Profiles Research Networking Software API Guide

Documentation Version: August 5, 2014

Software Version: ProfilesRNS_2.5.0

Table of Contents

Overview

Profiles RNS consists of three components: 1) a database; 2) Application Programming Interfaces (APIs), which enable computer programs to access the information in the database; and 3) a website, which illustrates one way of calling the APIs and presenting the data. Although the website is the aspect of Profiles RNS that is most visible to users, the APIs have the potential to have far greater impact because they allow any application developer to discover new uses for the data and build new functionality not provided by the website.

Profiles RNS is a Semantic Web application that uses the Resource Description Framework (RDF) data model. In RDF, every entity (e.g., person, publication, concept) is given a unique URI. Entities are linked together using "triples" that contain three URIs--a subject, predicate, and object. For example, the URI of a Person can be connected to the URI of a Concept through a predicate URI of hasResearchArea. An instance of Profiles RNS can have millions of URIs and triples. Semantic Web applications use an ontology, which describes the classes and properties used to define entities and link them together. Profiles RNS uses the VIVO Ontology, which was developed as part of an NIH-funded grant to be a standard for academic and research institutions. A growing number of sites around the world are adopting research networking platforms that use the VIVO Ontology. Because RDF can link different triple-stores that use the same ontology, software developers are able to create tools that span multiple institutions and data sources.

There are four types of APIs in Profiles RNS 1.0. Below are just brief descriptions. The rest of this document provides more detail. Note that the text references files in the API_Examples folder.

1) The old XML based web services from Profiles RNS Beta. This provides backwards compatibility for institutions that built applications using the older version of Profiles RNS. These web services do not take advantage of many of the new features of Profiles RNS 1.0. Users are encouraged to switch to one of the new APIs.

2) RDF crawl. Profiles RNS 1.0 uses Semantic Web technologies (described below). Basically, this means that for every HTML page in the Profiles RNS website, there is a corresponding RDF document, which contains the data for that page in RDF/XML format. Web crawlers can follow the links embedded within the RDF/XML to access additional content.

3) SPARQL endpoint. SPARQL is a programming language that enables arbitrary queries against RDF data. This provides the most flexibility in accessing data; however, the downsides are the complexity in coding SPARQL queries and performance.

4) Profiles RNS 1.0 Search API. This is a web service, which provides support for the most common types of queries. It is designed to be easier to use and to offer better performance than SPARQL, but at the expense of fewer options. It enables full-text search across all entity types, faceting, pagination, and sorting options. The request message to the web service is in XML format, but the output is in RDF/XML format.

PersonID, URI, and Aliases

Internally, Profiles RNS keeps track of each person who has a profile by assigning them a unique PersonID value. However, RDF enables the website to create profiles for any type of entity (e.g., a department, center, publication, concept, etc.). In RDF, a URI is the unique identifier for all entities. In Profiles RNS, the format of the URI is

http://[ProfilesRootPath]/profile/NodeID

where NodeID is some integer value. Each person in Profiles RNS has a URI in addition to his or her PersonID. Because the URI is the more general identifier, this should be used in most contexts instead of the PersonID. For backwards compatibility, the old Profiles RNS Beta API uses the PersonID. However, all the newer APIs use URIs instead.

The PersonID is generated by the Profiles RNS software. It is not the same as your institution's person identifier (e.g., Harvard ID), which was part of the data used to load the Profiles RNS database.

Profiles RNS 1.0 has a new Alias feature, which allows alternative URLs to render the same HTML profile page by pulling data from the same URI. This allows "user-friendly" URLs, such as "display/person/gweber" instead of "profile/123456". However, the SPARQL and Profiles RNS 1.0 Search APIs do not recognize the Aliases and require the actual URI.

1) Profiles RNS Beta API

Note: This API is included only for backwards compatibility with older versions of Profiles RNS. Users are encouraged to switch to one of the new APIs.

The Profiles RNS Beta included an API that enabled data to be queried and extracted via an XML-based web service. There are two ways to call this web service. The first returns a list of people who match search criteria. The list includes the Profiles PersonID for each person and summary information, such as their names and affiliations. The second returns the full profile for a single person, including publications and passive networks such as coauthors and similar people. In both cases, the web service is called by sending it a request XML message via HTTP POST. The file Request.xsd is the schema for the request XML message. The same schema is used for both ways of calling the web service. The output of the web service is a response XML message, whose schema is defined in the file Response.xsd. A typical use case for the web service is to request a list of people who match certain search criteria, and then to request the full profiles for a subset of the PersonIDs that are returned by the initial request.

To call the web service, post the request XML message to a URL that has the form:

http://[ProfilesRootPath]/ProfileService.svc/ProfileSearch

1.1) Search for multiple people

The file GetPersonList.xml is an example request XML message that returns a list of people based on search criteria. Query parameters include first and last name, department and institution name, faculty type, keyword search string, and pagination options. The response returns a summary for each person who matches the search criteria. The summary includes the PersonID and a few additional variables such as name and affiliation. In the Connects.Profiles.ProfileService web.config file, if the IsSecure variable is set to false, then only the first 100 matching people will be returned by the API. If it is set to true, then the number of people returned is limited only by MaxRecords parameter in the request XML.

1.2) Request the full profile for a single person

The file GetSinglePerson.xml is an example request XML message that returns the full profile for a given PersonID. The output filters in the request message define which passive networks should be returned. If the Summary attribute is set to true for an output filter, then only the top ranking items in the passive network will be returned.

2) RDF Crawl

The URIs in Profiles RNS have the form

http://[ProfilesRootPath]/profile/NodeID

Following URI/RDF conventions, this URI is simply an identifier. It does not return any content. If you enter the URI into a web browser, you will be redirected either to a URL that returns HTML content or RDF content, depending on the content-type in the request header. This process is called URI resolution. The corresponding HTML and RDF URLs are:

http://[ProfilesRootPath]/display/NodeID

and

http://[ProfilesRootPath]/profile/NodeID/NodeID.rdf

To end-users of the Profiles RNS website, the URI resolution will be seamless, and they will be able to navigate through pages in the same way as they do an ordinary website. They will see "display/NodeID" (or an Alias) in their browser window.

In order to obtain the RDF data for a profile, call its URI, but use a Request Content Type of "application/rdf+xml". This will redirect to the RDF URL and return RDF/XML data rather than rendered HTML. The Response Content Type of the RDF/XML is "application/rdf+xml".

Alternatively, the RDF URL can be called directly without a specified Request Content Type to obtain the same data. However, calling the URI is better since the address of the RDF URL might change over time, while the URI is intended to be permanent. The RDF URL is useful for development/testing purposes because it can be entered into a web browser to take a "quick look" at the data.

Note that the RDF/XML for a profile will contain URIs to other profiles. Applications consuming the RDF data often "crawl" these URIs to obtain additional information about the related profiles.

Profiles RNS includes a feature to reduce the number crawls needed for the most commonly requested types of data. If the request to the URI includes a Header variable named "Expand" with the value of "true", then the RDF/XML returned will include both the data for that URI as well as the data for a subset of linked URIs. For example, for a person, it will return additional information about that individual's positions, publications, awards, etc. other than just their URIs. When the Profiles RNS website renders a profile page, it typically uses Expand = true and avoids having to request any other data.

By default, all properties are returned when calling a URI. However, for many types of entities, there is a small subset of properties that are frequently used. To request just those properties, include a Header variable named "ShowDetails" with the value of "false". This advantage of this method is that it runs faster.

Obtaining RDF data about a profile by directly calling its URI is the replacement for the Profiles RNS Beta API single-person request. Because of the use of URIs instead of PersonIDs, the RDF data for any type of entity, not just people, can be obtained this way.

3) SPARQL

SPARQL (pronounced “sparkle”) stands for Simple Protocol and RDF Query Language. Details of the SPARQL language are beyond the scope of this document, but an introduction and links to additional information are available at:

http://en.wikipedia.org/wiki/SPARQL

SPARQL is similar to SQL for relational databases in that it enables arbitrary queries to the RDF data. However, SPARQL has a different syntax than SQL and better handles the network structure of RDF.

To run a SPARQL query, post the query text to the SPARQL API (aka “endpoint”), which will have the form

http://[ProfilesSPARQLRootPath]/ProfilesSPARQLAPI.svc/Search

The Request Content Type should be “text/xml”. The output of the API will contain a list of URIs or other data specified by the query. Additional data about those URIs can be obtained using the “RDF Crawl” method described above. The Response Content Type of the SPARQL output is “text/xml”.

The file QueryRequest.xsd is the schema for both the SPARQL request and response messages.

The file SPARQLQuery.txt is an example query that returns all RDF triples for all entities of type people whose last name is “Weber”.

4) Profiles RNS 1.0 Search API

This is a single XML based web service, which enables full-text search across all entity types, and provides options for faceting, pagination, and sorting. It is designed to make the most common types of queries easier to create and faster to execute than SPARQL. The general structure of the XML request message is:

```
<SearchOptions>
    <MatchOptions>
        <SearchString ExactMatch="true/false">text</SearchString>
        <ClassGroupURI>URI</ClassGroupURI>
        <ClassURI>URI</ClassURI>
        <SearchFiltersList>
            <SearchFilter IsExclude="0/1" Property="URI" Property2="URI"
                MatchType="Exact/Left">text</SearchFilter>
        </SearchFiltersList>
    </MatchOptions>
    <OutputOptions>
        <Offset>integer</Offset>
        <Limit>integer</Limit>
        <SortByList>
            <SortBy IsDesc="0/1" Property="URI" Property2="URI" Property3="URI" />
        </SortByList>
    </OutputOptions>
</SearchOptions>
```

This gets posted to a URL with the form

http://[ProfilesSearchAPIRootPath]/ProfilesSearchAPI.svc/Search

which returns RDF/XML data. The first rdf:Description tag has rdf:nodeID="SearchResults". The properties of this node describe the search parameters (SearchOptions), the number of matches (prns:numberOfConnections), and a breakdown of the number of matches by class and class group (prns:matchesCalssGroupList). The SearchResults node also has a list of prns:hasConnection properties that point to prns:Connection nodes, which correspond to the individual items matching the query. The RDF/XML returned by the API also contains rdf:Description tags for each of the prns:Connection nodes. Those tags contain properties that describe the search relevance (prns:connectionWeight), sort order (prns:sortOrder), class/type (rdf:type), name (rdfs:label), and the URI of the matching entity (rdf:object).

The file SearchOptions.xsd is the schema for the Search API request message. The Request Content Type should be “text/xml”. The response message is an RDF/XML document. The Response Content Type of the RDF/XML is “application/rdf+xml”.

Note that this Search API provides functionality similar to the Profiles RNS Beta API multiple-person request. A typical use case would be to query the Search API to get a list of URIs that match some search criteria, and then use RDF crawl to obtain detailed data for each of those URIs.

Below is a description of the tags in the request message. All tags and attributes are optional. Tags and attributes whose type is URI must use a full URI (e.g., “http://xmlns.com/foaf/0.1/firstName”), not namespace prefix (e.g., “foaf:firstName”).

Tag[@Attribute]	Type	Description
SearchString	text	This is a list of keywords or quoted phrases that will be used to identify matching nodes. A thesaurus is used to expand certain terms (e.g., “cancer” becomes “cancer OR neoplasm”), stop words (e.g., “the”, “of”, etc.) are removed, and stemming handles different parts of speech (e.g., “cancers” becomes “cancer*”). Microsoft’s SQL Server full-text search then compares the parsed and expanded search string to literal values in the RDF.
SearchString[@ExactMatch]	true/false	If this attribute is “true”, then the exact search string, without any parsing, will be compared to RDF literals.
ClassGroupURI	URI	This limits the search results to a specific group of RDF classes (e.g., "http://profiles.catalyst.harvard.edu/ontology/prns#ClassGroupOrganizations").
ClassURI	URI	This limits the search results to a specific class (e.g., "http://xmlns.com/foaf/0.1/Person").
SearchFilter	text	This limits the search results based on the value of a particular property. For example, if “Griffin” is used as the SearchString, then people with either a first or last name of “Griffin” will be returned. However, if “Griffin” is used as the SearchFilter with Property = “http://xmlns.com/foaf/0.1/firstName”, then the first name must be “Griffin”. Note that multiple SearchFilters can be defined. SearchString, ClassGroupURI/ClassURI, and SearchFilter can be used together, for example, to find people matching “cancer” whose first name is “Griffin”.
SearchFilter[@IsExclude]	0/1	If IsExclude = “1”, then only nodes that do NOT have a property with this value will be returned.
SearchFilter[@Property]	URI	This is the property used for the SearchFilter if Property2 is not defined.
SearchFilter[@Property2]	URI	If Property2 is defined, then the SearchFilter allows item X to be returned in the search results if X is linked through Property to some item Y whose Property2 matches the value of the SearchFilter. For example, if Property = “http://profiles.catalyst.harvard.edu/ontology/prns#personInPrimaryPosition”, and Property2 = “http://vivoweb.org/ontology/core#positionInOrganization” , and the value of the SearchFilter tag is the URI of an organization, then only nodes (e.g., people) whose primary position is at the organization with the specified URI will be returned. Determining the right combination of Property, Property2, and SearchFilter value to use requires a good understanding of the ontology, but it provides a lot of flexibility in doing targeted searches or faceting.
SearchFilter[@MatchType]	Exact/Left	This determines if the SearchFilter value should match the property value exactly (Exact) or as a prefix (Left).
Offset	Integer	This is used for pagination to indicate the starting item to return in the search results. An offset of “0” includes the first matching item. An offset of “1” starts with the second matching item.
Limit	Integer	This is used for pagination to indicate the maximum number of items to return in the search results. An offset

		of “5” and limit of “20” returns items 6 through 25.
SortBy[@IsDesc]	0/1	The SortBy tags indicate which properties should be used to sort the search results. The default is to sort by query relevance. Up to three SortBy tags can be defined. They are applied in the order in which they appear in the SearchOptions XML—the matching items are first sorted by the first SortBy tag, and if there are ties, then the second SortBy tag is applied, etc. The actual sorting uses the value of the SortBy property, not the URI of the property. The IsDesc attribute determines if the sorting is ascending (“0”) or descending (“1”).
SortBy[@Property]	URI	The property of the matching nodes whose value will be used to sort the search results list.
SortBy[@Property2]	URI	Similar to the Property2 attribute of the SearchFilter tag, if a Property2 is defined, then sorting occurs on the value of Property2 of the node connected to the matching node through Property.
SortBy[@Property3]	URI	If a Property3 is defined, then sorting occurs on the value of Property3 of the node connected to some node through Property2, which is connected to the matching node through Property.

The file SearchByKeyword.xml is an example XML request message that returns the 15 nodes most related to “asthma”. The file SearchByKeywordFaceting.xml is an example that returns just the publications related to “asthma”. The file SearchForPeopleOnly.xml is an example that returns the first 15 people whose last name starts with “Smith”, sorted by last name then first name.

Content Type Summary

Each API uses a different Request and Response Content Type. The table below provides a summary.

API	Request Content Type	Response Content Type
Profiles RNS Beta API	text/xml	text/xml
RDF Crawl	application/rdf+xml	application/rdf+xml
SPARQL	text/xml	text/xml
Profiles RNS 1.0 Search API	text/xml	application/rdf+xml

Frequently Used VIVO Classes and Properties

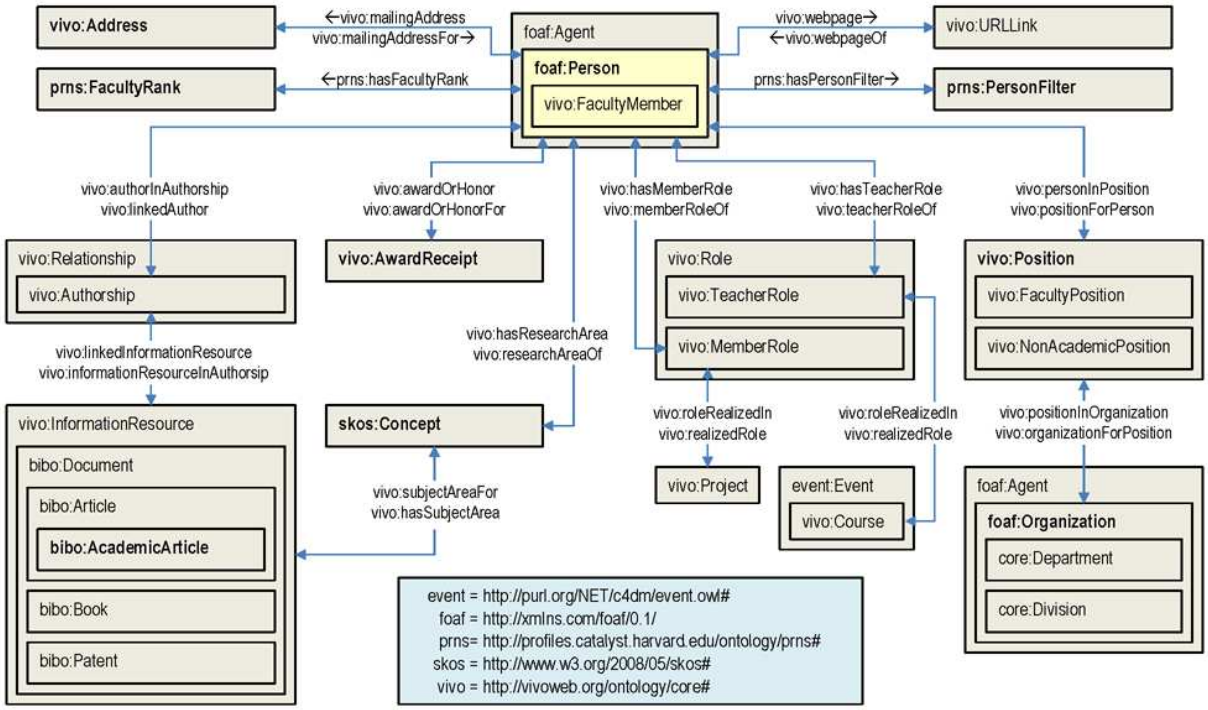
Creating SPARQL queries and taking full advantage of the Search API requires an understanding of the underlying ontology. Profiles RNS uses the VIVO ontology. This is an extensive ontology, with thousands of classes and properties. A full description of this is outside the scope of this document. Unfortunately, there is not a lot of information available about the VIVO ontology. An "OWL" file, which can be viewed by ontology editors such as Protégé can be downloaded from

http://vivoweb.org/download

A detailed diagram illustrating some portions of the ontology can be found at

http://sourceforge.net/apps/mediawiki/vivo/index.php?title=VIVO_1.3_Ontology_overview_diagram

Below is an even further simplified diagram, highlighting selected classes and object properties.



To help you get started, below is a brief summary of the VIVO namespaces, classes, and properties:

The VIVO ontology is actually the combination of many popular Semantic Web ontologies. Each ontology has its own namespace. The table below lists the namespaces and the prefix used in RDF/XML. The most important are (1) rdf, rdfs, and owl, which define generic RDF concepts, (2) foaf (friend-of-a-friend), which describes people, (3) bibo, which describes publications, (4) skco, which describes subject areas and concepts, (5) vivo, which are custom classes and properties created for the VIVO ontology to describe activities in academic institutions, and (6) prns, which are custom classes and properties created for the Profiles RNS software.

Prefix	Namespace
afn	http://jena.hpl.hp.com/ARQ/function#
bibo	http://purl.org/ontology/bibo/
dcelem	http://purl.org/dc/elements/1.1/
dcterms	http://purl.org/dc/terms/
event	http://purl.org/NET/c4dm/event.owl#
foaf	http://xmlns.com/foaf/0.1/
geo	http://aims.fao.org/aos/geopolitical.owl#
obo	http://purl.obolibrary.org/obo/
owl	http://www.w3.org/2002/07/owl#
owl2	http://www.w3.org/2006/12/owl2-xml#
prns	http://profiles.catalyst.harvard.edu/ontology/prns#
pvs	http://vivoweb.org/ontology/provenance-support#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
scirr	http://vivoweb.org/ontology/scientific-research-resource#
skco	http://www.w3.org/2004/02/skos/core#
skos	http://www.w3.org/2008/05/skos#
swvs	http://www.w3.org/2003/06/sw-vocab-status/ns#
vann	http://purl.org/vocab/vann/
vitro	http://vitro.mannlib.cornell.edu/ns/vitro/public#
vitro07	http://vitro.mannlib.cornell.edu/ns/vitro/0.7#
vivo	http://vivoweb.org/ontology/core#
xsd	http://www.w3.org/2001/XMLSchema#

Below are frequently used classes. Note that after the namespace prefix, class names generally start with an upper case letter. RDF is case sensitive.

Class	Description
foaf:Person	A person (faculty, staff, etc.)
foaf:Organization	Any type of organization (institution, hospital, etc.)
vivo:Department	A subclass of foaf:organization
vivo:Division	A subclass of foaf:organization
vivo:InformationResource	Publications, media files, data sets, etc.
bibo:AcademicArticle	A subclass of vivo:informationResource
vivo:Authorship	Connects a foaf:Person to a vivo:informationResource
vivo:Position	Connects a foaf:Person to a foaf:Organization
skco:Concept	A subject area or concept
vivo:AwardReceipt	An award given to an individual
prns:FacultyRank	An academic rank (e.g., Full Professor)

Below are frequently used properties. Note that after the namespace prefix, property names generally start with a lower case letter.

Property	Description
rdf:type	The class of a URI
rdfs:label	The default name of a URI
foaf:firstName	A person's first name
foaf:lastName	A person's last name
vivo:overview	A description of the URI (e.g., a person's research narrative)
vivo:awardOrHonor	Links a foaf:Person to a vivo:AwardReceipt
vivo:hasResearchArea	Links a foaf:Person to a skco:Concept

vivo:authorInAuthorship	Links a foaf:Person to a vivo:Authorship
vivo:linkedInformationResource	Links a vivo:Authorship to a vivo:InformationResource
vivo:personInPosition	Links a foaf:Person to a vivo:Position
vivo:positionInOrganization	Links a vivo:Position to a foaf:Organization

Note that people are not directly linked to publications or organizations; the links go through the classes vivo:Authorship and vivo:Position. Specifically:

foaf:Person → vivo:authorInAuthorship → vivo:Authorship → vivo:linkedInformationResource → vivo:InformationResource

foaf:Person → vivo:personInPosition → vivo:Position → vivo:positionInOrganization → foaf:Organization