromOntology] procedure copies this data into the [RDF.Stage].[Triple] table, and then the [RDF.Stage].[ProcessTriples] procedure loads this into the [RDF.] tables.

6) Presenting the RDF data in a format that can be used by SemWeb (SPARQL).

   [RDF.] → [RDF.SemWeb]

   This is an automatic transformation that occurs through [RDF.SemWeb] views.

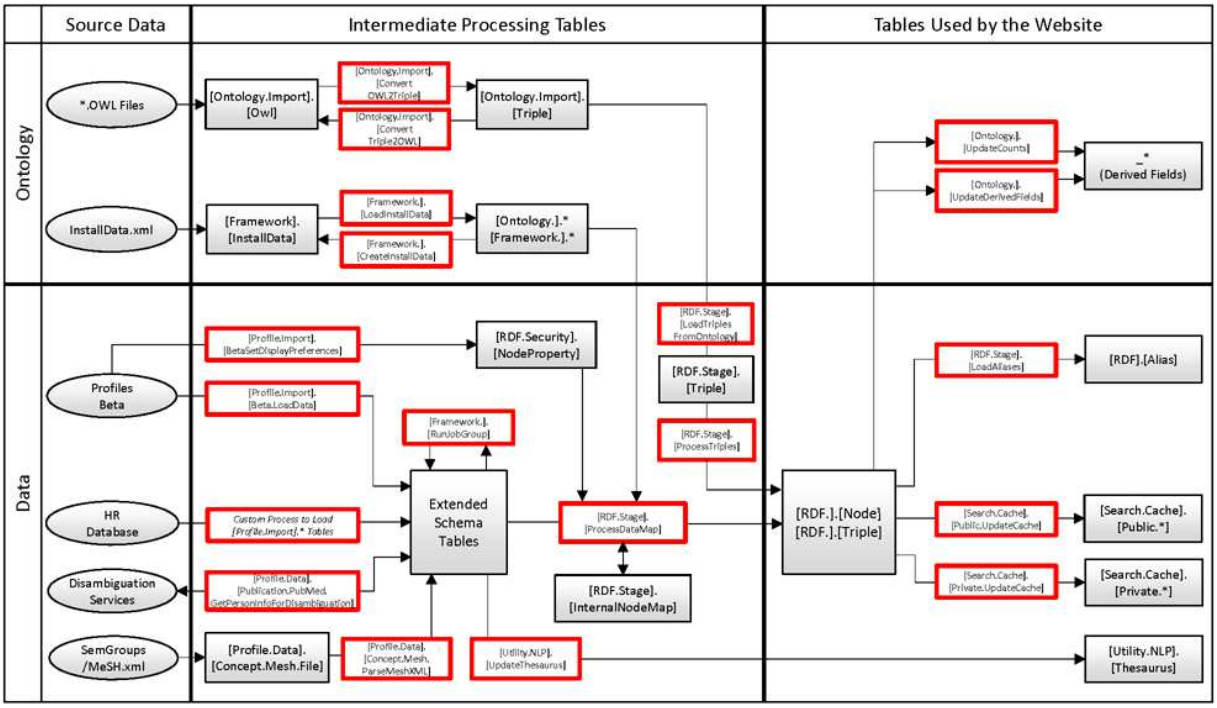
7) Populating the search cache based on the RDF data.

   [RDF.] → [Search.Cache]

   The [Search.Cache].[Public.UpdateCache] and [Search.Cache].[Private.UpdateCache] build the search indexes from the [RDF.] and [Ontology.] tables.

Each of these data load processes get launched by the [Framework.].[RunJobGroup] procedure, which is called by scheduled database jobs that are setup during the Profiles RNS installation. The [Framework.].[RunJobGroup] procedure iterates through the [Framework.].[Job] table to determine the next process to run. A record of what was run is stored in the [Framework.].[Log.Job] table.

The diagram below illustrates selected data feeds (circles), tables (grey boxes), and procedures (red boxes) involved in the installation of Profiles RNS.



Ontology Schema Tables

This section describes the [Ontology.] tables. An important note about these tables is that some field names start with an underscore "_". You should not directly edit the values in the fields.

Their values are set automatically by the stored procedure [Ontology.].[UpdateDerivedFields]. After you make changes to any of the [Ontology.] tables, you should run [Ontology.].[UpdateDerivedFields] to make sure the "_" fields are up-to-date. Also, although it is not required, there is a logical order in which rows in several of the [Ontology.*] tables should be sorted. The stored procedure [Ontology.].[Cleanup] @Action = 'UpdateIDs' will update the sort order.

[Ontology.].[ClassGroup] – ClassGroups group related classes. They are used as the top level of faceting when running a keyword search to narrow results to, for example, people, organizations, or concepts. This table lists the URI for each ClassGroup.

[Ontology.].[ClassGroupClass] – This table lists the classes for each ClassGroup. All classes in Profiles RNS should be assigned to at least one ClassGroupURI. Otherwise, they will not appear properly in search results.

[Ontology.].[PropertyGroup] – PropertyGroups group related properties. They are used to organize content on profile pages. This table lists the URI for each PropertyGroup.

[Ontology.].[PropertyGroupProperty] – This table lists the properties for each PropertyGroup. All properties in Profiles RNS should be assigned to exactly one PropertyGroupURI. Profiles RNS has a default way of displaying and editing all properties on the website. You can override this for a particular property by entering ModuleXML in the CustomDisplayModule or CustomEditModule fields. Placing the ModuleXML in this table affects the property globally. If you have a custom display or edit module for a property that should only be used in the context of a particular class, then use the CustomDisplayModule and CustomEditModule fields in the [Ontology.].[ClassProperty] table.

[Ontology.].[ClassProperty] – This table defines which classes and properties are used by the website. Even if RDF data are loaded into the [RDF.].[Node] and [RDF.].[Triple] tables, they will not be available to the website unless the classes and properties are listed in [Ontology.].[ClassProperty]. (SemWeb/SPARQL is an exception in that it does not use the [Ontology.].[ClassProperty] table.) The three columns, Class, NetworkProperty, and Property uniquely identify each row in this table. There must always be a value for Class and Property, though the NetworkProperty can be null. The NetworkProperty is used for reitifications (triples about other triples) and will be described in more detail in a future version of Profiles RNS. The remaining fields are:

- IsDetail – In Profiles RNS, each URI has short-form and long-form RDF. The short-form includes just the most important properties that need to be returned wherever the item is displayed. For example, in nearly all cases, both the firstName and lastName properties of a person are needed when presenting a list of people. The long-form RDF includes properties that provide additional detail, such as a person's overview text, which typically only appears when viewing that item's full profile page. When IsDetail = 0, the property is always included; when IsDetail = 1, it is only included when the long-form RDF is explicitly requested by setting @showDetails = 1 when calling [RDF.].[GetDataRDF].
- Limit – If a value is provided, then only that number of properties is returned. For example, by default, a person profile only displays the top 5 concepts. The SortOrder field in the [RDF.].[Triple] table is used to select the triples (e.g., "where SortOrder <= Limit").
- IncludeDescription – When calling [RDF.].[GetDataRDF], if @expand = 1, then the RDF for any objects linked to the subject via properties whose IncludeDescription value is 1

will also be returned. This makes it easier to retrieve all the RDF data needed to display a profile. Note that IncludeDescription is recursive—the stored procedure keeps iterating through triples until everything is expanded. As a result, improper use of IsDetail and IncludeDescription can have unexpected consequences, with far more data being processed than needed. As a rule of thumb, start by setting both attributes to 0, and only change them to 1 when absolutely necessary.

- IncludeNetwork - When calling [RDF.].[GetDataRDF] with @showDetails = 1, the RDF will include summary statistics for properties with IncludeNetwork = 1. These statistics are calculated on-the-fly, which can affect performance. So, minimize the use of this feature.
- SearchWeight – This is a number between 0 and 1, which indicates how relevant a property is for search results. Generally, the rdfs:label property is one of the most important properties, and its SearchWeight value is set close to 1. If the SearchWeight is 0, then the property is not considered by the search algorithm.
- CustomDisplay – If set to 1, then Profiles RNS will not display this property on the website by default. A custom module must be defined somewhere to display it.
- CustomEdit – If set to 1, then Profiles RNS will not give the user access to the default DataType and ObjectType property editing tools. A custom module must be defined somewhere in order to enable editing.
- ViewSecurityGroup – This value determines which types of people can view this property. For example, some properties might only be visible to users who have logged into the website. See the section on Security Groups for more information.
- EditSecurityGroup – This value determines which types of people can edit this property. If, for example, the EditSecurityGroup = -50, then a person who is editing her own profile will not see the property listed on the Edit Menu page unless she is also an administrator. The additional Edit*SecurityGroup fields determine the level of access for specific edit features/operations.
- EditPermissionsSecurityGroup – This is the SecurityGroup needed to change the ViewSecurityGroup of an item.
- EditExistingSecurityGroup – This is the SecurityGroup needed to edit an existing triple.
- EditAddNewSecurityGroup – This is the SecurityGroup needed to add a new triple. In the case of a DataType property, this creates a new triple whose object is a literal. In the case of an ObjectType property, this creates a new triple whose object is a new entity.
- EditExistingSecurityGroup – This is the SecurityGroup needed to add a new triple, using an ObjectType property, where the entity already exists. For example, this allows a user to link herself to an organization that has already been given a URI. In contrast, EditAddNewSecurityGroup allows a user to define a new organization.
- EditDeleteSecurityGroup – This is the SecurityGroup needed to delete an existing triple.
- MinCardinality – This is the minimum required number of triples of the particular property. A user cannot delete a triple if fewer than MinCardinality would remain of that property. [This is not fully implemented in the current version of Profiles RNS.]
- MaxCardinality – This is the maximum number of triples of the particular property that can be added to a property. [This is not fully implemented in the current version of Profiles RNS.]
- CustomDisplayModule – Profiles RNS has a default way of displaying properties on the website. You can override this for a particular class-property by entering ModuleXML in the CustomDisplayModule field. CustomDisplay must also be 1 for the CustomDisplayModule to be used.
- CustomEditModule – Profiles RNS has a default way of editing properties. You can override this for a particular class-property by entering ModuleXML in the

CustomEditModule field. CustomEdit must also be 1 for the CustomEditModule to be used.

[Ontology.].[ClassTreeDepth] – This table is automatically derived from the ontology using Subclass relationships.

[Ontology.].[Namespace] – This table lists the namespaces used by Profiles RNS. If you import a custom OWL file using your own namespace, it must be added to this table.

[Ontology.].[DataMap] – This provides mappings from database objects to classes and properties in the ontology. It is used by [RDF.Stage].[ProcessDataMap] as part of scheduled jobs to update the [RDF.].[Node] and [RDF.].[Triple] tables. See the section on Loading Data Using ProcessDataMap for more information.

Loading Data Using ProcessDataMap

The stored procedure [RDF.Stage].[ProcessDataMap] uses the mappings in the table [Ontology.].[DataMap] to copy data from various tables and views into the [RDF.].[Node] and [RDF.].[Triple] table. This is designed to happen on a scheduled basis (typically nightly) to keep data in synch. While the [Ontology.].[DataMap] table provides a semantic mapping (column name to class or property), the [RDF.Stage].[InternalNodeMap] table maps actual IDs. For example, the [Ontology.].[DataMap] table has the following record

Class = 'http://vivoweb.org/ontology/core#Address'
NetworkProperty = NULL
Property = NULL
MapTable = '[Profile.Data].[Person]'
sInternalType = 'Person'
sInternalID = 'PersonID'

Because NetworkProperty and Property are NULL, this is defining a class, rather than the property of a class. In this case, the class is vivo:Address. Note that even though the VIVO ontology treats address as a class, Profiles RNS does not have a separate Address table. When importing data, a person's address is stored in additional columns in the [Profile.Data].[Person] table (MapTable). The unique identifier in the [Profile.Data].[Person] table is "PersonID" (sInternalID), which is used in that table as a "Person" identifier (sInternalType).

In this example, the [RDF.Stage].[ProcessDataMap] procedure will create a new RDF node for each record in [Profile.Data].[Person]. Also, for each record in [Profile.Data].[Person], it will add a record to [RDF.Stage].[InternalNodeMap] with the following values:

Class = 'http://vivoweb.org/ontology/core#Address'
InternalType = 'Person'
InternalID = *the PersonID that uniquely identifies the address*
NodeID = *the NodeID that was created for that PersonID*

The next time [RDF.Stage].[ProcessDataMap] is run, a new node will not be created for a previously processed address since [RDF.Stage].[InternalNodeMap] will already contain the mapping. Because the NodeID determines the URI, this ensures that the URI for the address does not change.

Consider a different example, where nodes are being created for people. The records that are added to [RDF.Stage].[InternalNodeMap] look like:

Class = 'http://xmlns.com/foaf/0.1/Person'
InternalType = 'Person'
InternalID = *the PersonID that uniquely identifies the person*
NodeID = *the NodeID that was created for that PersonID*

As with addresses, the InternalType is “Person” and the InternalID is a “PersonID”. However, in this case, the Class is “http://xmlns.com/foaf/0.1/Person”. The combination Class, InternalType, and InternalID is unique to each row in the [RDF.Stage].[InternalNodeMap] table.

Another example from [Ontology.].[DataMap] is

Class = ‘http://www.w3.org/2004/02/skos/core#Concept’
NetworkProperty = NULL
Property = ‘http://www.w3.org/2000/01/rdf-schema#label’
MapTable = '[profile.data].[concept.mesh.descriptor]'
sInternalType = 'MeshDescriptor'
sInternalID = 'DescriptorUI'
oValue = ‘DescriptorName’
oObjectType = 1

In this case, a property is defined. As a result, instead of creating a node, this creates a triple in the [RDF.].[Triple] table. (An [RDF.].[Node] record might also be created if the object of the triple is a literal that does not yet have a corresponding node.) This example maps the DescriptorName field in the [profile.data].[concept.mesh.descriptor] table to the rdfs:label property of entities of type skco:Concept. The DescriptorUI is the unique identifier for the concepts listed in the MapTable. The oObjectType = 1 indicates that the DescriptorName is a literal value.

When oObjectType = 0, the oValue is the URI of an entity, rather than a literal. Instead of specifying a URI in the oValue column, the URI can be derived from the [RDF.Stage].[InternalNodeMap] table by providing an oClass, oInternalType, and oInternalID.

The DataMapGroup value in the [Ontology.].[DataMap] table indicates records that should be processed together as a batch. If the IsAutoFeed column is set to 0, then that record will be ignored during scheduled jobs. Records whose IsAutoFeed = 0 are typically for one-time data imports.

The Graph value in the [Ontology.].[DataMap] table is copied into the Graph column of [RDF.].[Triple] when a new triple is created. This can be used to trace the triple back to the ProcessDataMap process.

The oStartDate, oStartDatePrecision, oEndDate, and oEndDatePrecision columns in the [Ontology.].[DataMap] table will be used in a future version of Profiles RNS.

Loading Data Using ProcessTriples

If you would like to import a data feed containing data in triples (i.e. subject, predicate, object), you can use the [RDF.Stage].[ProcessTriples] stored procedure. There are several different options for how the subject, predicate, and object can be defined. These are described in the comments of the [RDF.Stage].[ProcessTriples] stored procedure, though not every method is fully implemented in this version of Profiles RNS. Typically, ProcessDataMap is the better way of loading data.

Extending Profiles RNS

To illustrate the process of extending the Profiles RNS ontology and importing data, we will use a brief tutorial/example. Suppose you have a data feed describing the cars people drive to work, which contains three fields: the PersonID, the make and model, and the license plate number. The general steps are:

1) Extend the ontology
    a. Define a namespace
    b. Define the new class in that namespace
    c. Define the new properties in that namespace
2) Import the data feed to an extended schema table
    a. Create a new extended schema table (i.e., [Profile.Data].*)
    b. Load the feed into the new table
3) Create a mapping from the new table to the ontology
4) Run ProcessDataMap to generate RDF

The file SQL_Examples\ProcessDataMap.sql contains additional details and example queries that show how each of these steps work.

When extending Profiles RNS, carefully consider how to use the ontology. Whenever possible, use existing classes and properties. When you create custom ontology extensions, your data will no longer be compatible with other institutions.